Logic-Based Discrete-Steepest Descent: A Solution Method for Process Synthesis Generalized Disjunctive Programs

A Preprint

Daniel Ovalle[1,2], David A. Liñán [3], Albert Lee [4], Jorge M. Gómez[2], Luis Ricardez-Sandoval[3], Ignacio E. Grossmann[1], and David E. Bernal Neira [1,4,5,6,*]

[1] *Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA*
[2] *Department of Chemical and Food Engineering, Universidad de los Andes, Bogotá 1100123, Colombia*
[3] *Department of Chemical Engineering, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1*
[4] *Davidson School of Chemical Engineering, Purdue University, West Lafayette, IN, USA*
[5] *Quantum Artificial Intelligence Laboratory, NASA Ames Research Center, Mountain View, CA, USA*
[6] *Research Institute of Advanced Computer Science, Universities Space Research Association, Mountain View, CA, USA*
[*] *Corresponding author: dbernaln@purdue.edu*

November 13, 2024

Abstract

~~The optimization~~ Optimization of chemical processes is challenging due ~~to the~~ nonlinearities arising from ~~process physics~~ chemical principles and discrete design decisions. ~~In particular,~~ The optimal synthesis and design of chemical processes can be posed as a Generalized Disjunctive Programming (GDP) ~~superstructure problem. Various solution methods are available to address these problems , such as reformulating them as~~ problem. While reformulating GDP problems as Mixed-Integer Nonlinear Programming (MINLP) problems ~~; nevertheless, algorithms explicitly designed to solve the GDP problem and potentially leverage its structure~~ is common, specialized algorithms for GDP remain scarce. This ~~paper presents the Logic-based~~ study introduces the Logic-Based Discrete-Steepest Descent Algorithm (LD-SDA) as a solution method for GDP problems involving ordered Boolean variables. ~~The~~ LD-SDA ~~reformulates these ordered Boolean variables into integer decisions called external variables. The LD-SDA solves the reformulated GDP problem using~~ transforms these variables into external integer decisions and uses a two-level decomposition~~approach where~~ : the upper-level ~~subproblem determines external variable configurations. Subsequently, the remaining continuous and discrete variablesare solved as a subproblem only involving those constraints relevant to the given external variable arrangement, effectively taking advantage of the structure of the GDP problem. The advantages of~~ sets external configurations, and the lower-level solves the remaining variables, efficiently exploiting the GDP structure. In the case studies presented in this work, including batch processing, reactor superstructures, and distillation columns, LD-SDA ~~are illustrated through a batch processingcase study, a reactor superstructure, a distillation column, and a catalytic distillation column, and its open-source implementation is available online. The results show convergence efficiency and solution quality improvements compared to~~ consistently outperforms conventional GDP and MINLP solvers, especially as problem size grows, and proves superior in challenging problems where other solvers encounter difficulties in finding optimal solutions.

Keywords Superstructure Optimization · Optimal Process Design · Generalized Disjunctive Programming · MINLP · Process Intensification

1 Introduction

The ongoing research in modeling and optimization provides computational strategies to enhance the efficiency of chemical processes across various time scales, e.g., design, control, planning, and scheduling [1, 2]. ~~In addition, optimization~~ Optimization tools help develop novel processes and products that align with environmental, safety, and economic standards, thus promoting competitiveness. Despite advances in the field, ~~the deterministic solution of~~ deterministically solving optimization problems that ~~include discrete decisions together with nonlinearities is still challenging. For instance,~~ involve discrete decisions alongside nonlinearities remains a significant challenge.

A key application where these challenges arise is the optimal synthesis and design of reactor and separation processes~~must incorporate~~ , which requires incorporating discrete decisions to ~~decide~~ determine the arrangement and sizes of distillation sequences and reactors, as well as the non-ideal relationships ~~required~~ needed to model vapor-liquid phase equilibrium. The ~~interactions~~ interaction between nonlinear models and discrete decisions in this problem ~~introduce~~ introduces nonconvexities and numerical difficulties (e.g., zero-flows of inactive stages/units), ~~which complicates the direct solution of~~ that complicate directly solving these problems with ~~the~~ traditional optimization solvers [3, 4]. ~~The computational burden of these problems constitutes another significant limitation, impeding timely solutions, particularly in online applications or large-scale systems. For instance, one of the main limitations in implementing~~ Another important area where these challenges become evident is in economic nonlinear model predictive control~~is~~ , where solving optimization problems within the ~~sampling time of the controller~~ controller sampling time presents a significant obstacle [5]. This ~~issue aggravates when coupling control with~~ difficulty increases when design or scheduling decisions ~~, which adds discrete decisions into~~ are coupled with control, adding discrete decisions to the formulation [2]. ~~Given the above, there remains a~~

The computational burden of these problems presents a significant limitation, often preventing the ability to find timely solutions, particularly in online applications or large-scale systems. This challenge highlights the need for advanced optimization algorithms capable of efficiently exploring the search space of discrete variables ~~and handling~~ while managing nonlinear discrete-continuous ~~variable interactions to tackle~~ interactions to address relevant chemical engineering optimization problems.

~~Two of the main~~ Two major modeling approaches that ~~incorporate~~ address these issues by incorporating discrete decisions and ~~activate or deactivate~~ activating or deactivating groups of nonlinear constraints ~~in the formulation~~ are Mixed-Integer Nonlinear Programming (MINLP) and Generalized Disjunctive Programming (GDP).

Typically, optimization problems are posed using MINLP formulations ~~, which~~ that incorporate both continuous variables, here denoted as ~~$\mathbf{x} = [x_1, \dots, x_{n_x}]$~~ $\mathbf{x} = (x_1, \dots, x_{n_x})$, and discrete variables, here denoted as ~~$\mathbf{z} = [z_1, \dots, z_{n_z}]$~~ $\mathbf{z} = (z_1, \dots, z_{n_z})$. The resulting optimization problems involve the minimization of a function $f : \mathbb{R}^{n_x} \times \mathbb{Z}^{n_z} \to \mathbb{R}$ subject to nonlinear inequality constraints $\mathbf{g} : \mathbb{R}^{n_x} \times \mathbb{Z}^{n_z} \to \mathbb{R}^l$. ~~The variables~~ Variables are usually considered to ~~be bounded, meaning they~~ belong to a closed set (referred to as bounded) $\mathbf{x} \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}]$ and $\mathbf{z} \in \{\underline{\mathbf{z}}, \dots, \overline{\mathbf{z}}\}$, respectively. Throughout this paper, for a variable x, we use underbar $\underline{x}$ and overbar $\overline{x}$ notation to denote lower and upper bounds, respectively. The mathematical formulation of an MINLP is as follows:

$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{z}} \; & f(\mathbf{x}, \mathbf{z}) \\
\text{s.t. } & \mathbf{g}(\mathbf{x}, \mathbf{z}) \leq \mathbf{0} \\
& \mathbf{x} \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}] \subseteq \mathbb{R}^{n_x}; \mathbf{z} \in \{\underline{\mathbf{z}}, \dots, \overline{\mathbf{z}}\} \subseteq \mathbb{Z}^{n_z}
\end{aligned} \tag{MINLP}
$$

Problem (MINLP) belongs to the NP-hard complexity class [6]~~, nevertheless~~ . Despite the challenges associated with solving NP-hard problems, efficient MINLP solution algorithms have been developed, motivated by its various applications [7]. These algorithms ~~take advantage of the discrete nature of variables $\mathbf{z}$ to explore the feasible set of (MINLP) to find the~~ leverage both discrete and continuous information to find a feasible optimal solution$(\mathbf{x}^*, \mathbf{z}^*)$. Among the most ~~common approaches for finding deterministic solutions to (MINLP), there exist methods based either~~ commonly used approaches to solve (MINLP) deterministically are methods based on decomposition or ~~on~~ branch-and-bound (BB) [8]. These techniques separately address the two ~~sources of hardness of the problem (MINLP), i.e.,~~ main sources of difficulty in (MINLP): the discreteness of $\mathbf{z}$ and the nonlinearity of $\mathbf{g}$. ~~Both of these methods~~

Both decomposition and branch-and-bound rely on bounding the optimal objective ~~function $f(\mathbf{x}^*, \mathbf{z}^*) = f^*$~~ value $f(\mathbf{x}^*, \mathbf{z}^*)$. This involves searching for values $(\hat{f}, \tilde{f})$ such that $\hat{f} \leq f^* \leq \tilde{f}$, and progressively tighten them. ~~The optimal solution is bounded from above by finding~~ We denote f^* as an optimal solution value, meaning that $f(\mathbf{x}^*, \mathbf{z}^*) = f^*$ if the $(\mathbf{x}^*, \mathbf{z}^*)$ is an optimal solution. An upper bound on an optimal solution value is found by identifying feasible solutions to the problem $\{(\tilde{\mathbf{x}}, \tilde{\mathbf{z}}) \mid \mathbf{g}(\tilde{\mathbf{x}}, \tilde{\mathbf{z}}) \leq 0, \tilde{\mathbf{x}} \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}], \tilde{\mathbf{z}} \in \{\underline{\mathbf{z}}, \dots, \overline{\mathbf{z}}\}\}$, i.e., $f(\mathbf{x}^*, \mathbf{z}^*) \leq f(\tilde{\mathbf{x}}, \tilde{\mathbf{z}}) = \tilde{f}$. ~~The relaxations~~ Relaxations of problem (MINLP), which are optimization problems defined over a larger feasible set, have an optimal solution $\hat{f}$ that is guaranteed to underestimate ~~the~~ an optimal objective value of the original problem i.e., $\hat{f} \leq f^*$.

The second modeling approach used in the literature is GDP, which generalizes the problem in (MINLP) by introducing Boolean variables $\mathbf{Y}$ and disjunctions ~~[·]~~ into the formulation [9]. In GDP, the Boolean variable Y_{ik} indicates whether a set of constraints $\mathbf{h}_{ik}(\mathbf{x}, \mathbf{z}) \leq \mathbf{0}$ is enforced or not. We refer to this enforcing ~~alternative as~~ condition as a *disjunct* i, in *disjunction* k. Only one disjunct per disjunction is to be selected; hence, we relate disjunctions with an exclusive OR (XOR $\veebar$) operator , which can be interpreted as an $\texttt{Exactly}(1, \cdot)$ operator when ~~$|D_k| > 2$~~ there are more than two disjuncts in a disjunction [10]. Boolean variables can be related through a set of logical propositions $\Omega(\mathbf{Y}) = True$ by associating them through the operators AND ($\wedge$), OR($\vee$), XOR ($\veebar$), negation ($\neg$), implication ($\Rightarrow$) and equivalence ($\Leftrightarrow$). Furthermore, GDP considers a set of *global* constraints $\mathbf{g}(\mathbf{x}, \mathbf{z}) \leq 0$ existing outside the disjunctions, which are enforced regardless of the values of the Boolean variables. The mathematical formulation for GDP is as follows:

$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{Y},\mathbf{z}} \; & f(\mathbf{x}, \mathbf{z}) \\
\text{s.t. } & \mathbf{g}(\mathbf{x}, \mathbf{z}) \leq \mathbf{0} \\
& \Omega(\mathbf{Y}) = True \\
& \bigvee_{i \in D_k} \begin{bmatrix} Y_{ik} \\ \mathbf{h}_{ik}(\mathbf{x}, \mathbf{z}) \leq \mathbf{0} \end{bmatrix} \quad k \in K \\
& \mathbf{x} \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}] \subseteq \mathbb{R}^{n_x}; \mathbf{Y} \in \{False, True\}^{n_y}; \mathbf{z} \in \{\underline{\mathbf{z}}, \ldots, \overline{\mathbf{z}}\} \subseteq \mathbb{Z}^{n_z}
\end{aligned}
\tag{GDP}
$$

where $n_y = \sum_{k \in K} |D_k|$. Moreover, Boolean variables may be associated with empty disjunctions and still appear in the logical propositions $\Omega(\mathbf{Y})$ to model complex logic that does not involve a set of constraints ~~$\mathbf{h}(\mathbf{x}, \mathbf{z}) \leq \mathbf{0}$~~ $\mathbf{h}_{ik}(\mathbf{x}, \mathbf{z}) \leq \mathbf{0}$.

Throughout this work, we make several important assumptions: the problem (GDP) has at least one feasible solution, the search space for continuous, integer, and Boolean variables is bounded, and the objective function remains bounded as well. Additionally, the main problem and the subproblems obtained by fixing Boolean configurations satisfy the necessary conditions for standard Nonlinear Programming (NLP) and MINLP algorithms to find a solution. Specifically, the functions f, $\mathbf{g}$, and $\mathbf{h}_{ik}$ are assumed to be smooth, with available first and second derivatives when solving NLP subproblems.

Different strategies are available to solve problem (GDP). The traditional approach is to reformulate the problem into an MINLP, and the two classic reformulations are the big-M reformulation (BM) [11, 12] and the hull or extended reformulation (HR) [13, 14]. However, there ~~exist~~ are algorithms specifically designed for the GDP framework that exploit the intrinsic logic of the problem. These tailored algorithms include logic-based outer approximation (LOA) [15] and logic-based ~~branch and bound~~ branch-and-bound (LBB) [16].

The GDP framework has recently been used in the optimization of chemical processes. Some modern applications in process design include co-production plants of ethylene and propylene [17], reaction-separation processes [18], and once-through multistage flash process [19]. Other advances in process synthesis include effective modular process [20], refrigeration systems [21], and optimization of triple pressure combined cycle power plants [22]. Recently, new solvent-based adhesive products [23] and optimal mixtures [24, 25] have been designed using this methodology. ~~Scheduling of multiproduct~~ The scheduling of multi-product batch production [26], blending operations [27], refineries [28, 29], modeling of waste management in supply chains [30], and multi-period production planning [31, 32] are some modern applications of the GDP framework in planning and scheduling. We refer the reader to the review by Trespalacios and Grossmann [9] for other developments in GDP applications.

~~A common feature in many applicationsis that~~ In many applications, Boolean variables and disjunctions in GDP formulations often represent discrete decisions with intrinsic ordering. Examples of these ordered decisions include selecting discrete locations (e.g., feed location in a distillation superstructure), determining discrete points in time (such as the starting date of a task in scheduling), or ~~integer numbers~~ specifying integer values (as seen in the number of units in a design problem, either in parallel or series). A key characteristic of these problems is that increasing or decreasing the value of ~~those~~ these discrete decisions implies an ordered inclusion or exclusion of nonlinear equations from the model. However, ~~Boolean variables that model these decisions in the (GDP) problem do not usually consider~~ the Boolean variables used in (GDP) typically do not capture this ordered structure, failing to ~~capture and leverage~~ leverage the potential relationships between ~~subsequent~~ successive sets of constraints.

To exploit ~~this structure~~ the ordered structure in optimization problems, a solution strategy was recently ~~proposed~~ introduced in the mixed-integer context to ~~efficiently solve MINLP~~ solve MINLP superstructure optimization problems ~~. Here, the~~ more efficiently. In this approach, ordered binary variables are reformulated ~~into~~ as discrete variables (called *external variables*) to ~~account for~~ explicitly represent their ordered structure ~~~~[33]. The ~~solution strategy lifts these integer external variables to an upper-layer problem~~ reformulated problem is then lifted to an upper-level optimization, where a Discrete-Steepest Descent Algorithm (D-SDA) is applied. This algorithm is ~~theoretically supported by the principles of~~ based on principles from discrete convex analysis, which ~~establishes~~ provides a different theoretical ~~framework~~ foundation for discrete optimization [34]. ~~The~~

D-SDA ~~was applied as an MINLP algorithm to the~~ has demonstrated its effectiveness in several MINLP applications, including optimal design of equilibrium ~~~~[35] and rate-based catalytic distillation columns [36]~~and~~. It proved to be more efficient than state-of-the-art MINLP solvers in ~~terms of~~ both computational time and solution quality. ~~The first computational experiments that showed the application of~~ Early computational experiments applying D-SDA as a logic-based solver for GDP ~~applications also showed~~ formulations also showed promising improvements in solution quality and computational ~~time when applied to case studies involving the design of a reactor network , the designof a~~ efficiency. These experiments involved case studies such as reactor network design, rate-based catalytic distillation column ~~, and the~~ design, and simultaneous scheduling and dynamic optimization of network batch processes [37, 38].

This paper presents the logic-based D-SDA (LD-SDA) as a logic-based solution approach specifically designed for GDP problems whose Boolean or integer variables follow an ordered structure. Our work builds on our previous work in [37] and provides new information on the theoretical properties and details of the computational implementation of the LD-SDA as a GDP solver. The LD-SDA uses optimality termination criteria derived from discrete convex analysis [34, 39] that allow the algorithm to find local optima not necessarily considered by other MINLP and GDP solution algorithms. This study also presents new computational experiments that showcase the performance of the LD-SDA compared to the standard MINLP and GDP solution techniques. The novelties of this work can be summarized as follows:

- A generalized version of the external variables reformulation applied to GDP problems is presented, thus extending this reformulation from MINLP to a general class of GDP problems.
- The proposed framework is more general than previous MINLP approaches, allowing the algorithm to tackle a broader scope of problems. Through GDP, the subproblems can be either NLP, MINLP, or GDP, instead of the previous framework where only NLP subproblems were supported.
- An improved algorithm that uses external variable bound verification, fixed external variable feasibility via ~~Feasibility-based~~ Feasibility-Based Bounds Tightening (FBBT), ~~globally visited set verification~~ verification of already visited configurations, and a reinitialization scheme to improve overall computational time.
- The open-source implementation of the algorithm is generalized for any GDP problem, leading to an automated methodology. Before executing the LD-SDA, the user only needs to identify the variables to be reformulated ~~into~~ as external variables and the constraints that relate them to the problem. This implementation, formulated in Python, is based on the open-source algebraic modeling language Pyomo [40] and its Pyomo.GDP extension [41], and ~~it~~ can be found in an openly available GitHub repository[1].

The remainder of this work is organized as follows. §2 presents a general background in both ~~solution techniques for GDP~~ GDP solution techniques and in discrete-steepest optimization through discrete convex analysis. §3 illustrates the external variable reformulation for Boolean variables. Furthermore, this section formally describes the LD-SDA and discusses relevant properties and theoretical implications. The ~~implementation details and~~ details of the implementation and the algorithmic enhancements are described in §4. Numerical experiments were conducted to assess the performance of the LD-SDA across various test cases, including reactor networks, batch process design, and distillation columns with and without catalytic stages. The ~~outcomes~~ results of these experiments are detailed in §5. Finally, the conclusions of the work along with future research directions are stated in §6.

2 Background

This section serves two primary objectives. Firstly, it provides an introduction to the solution methods employed in Generalized Disjunctive Programming~~(GDP)~~. Secondly, it describes the Discrete-Steepest Descent Algorithm ~~(D-SDA)~~ along with its underlying theoretical framework, discrete convex analysis.

2.1 Generalized Disjunctive Programming Reformulations Into MINLP

A GDP can be reformulated into ~~a~~ an MINLP, enabling the use of specialized codes or solvers that have been developed for MINLP problems [42, 14]. In general, the reformulation is done by transforming the logical constraints into algebraic

[1] https://github.com/SECQUOIA/dsda-gdp

constraints and Boolean variables into binary variables [9]. Moreover, MINLP reformulations handle disjunctions by introducing binary decision variables $\mathbf{y} \in \{0,1\}^{n_y}$, instead of Boolean ($False$ or $True$) variables $\mathbf{Y}$. The exclusivity requirement of disjunctions is rewritten as the sum of binary variables adding to one, thus implying that only a single binary variable can be active for every disjunction $k \in K$.

$$
\begin{aligned}
\mathbf{Y} \in \{False, True\}^{n_y} &\rightarrow \mathbf{y} \in \{0,1\}^{n_y} \\
\Omega(\mathbf{Y}) = True &\rightarrow \mathbf{A}\mathbf{y} \geq \mathbf{a} \\
\bigvee_{i \in D_k} [Y_{ik}] \Leftrightarrow \texttt{Exactly}(1, [Y_{ik}\ i \in D_k]) &\rightarrow \sum_{i \in D_k} y_{ik} = 1
\end{aligned}
\tag{GDP-MINLP}
$$

In this section, we describe the two most common approaches to transform a GDP into a MINLP, namely the big-M (BM) and the hull reformulations (HR). Different approaches to reformulating the disjunctions of the GDP into MINLP result in diverse formulations. These formulations, in turn, yield distinct implications for specific ~~problem-solving~~problem solving [43].

The big-M reformulation uses a large positive constant M ~~in an inequality such that it renders the constraint nonbinding or redundant depending~~ into the inequalities to either activate or relax constraints based on the values of the binary variables. ~~The status of constraints (~~A constraint being active or redundant ~~) depends~~ is dependent on the values taken by their corresponding binary variables~~, that is , a~~. When a binary variable is $True$, the corresponding vector of constraints $\mathbf{h}_{ik}(\mathbf{x},\mathbf{z}) \leq \mathbf{0}$ is ~~activated when Y_{ik} is $True$~~enforced. Otherwise, the right-hand side is relaxed by the large value M such that the constraint is satisfied irrespective of the values of $\mathbf{x}$ and $\mathbf{z}$, effectively ~~ignoring the constraint~~ making the constraint nonbinding. This behaviour can be expressed as $\mathbf{h}_{ik}(\mathbf{x},\mathbf{z}) \leq M(1-y_{ik})$ where y_{ik} is a binary variable replacing Y_{ik}. The resulting GDP-transformed MINLP using (BM) is given by:

$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{y},\mathbf{z}}\ & f(\mathbf{x},\mathbf{z}) \\
\text{s.t. } & \mathbf{g}(\mathbf{x},\mathbf{z}) \leq \mathbf{0} \\
& \mathbf{A}\mathbf{y} \geq \mathbf{a} \\
& \sum_{i \in D_k} y_{ik} = 1 \quad k \in K \\
& \mathbf{h}_{ik}(\mathbf{x},\mathbf{z}) \leq M_{ik}(1-y_{ik}) \quad i \in D_k, k \in K \\
& \mathbf{x} \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}] \subseteq \mathbb{R}^{n_x}; \mathbf{y} \in \{0,1\}^{n_y}; \mathbf{z} \in \{\underline{\mathbf{z}}, \dots, \overline{\mathbf{z}}\} \subseteq \mathbb{Z}^{n_z}
\end{aligned}
\tag{BM}
$$

The hull reformulation (HR) ~~uses binary variables to handle disjunctive inequalities . However, this method disaggregates the~~ offers an alternative by handling disjunctive inequalities using binary variables, but it takes a different approach by disaggregating both continuous and discrete variables~~, and a copy~~. For each disjunct in the GDP, a copy of each variable $v_{ik} \in [\underline{x}, \overline{x}]$ or $w_{ik} \in \{\underline{z}, \dots, \overline{z}\}$ ~~of each variable is added~~ is created for each element i in the disjunction D_k. ~~By becoming zero when their~~ When the corresponding binary variable is ~~0~~set to zero, these new variables ~~enforce the constraintsdepending on which binary variable is 1. Only copies corresponding~~ become zero as well, effectively deactivating their associated constraints. Conversely, only the copies of variables linked to binary variables equal to ~~1~~ one are involved in ~~their corresponding constraints. Furthermore~~enforcing the constraints. Additionally, the constraints in each disjunct are ~~enforced through~~ governed by the binary variables ~~by their perspective reformulation evaluated over~~ through a perspective reformulation applied to the disaggregated variables~~, that is,~~. Specifically, each disjunct that activates a ~~vector~~ set of constraints $\mathbf{h}_{ik}(\mathbf{x},\mathbf{z}) \leq 0$ is reformulated as $y_{ik}\mathbf{h}_{ik}(\mathbf{v}_{ik}/y_{ik}, \mathbf{w}_{ik}/y_{ik}) \leq 0$, where y_{ik} is a binary variable ~~replacing~~ that replaces Y_{ik}.

The difficulty in applying the HR to a GDP is that the perspective function $y_{ik}\mathbf{h}_{ik}(\mathbf{v}_{ik}/y_{ik}, \mathbf{w}_{ik}/y_{ik})$ is numerically ~~unstable~~ undefined when $y_{ik} = 0$ if the constraints in the disjuncts are nonlinear. Therefore, the method ~~potentially causes failures in finding~~ can potentially fail to find a solution to the GDP problem. This issue can be overcome by approximating the perspective function with an inequality as demonstrated in [44]. The ~~obtained~~ resulting GDP-transformed MINLP using (HR) ~~goes as:~~

yields:

$$
\begin{aligned}
\min_{\mathbf{v},\mathbf{w},\mathbf{x},\mathbf{y},\mathbf{z}} \quad & f(\mathbf{x},\mathbf{z}) \\
\text{s.t.} \quad & \mathbf{g}(\mathbf{x},\mathbf{z}) \leq \mathbf{0} \\
& \mathbf{A}\mathbf{y} \geq \mathbf{a} \\
& \sum_{i \in D_k} y_{ik} = 1 \quad k \in K \\
& \mathbf{x} = \sum_{i \in D_k} \mathbf{v}_{ik} \quad k \in K \\
& \mathbf{z} = \sum_{i \in D_k} \mathbf{w}_{ik} \quad k \in K \\
& y_{ik}\mathbf{h}_{ik}(\mathbf{v}_{ik}/y_{ik}, \mathbf{w}_{ik}/y_{ik}) \leq 0 \quad i \in D_k, k \in K \\
& y_{ik}\underline{\mathbf{x}} \leq \mathbf{v}_{ik} \leq y_{ik}\overline{\mathbf{x}} \quad i \in D_k, k \in K \\
& y_{ik}\underline{\mathbf{z}} \leq \mathbf{w}_{ik} \leq y_{ik}\overline{\mathbf{z}} \quad i \in D_k, k \in K \\
& \mathbf{v}_{ik} \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}] \subseteq \mathbb{R}^{n_x}; \mathbf{w}_{ik} \in \{\underline{\mathbf{z}}, \dots, \overline{\mathbf{z}}\} \subseteq \mathbb{Z}^{n_z} \\
& \mathbf{x} \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}] \subseteq \mathbb{R}^{n_x}; \mathbf{y} \in \{0,1\}^{n_y}; \mathbf{z} \in \{\underline{\mathbf{z}}, \dots, \overline{\mathbf{z}}\} \subseteq \mathbb{Z}^{n_z}
\end{aligned}
\tag{HR}
$$

The hull reformulation introduces ~~extra~~ a larger number of constraints compared to the big-M method. However, it yields a tighter relaxation in the continuous space, refining the representation of the original GDP problem. This can potentially reduce the number of iterations required for MINLP solvers to reach ~~the~~ an optimal solution. Depending on the solver and the problem, the trade-off between these two reformulations might result in one of them yielding problems that are ~~more efficiently solvable~~solved more efficiently [43].

~~Despite MINLP reformulations being the default method to solve~~ Although MINLP reformulations are the standard approach for solving GDP problems, ~~these reformulations~~ they often introduce numerous algebraic constraints~~, some of which might not be relevant to a particular solution and might even lead to~~ . Some of these constraints may be irrelevant to a specific solution and can cause numerical instabilities when their corresponding variables are ~~equal to~~ zero. This ~~net effect might make the problem harder to solve and extend its~~ can increase the complexity of the problem and solution time, ~~opening the door for other~~ highlighting the potential of alternative GDP solution techniques that ~~do not transform~~ avoid transforming the problem into ~~a~~ an MINLP.

2.2 Generalized Disjunctive Programming ~~Logic-based~~ Logic-Based Solution Algorithms

Instead of reformulating the GDP into ~~MINLP~~ an MINLP, and solving the problem using MINLP solvers, some methods developed in the literature aim to directly exploit the logical constraints inside the GDP. Attempts to tackle the logical propositions for solving the GDP problem are known as logic-based methods. ~~Logic-based~~ Logic-Based solution methods are generalizations of MINLP algorithms that apply similar strategies to process Boolean variables to those used for integer variables in MINLP solvers. This category of algorithms includes techniques such as logic-based outer-approximation ~~(LOA)~~ and logic-based branch and bound ~~(LBB)~~ [9].

In GDP algorithms, the (potentially mixed-integer) ~~Nonlinear Programming (NLP)~~ nonlinear programming subproblems generated upon setting specific discrete combinations, which now encompass logical variables, are confined to only those constraints relevant to the logical variables set to $True$ in each respective combination. In logic-based algorithms, the generated Nonlinear Programming ~~(NLP) subproblems, which~~ subproblems, that could potentially be mixed-integer as well, arise from fixing specific Boolean configurations. These configurations constrain the (MI)NLP subproblems to only relevant constraints corresponding to logical variables set to $True$ in each setting. Specifically, when considering a given assignment for the logical variables denoted by $\hat{\mathbf{Y}}$, the resulting subproblem is defined as:

$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{z}} \quad & f(\mathbf{x},\mathbf{z}) \\
\text{s.t.} \quad & \mathbf{g}(\mathbf{x},\mathbf{z}) \leq \mathbf{0} \\
& \mathbf{h}_{ik}(\mathbf{x},\mathbf{z}) \leq \mathbf{0} \ \text{ if } \hat{Y}_{ik} = True \quad i \in D_k, k \in K \\
& \mathbf{x} \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}] \subseteq \mathbb{R}^{n_x}, \ \ \mathbf{z} \in \{\underline{\mathbf{z}}, \dots, \overline{\mathbf{z}}\} \subseteq \mathbb{Z}^{n_z}
\end{aligned}
\tag{Sub}
$$

~~This formulation~~ The formulation of Problem (Sub) represents the optimization problem under the constraints governed by the chosen logical assignment $\hat{\mathbf{Y}}$. In the most general case, after fixing all Boolean variables, the Problem (Sub) is a

MINLP. Still, in most applications, where there are no discrete decisions besides the ones represented in the Boolean space $n_z = 0$, Problem (Sub) becomes an NLP. This problem avoids evaluating numerically challenging nonlinear equations whenever their corresponding logical variables are irrelevant (i.e., "zero-flow" issues) [3]. The feasibility of Boolean variables in the original equation (GDP) depends on logical constraints $\Omega(\hat{\mathbf{Y}}) = True$. By evaluating these logical constraints, infeasible Boolean variable assignments can be eliminated without needing to solve their associated subproblems.

In general, logic-based methods can be conceptualized as decomposition algorithms. At the ~~upper level~~ upper-level problem, these methods focus on identifying ~~the~~ an optimal logical combination $\hat{\mathbf{Y}}$. This combination ensures that the subproblems (Sub), when solved, converge to ~~the~~ an optimal solution of Eq. (GDP). Overall, given a Boolean configuration, the subproblem (Sub) is a reduced problem that only considers relevant constraints, ~~is~~ and is therefore numerically more stable, and yields faster evaluations than a monolithic MINLP. Consequently, unlike mixed-integer methods, logic-based approaches can offer advantages, given they exploit the structure of the logical constraints.

A prevalent logic-based ~~approach is the Logic-based~~ method is the Logic-Based Outer-Approximation ~~(LOA) algorithm, which utilizes linear relaxations of the nonlinear constraints at iterations $l = 1, \ldots, L$ and iterations $L_{ik} = \{l \mid Y_{ik} = True$ for iteration $l\}$~~ algorithm, that uses linear relaxations to approximate the feasible region of the original problem. ~~This approach~~ Linear relaxations of nonlinear functions involve replacing nonlinear constraints with linear approximations over the feasible region. This approach simplifies complex optimization problems by replacing nonlinear constraints or objectives with linear approximations, making them easier to solve while providing bounds on an optimal solution. By utilizing linear approximations at iterations $l = 1, \ldots, L$ and iterations $L_{ik} = \{l \mid Y_{ik} = True$ for iteration $l\}$, LOA leads to the formulation of a linearized GDP, where ~~the~~ an optimal solution provides the integer combinations necessary for problem resolution. The upper-level problem (Main l-GDP) in the LOA method is as follows:

$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{z},\alpha} \ & \alpha \\
\text{s.t. } & \alpha \geq \bar{f}(\mathbf{x},\mathbf{z};\mathbf{x}^l,\mathbf{z}^l) \quad \forall\, l = 1,\ldots,L \\
& \bar{\mathbf{g}}(\mathbf{x},\mathbf{z};\mathbf{x}^l,\mathbf{z}^l) \leq 0 \quad \forall\, l = 1,\ldots,L \\
& \bigvee_{i \in D_k} \begin{bmatrix} Y_{ik} \\ \bar{\mathbf{h}}_{ik}(\mathbf{x},\mathbf{z};\mathbf{x}^l,\mathbf{z}^l) \leq \mathbf{0} \quad l \in L_{ik} \end{bmatrix} \quad k \in K \\
& \Omega(\mathbf{Y}) = True \\
& \mathbf{x} \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}] \subseteq \mathbb{R}^{n_x}, \ \ \mathbf{z} \in \{\underline{\mathbf{z}}, \ldots, \overline{\mathbf{z}}\} \subseteq \mathbb{Z}^{n_z}, \alpha \in \mathbb{R}_+
\end{aligned}
\tag{Main l-GDP}
$$

where $\bar{f}(\mathbf{x},\mathbf{z};\mathbf{x}^l,\mathbf{z}^l)$ is the linear relaxation of function $f(\mathbf{x},\mathbf{z})$ ~~for point $\{\mathbf{x}^l,\mathbf{z}^l\}$~~ at point $(\mathbf{x}^l,\mathbf{z}^l)$. A similar definition is given for the linear relaxations of the global constraints $\bar{\mathbf{g}}(\mathbf{x},\mathbf{z};\mathbf{x}^l,\mathbf{z}^l)$, and of the constraints inside of the disjunctions $\bar{\mathbf{h}}_{ik}(\mathbf{x},\mathbf{z};\mathbf{x}^l,\mathbf{z}^l)$. Inspired by the outer-approximation algorithm for MINLP [45], these linear relaxations can be built using ~~the~~ a first-order Taylor expansion around point $\{\mathbf{x}^l,\mathbf{z}^l\}$, i.e., $\bar{f}(\mathbf{x},\mathbf{z};\mathbf{x}^l,\mathbf{z}^l) = f(\mathbf{x}^l,\mathbf{z}^l) + \nabla_x f(\mathbf{x}^l,\mathbf{z}^l)^\top(\mathbf{x} - \mathbf{x}^l) + \nabla_z f(\mathbf{x}^l,\mathbf{z}^l)^\top(\mathbf{z} - \mathbf{z}^l)$. It is important to note that linear approximations are guaranteed to be relaxations only when the functions f, $\mathbf{g}$, and $\mathbf{h}_{ik}$ are convex. For convex nonlinear functions, these linear approximations provide valid bounds on the optimal solution.

Problem (Main l-GDP) is usually reformulated into a Mixed-Integer Linear Programming (MILP) problem using the reformulations outlined in §2.1. Upon solving the main MILP problems, the logical combination $\hat{\mathbf{Y}}$ is determined, defining the subsequent Problem (Sub) with the resulting logical combination. Expansion points for additional constraints are then provided to solve the subproblem (Sub) within the context of (Main l-GDP). While (Main l-GDP) yields a rigorous lower bound, the (Sub) subproblem provides feasible solutions, thus establishing feasible upper bounds. Each iteration refines the linear approximation of (Main l-GDP), progressively tightening the constraints and guiding the current solution towards an optimal solution~~toward the optimal of the GDP~~.

~~Gradient-based~~ Gradient-Based linearizations provide a valid relaxation for convex nonlinear constraints, but do not guarantee an outer-approximation for nonconvex ones. This limitation jeopardizes the convergence guarantees to globally optimal solutions of LOA for nonconvex GDP problems. To address this problem, if the linearization of the functions defining the constraints is ensured to be a relaxation of the nonlinear constraints, LOA can converge to global solutions in nonconvex GDP problems. These relaxations remain to be linear constraints, often constructed using techniques such as multivariate McCormick envelopes [46]. This generalization is known as Global ~~Logic-based~~ Logic-Based Outer-Approximation (GLOA).

Another important logic-based solution method is the ~~Logic-based~~ Logic-Based Branch and Bound ~~(LBB)~~ algorithm that systematically addresses GDP by traversing Boolean variable values within a search tree. Each node in this tree ~~signifies~~ corresponds to a partial assignment of these variables. LBB solves optimization problems by splitting them into smaller subproblems with fixed logic variables and eliminating subproblems that violate the constraints through a branch and bound technique.

The core principle of LBB is to branch based on the disjunction, enabling it to neglect the constraints in inactive disjunctions. Furthermore, LBB accelerates the search for ~~the~~ an optimal solution by focusing solely on logical propositions that are satisfied. Initially, all disjunctions are unbranched, and we define this set of unbranched disjunctions as KN. The LBB starts with the relaxation of the GDP model (node-GDP) in which all nonlinear constraints from the disjunctions are ignored. For every node l, the set of branched disjunctions KB can be defined as $KB^l = K \setminus KN^l$.

$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{Y},\mathbf{z}} \; & f^l(\mathbf{x},\mathbf{z}) \\
\text{s.t. } & \mathbf{g}(\mathbf{x},\mathbf{z}) \leq \mathbf{0} \\
& \Omega(\mathbf{Y}^l) = True \\
& \mathbf{h}_{ik}(\mathbf{x},\mathbf{z}) \leq \mathbf{0} \quad \text{if } \hat{Y}^l_{ik} = True, \quad k \in KB^l \\
& \bigvee_{i \in D_k} \begin{bmatrix} Y_{ik} \\ \Psi_{ik}(\hat{\mathbf{Y}}^l) = True \end{bmatrix} \quad k \in KN^l \\
& \mathbf{x} \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}] \subseteq \mathbb{R}^{n_x}; \mathbf{Y} \in \{False, True\}^{n_y}; \mathbf{z} \in \{\underline{\mathbf{z}}, \dots, \overline{\mathbf{z}}\} \subseteq \mathbb{Z}^{n_z}
\end{aligned}
\tag{node-GDP}
$$

where Ψ denotes the ~~set~~ set-valued function of constraints relevant for the unbranched nodes KN^l.

At each iteration, the algorithm selects the node with the minimum objective solution from the queue. The objective value of each evaluated node in the queue serves as a lower bound for subsequent nodes. Eventually, the minimum objective value among all nodes in the queue establishes a global lower bound on the GDP. Branching out all the disjunctions, the algorithm terminates if the upper bound to the solution, determined by the best-found feasible solution, matches the global lower bound.

As mentioned above, logic-based methods leverage logical constraints within the GDP by activating or deactivating algebraic constraints within logical disjunctions during ~~problem-solving~~problem solving. In the branching process, infeasible nodes that violate logical propositions may be found. These nodes are pruned if they do not satisfy the relevant logical constraints $\Psi(\mathbf{Y}) = True$.

~~The methods described in this section require access to the original GDP problem. Such an interface has been provided by a few software packages, including Pyomo.GDP [41]. The LOA, GLOA, and LBB algorithms are evaluated in this work through their implementation in the GDP solver in Pyomo, GDPOpt [41].~~

While logic-based methods offer ~~advantages, there are still limitationsassociated with algorithms~~several advantages, they also have some limitations. For nonconvex GDP problems, LOA may ~~face challenges in identifying~~ struggle to identify the global optimum, as ~~the~~ solutions to the NLP subproblems ~~may~~ might not correspond to the global optimum. ~~Analogously, LBB requires significant~~ Similarly, the LBB method can be resource-intensive, requiring substantial computational time and resources, ~~especially~~ particularly for large and complex problems [41]. More specifically, as the problem size increases, the number of subproblems tends to grow exponentially. ~~Hence,~~ Thus, there is an ongoing need for more efficient logic-based algorithms ~~capable of leveraging~~ that can effectively leverage the logical structure of GDP problems~~remain needed~~.

The methods described in this section require access to the original GDP problem. Such an interface has been provided by a few software packages, including Pyomo.GDP [41]. The LOA, GLOA, and LBB algorithms are evaluated in this work through their implementation in the GDP solver in Pyomo, GDPOpt [41].

2.3 Discrete Convex Analysis and the Discrete-Steepest Descent Algorithm

Unlike traditional MINLP and GDP solution strategies, which rely on conventional convexity theory treating discrete functions as inherently nonconvex, the Discrete-Steepest Descent Algorithm ~~(D-SDA)~~ incorporates an optimality condition based on discrete convex analysis. This framework provides an alternative theoretical foundation for discrete optimization, defining convexity structures for discrete functions [34].

~~In discrete convex analysis, the solution of~~ In this context, an Integer Programming (IP) problem is considered locally optimal when the discrete variables ~~are optimal within a predefined neighborhood. Thus, the neighborhood choice has~~

~~a direct impact on the~~ yield the lowest objective value within a defined *neighborhood*. Specifically, this means that the point $\mathbf{z}$ must have an objective value lower or equal than all its neighboring points. Formally, the neighborhood N_k of a point $\mathbf{z}$ is defined as all the integer points α (called *neighbors*) within a k-ball of radious one centered around $\mathbf{z}$:

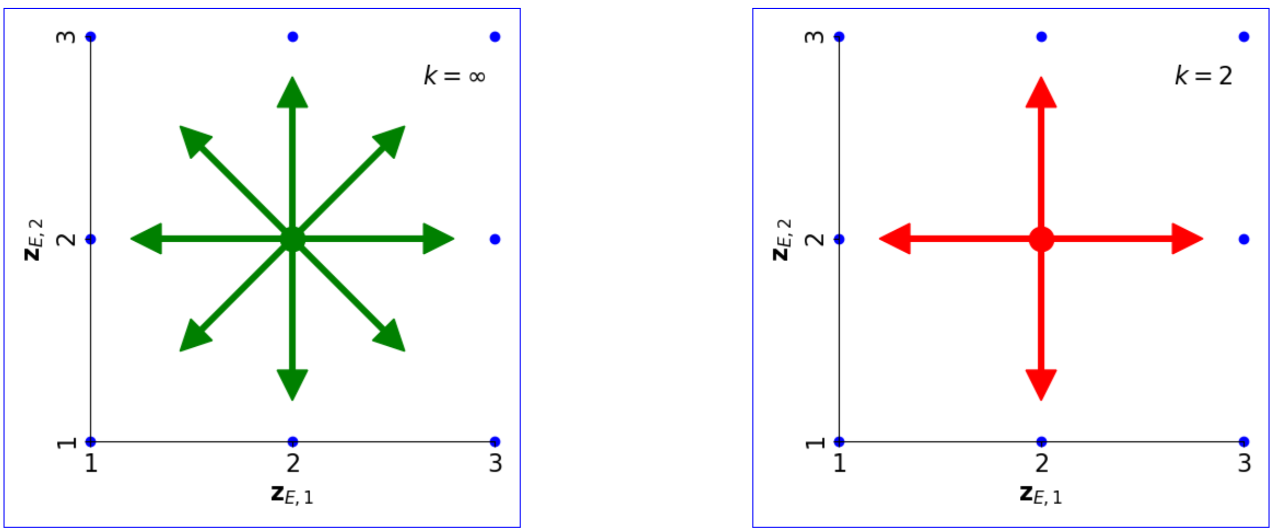
$$N_k(\mathbf{z}) = \{\alpha \in \mathbb{Z}^{n_z} : \ \|\alpha - \mathbf{z}\|_k \leq 1\} \tag{1}$$

Once the neighborhood of a point $\mathbf{z}$ is identified, the set of directions $\Delta_k(\mathbf{z})$ to each of its neighboring points can be computed through vector subtraction, as shown by the following equation,

$$\Delta_k(\mathbf{z}) = \{\mathbf{d} : \ \alpha - \mathbf{z} = \mathbf{d}, \forall\, \alpha \in N_k(\mathbf{z})\} \tag{2}$$

which measures of how far apart the neighbors are in the lattice.

In this work, we consider $k \in \{2, \infty\}$ and Figure 1 illustrates both neighborhoods for the case of two dimensions. Therefore, the choice of neighborhood directly affects the local optimum obtained. ~~Notably, local optimality~~ Importantly, under certain conditions, local optimality within specific neighborhoods can imply global optimality~~for specific neighborhoods under certain conditions. For instance~~. For example, global optimality is ~~assured~~ guaranteed for unconstrained integer problems with a separable convex objective function ~~by employing~~ when the positive and negative coordinates of the ~~axis~~ axes are used as neighbors.



(a) ∞-neighborhood (N_∞) (b) 2-neighborhood (N_2)

Figure 1: Visualization of the two neighborhoods N_∞ and N_2 on a two-variable discrete lattice, centered at the point $\mathbf{z_E} = (2, 2)$. The ∞-neighborhood allows movement to all points within unitary Euclidean distance, offering a more flexible search space, while the 2-neighborhood restricts movement to orthogonal directions, providing a more constrained search. This illustrates how the choice of neighborhood affects the directions explored during optimization.

Within the discrete convex analysis framework, an important concept is the ~~idea~~ notion of *integrally convex* objective functions, as introduced by Favati [39]. A function is ~~considered *integrally convex*~~ classified as integrally convex when its local convex extension is convex. This extension is constructed by linearly approximating the original function within unit hypercubes of its domain~~(for more details,~~, allowing for a more flexible approach to defining convexity in discrete spaces (see Murota [34]). Integrally convex functions are particularly relevant because they encompass ~~most~~ many of the discrete convex functions ~~found~~ commonly studied in the literature, including separable convex functions [47]. ~~It is worth noting that a MINLP may be integrally convex~~

An MINLP problem may exhibit integral convexity even if it is nonconvex ~~according to the common understanding of convexity in MINLP optimization, i.e., a MINLP is traditionally classified as~~ by traditional MINLP standards, where a problem is considered convex if its continuous relaxation is convex [38].

~~As a convention~~This distinction is important for understanding how integral convexity can be leveraged in optimization problems that might otherwise be categorized as nonconvex. In discrete convex analysis, an optimal solution over the ~~infinity neighborhood (or~~ ∞-neighborhood (denoted N_∞) is referred to as ~~integrally local~~ *integrally local* (*i-local*)~~because~~ , meaning it is globally optimal for an integrally convex objective function. Similarly, ~~a solution~~

~~optimal within the separable neighborhood (or 2-neighborhood)is called separable local~~ an optimal solution within 2-neighborhood (denoted N_2), which is sometimes referred to as the separable neighborhood, is known as *separable local* (*s-local*) ~~since it is globally optimal~~ as it represents a global optimum for a separable convex objective function ~~.[33, 36]. Both neighborhoods are shown in Figure 1 for the case of two dimensions~~[33, 36]. This distinction between i-local and s-local optimality is important for understanding how different neighborhoods affect the global optimality guarantees for discrete optimization problems under the lens of discrete convex analysis.

~~All neighbors $k = \infty$ Single-step neighbors $k = 2$ Different neighborhood explorations alternatives in discrete variables lattice.~~

~~The first extension to this theory for MINLPs was introduced~~ The theory of discrete convex analysis was first extended to MINLP in [33]. ~~This work proposed~~ Here, the authors introduced a decomposition approach designed for problems with ordered binary variables~~that were reformulated with~~ , which were reformulated using the *external variable* method. ~~The external variables created were then~~ In this approach, external variables were decoupled from the ~~rest of the problem~~ main problem and addressed in an upper-level problem. ~~Later~~Then, the D-SDA was ~~applied to optimize the external variablesin the upper-level problem where binary variables were~~ utilized to optimize these external variables, with binary variables fixed accordingly. ~~Here~~Furthermore, the objective function values came from the solution of NLP optimization subproblems. ~~The main advantage of addressing superstructure optimization problems with the~~

A significant advantage of using D-SDA is that binary variables reformulated ~~into external variables no longer need to be evaluated at~~ as external variables are evaluated only at discrete points. This avoids the issue of evaluating fractional solutions (e.g., a binary variable evaluated at 0.5) ~~because~~ given that evaluating points at discrete points ~~is enough~~ suffices to assess discrete optimality requirements. As a result, D-SDA avoids the potential nonconvexities introduced by the continuous relaxation of MINLP superstructures, e.g., the multi-modal behavior found when optimizing the number of stages in a catalytic distillation column or the number of reactors in series [33, 35]. ~~Also, while updating and fixing *external variables* successively with the D-SDA, the initialization of variables optimized in the subproblems can be monitored and updated, allowing the application of the D-SDA~~ Similarly, this algorithm was succesfully applied to highly nonlinear MINLP problems such as the optimal design of rate-based and dynamic distillation systems [36, 48].

~~In contrast to the IP case studied by other authors[47, 49], when considering~~ Guaranteeing global optimality for MINLPs or GDPs ~~, providing global optimality guarantees from a discrete convex analysis perspective is challenging since the objective function value~~ problems remains challenging. Unlike previously studied IP problems, where global optimality can sometimes be ensured, MINLP problems involve nonlinear objective functions for each discrete point~~corresponds to the solution of a subproblem, which is usually nonlinear. Thus, an inherent limitation of applying the D-SDA to MINLP problems is that global optimality cannot be guaranteed. Nevertheless, the~~ . This complexity makes it difficult to guarantee global optimality [47, 49]. The D-SDA ~~aims~~ seeks to find the best ~~solution possible by choosing neighborhoods adequately. So far,~~ possible solution by choosing appropriate neighborhoods, with the ∞-neighborhood ~~has been used as a reference~~ often used for local optimality in MINLP problems. ~~One reason for this is the inclusiveness of this neighborhood , encompassing every discrete point~~ This neighborhood includes all discrete points within an infinity norm of the evaluated point, ~~as depicted in Figure 1, instead of~~ offering more comprehensive coverage than just positive and negative coordinates~~.~~ , as illustrated in Figure 1. Additionally, when applying the D-SDA with the ∞-neighborhood to a binary optimization problem (without reformulation), a complete enumeration over discrete variables is required. While not computationally efficient, this method offers a "brute-force" alternative for addressing small-scale discrete optimization problems.

~~Motivated by those previous works, in this paper ,~~ Building on these advancements, this paper extends the D-SDA ~~is extended~~ methodology to address more general GDP problems~~in the following section by directly~~ . This extension involves exploring the search space of reformulated Boolean variables ~~without~~ directly, eliminating the need for a (BM) or (HR) ~~reformulations step and without the need for a linearization of the original problem as~~ reformulation step and avoiding the linearization required in the LOA method.

3 The ~~Logic-based~~ Logic-Based Discrete-Steepest Descent Algorithm as a Generalized Disjunctive Programming Algorithm

This section presents ~~Logic-based~~ Logic-Based D-SDA (LD-SDA) as a GDP algorithm. It begins with an explanation of the reformulation process for Boolean variables into external variables, outlining the requirements necessary for reformulation. For this, we provide a comprehensive example for demonstration. Second, the basis of the LD-SDA as a decomposition algorithm that utilizes the structure of the external variables is elucidated. The following subsection describes the different algorithms that compose the ~~Logic-based~~ Logic-Based D-SDA in the context of solving a GDP problem. The properties of LD-SDA are explained in the final subsection.

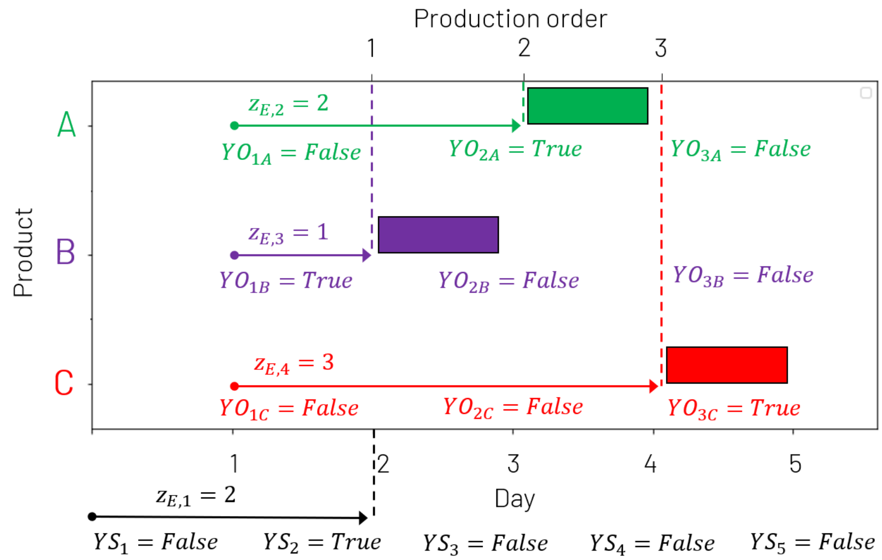


Figure 2: ~~Reformulation~~ Visualization of ~~independent Boolean variables to~~ the external ~~variables~~ variable reformulation for an illustrative multi-product batch scheduling example. The figure explicitly displays the Boolean variables for starting time ($\mathbf{YS}$) and production order ($\mathbf{YO}_c$). Production begins on the second day, represented by $YS_2 = True$ in black. The production order is B, A, and C, indicated by $YO_{1B} = True$ (purple), $YO_{2A} = True$ (green), and $YO_{3C} = True$ (red). The maintenance variable YM is not reformulated as it does not meet the necessary criteria for this transformation.

3.1 GDP Reformulations Using External Variables

Consider GDP problems where a subset of the Boolean variables in $\mathbf{Y}$ can be reformulated into a collection of integer variables referred to as *external variables*. Thus, $\mathbf{Y}$ in (GDP) is defined as ~~$\mathbf{Y} = [\mathbf{Y_R}, \mathbf{Y_N}]$, where $\mathbf{Y_R} = [\mathbf{Y_{R1}}, \mathbf{Y_{R2}}, ..., \mathbf{Y_{Rn_R}}]$~~ $\mathbf{Y} = (\mathbf{Y_R}, \mathbf{Y_N})$, where $\mathbf{Y_R} = (\mathbf{Y_{R1}}, \mathbf{Y_{R2}}, ..., \mathbf{Y_{Rn_R}})$ contains those vectors of independent Boolean variables that can be reformulated using external variables. This means that each vector $\mathbf{Y_{R}}_j$ will be reformulated with one external variable, and this reformulation is applied for every j in $\{1, 2, ..., n_R\}$. ~~It is important to note that unless explicitly stated otherwise, all indices j are referenced within the set $\{1, 2, ..., n_R\}$, although the explicit mention is omitted for notation simplicity.~~ Finally, as a requisite to apply the reformulation, each vector ~~$\mathbf{Y_{R}}_j$~~ $\mathbf{Y_{R}}_j \forall j \in \{1, 2, ..., n_R\}$ must satisfy the following conditions:

- **Requirement 1**: Every Boolean variable in $\mathbf{Y_{R}}_j$ must be defined over a finite well-ordered set S_j [50, p. 38]. This set may be different for each vector of variables; thus, it is indexed with j. In addition, variables defined over S_j must represent ordered decisions such as finding discrete locations, selecting discrete points in time, counting the number of times a task is performed, etc. Notably, these independent Boolean variables can have ~~indexes~~ indices other than the ordered set. Also, not every Boolean variable defined over S_j is necessarily required to be in $\mathbf{Y_{R}}_j$. For instance, the Boolean variables that determine the feed stage in a distillation column are defined over the set of trays, but some trays may be excluded from $\mathbf{Y_{R}}_j$ if needed.
- **Requirement 2**: Boolean variables $\mathbf{Y_{R}}_j$ are subject to a partitioning constraint $\texttt{Exactly}(1, \mathbf{Y_{R}}_j)$, i.e., exactly ~~1~~ one variable within $\mathbf{Y_{R}}_j$ is *True* [51]. For example, in the case where there are only two independent Boolean variables (~~$\mathbf{Y_{R}}_j = [Y_1, Y_2]$~~ $\mathbf{Y_{R}}_j = (Y_1, Y_2)$) the constraint is equivalent to $Y_1 \veebar Y_2$. Note that, if the Boolean variables are transformed into binary variables, this is equivalent to a cardinality constraint $\sum_{i \in S_j} y_i = 1$ [52].

3.1.1 External Variable Reformulation: Illustrative Example

To illustrate these requirements, consider the following example ~~.~~

~~**Example 1.**~~ ~~There exists~~ where a multi-product batch reactor that ~~produces~~ manufactures three products, A, B, and C ~~where the optimal starting day for each substance needs to be determined for~~ . The goal is to determine an optimal starting date for each product within a five-day time horizon. Additionally, the ~~order in which~~ production order for A, B, and C ~~are produced~~must be established, subject to demand constraints~~, has to be established. Furthermore, it must~~

~~be decided whether to perform maintenance or not before production starts. To formulate~~. Another decision involves whether or not to perform routine maintenance before production begins.

To model this with Boolean variables, we define $YS_t, \forall\ t \in T = \{1,2,3,4,5\}$ to indicate the starting time~~; ~~, $YO_{pc}, \forall\ c \in C = \{A,B,C\}, \forall\ p \in P = \{1,2,3\}$ to ~~determine~~ represent the production order, and YM to indicate the ~~existence of routine~~ decision about performing maintenance. The constraints of this problem dictate that there must be only one starting day (~~$\texttt{Exactly}(1, \mathbf{YS} = [YS_1, YS_2, ..., YS_5])$~~$\texttt{Exactly}(1, \mathbf{YS} = (YS_1, YS_2, ..., YS_5))$) and that each product must be produced only once (~~$\texttt{Exactly}(1, \mathbf{YO}_c = [YO_{1c}, YO_{2c}, YO_{3c}])\ \forall\ c \in C$~~$\texttt{Exactly}(1, \mathbf{YO}_c = (YO_{1c}, YO_{2c}, YO_{3c}))\ \forall\ c \in C$). These constraints imply that variables $\mathbf{YS}, \mathbf{YO}_A, \mathbf{YO}_B$, and $\mathbf{YO}_C$ satisfy **Requirement 2**. ~~Furthermore, since~~ Moreover, since both T and P are ordered sets, we conclude the aforementioned group of variables also satisfies **Requirement 1**. Hence, ~~these~~ the vectors of independent Boolean variables can be grouped as ~~$\mathbf{Y_R} = [\mathbf{YS}, \mathbf{YO}_A, \mathbf{YO}_B, \mathbf{YO}_C]$~~$\mathbf{Y_R} = (\mathbf{YS}, \mathbf{YO}_A, \mathbf{YO}_B, \mathbf{YO}_C)$, and reformulated with one external variable assigned to each vector in $\mathbf{Y_R}$. This means that $z_{E,1}$ is assigned to $\mathbf{YS}$, while $z_{E,2}, z_{E,3}$, and $z_{E,4}$ are assigned to $\mathbf{YO}_A, \mathbf{YO}_B$, and $\mathbf{YO}_C$~~respectively. This~~, respectively.

The resulting reformulation is illustrated in Figure 2 ~~, where the values of reformulated Boolean variables and external variables of a potential solutionare shown~~which shows the Boolean and external variable values of a possible solution. In this ~~possible solution, the operation starts on day~~ solution, production starts on Day 2, implying $YS_2 = True \Leftrightarrow \mathbf{z}_{E,1} = 2$, as ~~shown~~ indicated in black at the lower horizontal axis. ~~Analogously~~Similarly, the upper horizontal axis indicates the production order where ~~the arrangement depicted is to produce~~ B ~~first, then~~ is produced first, followed by A, and ~~finally~~ then C. ~~Such~~ This production order is represented by the reformulation $YO_{1B} = True \Leftrightarrow \mathbf{z}_{E,2} = 1, YO_{2A} = True \Leftrightarrow \mathbf{z}_{E,3} = 2$, and $YO_{3C} = True \Leftrightarrow \mathbf{z}_{E,4} = 3$. ~~Note that the remaining Boolean variable $\mathbf{Y_N} = YM$ in this example~~ Finally, note that the Boolean variable YM does not satisfy the ~~stated requirements . Hence, it is not reformulated. This work generalizes this external variable as discussed below.~~

~~To formally state the external variable reformulation, we~~ requirements for reformulation, so it remains as is.

3.1.2 External Variable Reformulation: Extension

This work extends the external variable reformulation to general cases. To formally describe this approach, consider an optimization problem in the ~~(GDP) form~~ form of (GDP). If the GDP problem satisfies **Requirements 1** and **2** over the vectors of independent Boolean variables $\mathbf{Y_R}$, then one external variable can be assigned to each vector ~~$\mathbf{Y_R}_j$.~~ $\mathbf{Y_R}_j, \forall\ j \in \{1, 2, ..., n_R\}$.

Requirement 1 indicates that each vector ~~$\mathbf{Y_R}_j$ must be defined over a well-ordered set S_j. Since not every Boolean variable defined over S_j is required to be in $\mathbf{Y_R}_j$, we declare subset $S'_j \subseteq S_j$ to denote the ordered sets where Boolean variables $\mathbf{Y_R}_j$ are declared. We define the vectors of independent variables as $(1, \mathbf{Y}_k), \forall\ k \in K$, where vector $\mathbf{Y}_k$ contains Boolean terms $Y_{ik}, \forall\ i \in D_k$. Consequently, if each disjunction k represented an ordered decision over the well-ordered set D_k, then **Requirement 1** would be directly satisfied, allowing to reformulate a standard (GDP) problem following the guidelines in equations (3) and (4) to instead obtain: $Y_{D_k(a),k} \iff z_{E,k} = a \quad \forall\ k \in K, \forall\ a \in \{1, 2, ..., |D_k|\}$ $1 \leq z_{E,k} \leq |D_k| \quad \forall\ k \in K$ respectively. In this case, there are as many external variables as disjunctions $k \in K$ in the formulation, making index j interchangeable with disjunction index k. Similarly, ordered subsets S'_j correspond to disjunct sets D_k. For this reason, indexes i in Y_{ik} are replaced by ordered index $D_k(a)$ in equation (5). In practice, not every Boolean variable in the formulation fulfills the requirements to be reformulated with external variables as suggested by equation (5). In addition, ordered discrete structures may appear outside disjunctions, e.g., within $\Omega(\mathbf{Y}) = True$. Therefore, the reformulation in equations (3) and (4) is more general and practical than equations (5) and (6). Generalizing the reformulation established in earlier research, our proposed reformulation is adapted to potentially be applied over variables $\mathbf{Y_R}_j$ defined over an ordered but unevenly spaced set. For this, instead of defining external variables using *elements* of ordered sets as in previous~~ $\mathbf{Y_R}_j, \forall\ j \in \{1, 2, ..., n_R\}$ must be defined over a well-ordered set S_j. Not all Boolean variables defined over S_j are required to belong to $\mathbf{Y_R}_j$. To account for this, we introduce the subset $S'_j \subseteq S_j$ for each $j \in 1, 2, ..., n_R$, representing the ordered sets in which the Boolean variables $\mathbf{Y_R}_j$ are declared. The vector of independent variables is then represented as $\mathbf{Y_R}_j = (Y_{Rj,S'_j(1)}, Y_{Rj,S'_j(2)}, ..., Y_{Rj,S'_j(|S'_j|)})$ where each $Y_{Rj,S'_j(a)}$ is a Boolean variable defined at position a of the well-ordered set S'_j and included in vector $\mathbf{Y_R}_j$. It is important to note that Boolean variables $Y_{Rj,S'_j(a)}, \forall\ j \in \{1, 2, ..., n_R\}, \forall\ a \in \{1, 2, ..., |S'_j|\}$ can be defined over other sets aside from $S'_j, \forall\ j \in \{1, 2, ..., n_R\}$. In other words, these variables may have multiple indices in the algebraic model formulation.

Requirement 2 indicates that $\Omega(\mathbf{Y}) = True$ in (GDP) must contain partitioning constraints of the form $\texttt{Exactly}(1, \mathbf{Y_R}_j), \forall\ j \in \{1, 2, ..., n_R\}$. Combining both requirements allows to define Boolean variables in $\mathbf{Y_R}$ as a function of n_R external variables $z_{E,j}, \forall\, j \in \{1,2,...,n_R\}$ as,

$$Y_{Rj,S'_j(a)} \iff z_{E,j} = a \quad \forall\, j \in \{1,2,...,n_R\}, \forall\, a \in \{1,2,...,|S'_j|\} \tag{3}$$

effectively expressing the external variables $z_{E,j}$ based on the values of Boolean variables $Y_{Rj,S'_j(a)}$. From this reformulation, the upper and lower bounds of the external variables can be directly inferred from the sets of ordered positions $\{1,2,...,|S'_j|\}, \forall\, j \in \{1,2,...,n_R\}$. These bounds are defined as:

$$1 \le z_{E,j} \le |S'_j| \quad \forall\, j \in \{1,2,...,n_R\} \tag{4}$$

The general external variable reformulation is given by equations (3) and (4). Next, we proceed to derive a simpler reformulation that follows from the special case when all the disjunctions are defined over well-ordered sets. First, note that **Requirement 2** is naturally satisfied by the disjunctions in a standard (GDP) formulation. This arises from the fact that the exclusivity requirement in disjunctions enforces constraints of the form $\texttt{Exactly}(1, \mathbf{Y}_k), \forall\, k \in K$, where vector $\mathbf{Y}_k$ contains Boolean terms $Y_{ik}, \forall\, i \in D_k$. Consequently, if each disjunction k represented an ordered decision over the well-ordered set D_k, then **Requirement 1** would be directly satisfied, allowing to reformulate a standard (GDP) problem following the guidelines in equations (3) and (4) to instead obtain:

$$Y_{D_k(a),k} \iff z_{E,k} = a \quad \forall\, k \in K, \forall\, a \in \{1,2,...,|D_k|\} \tag{5}$$

$$1 \le z_{E,k} \le |D_k| \quad \forall\, k \in K \tag{6}$$

respectively. In this case, there are as many external variables as disjunctions $k \in K$ in the formulation, making index j interchangeable with disjunction index k. Similarly, ordered subsets S'_j correspond to disjunct sets D_k. For this reason, indices i in Y_{ik} are replaced by ordered index $D_k(a)$ in equation (5). In practice, not every Boolean variable in the formulation fulfills the requirements to be reformulated with external variables as suggested by equation (5). In addition, ordered discrete structures may appear outside disjunctions, e.g., within $\Omega(\mathbf{Y}) = True$. Therefore, the reformulation in equations (3) and (4) is more general and practical than equations (5) and (6).

Generalizing the reformulation established in previous research, we adapt the proposed reformulation to handle variables $\mathbf{Y}_{\mathbf{R}j}$ defined over ordered, but unevenly spaced sets $\forall\, j \in \{1,2,...,n_R\}$. Instead of defining external variables based on the *elements* of these ordered sets S_j as in earlier works [33], we propose defining external variables ~~with respect to~~ based on the *positions* in ~~ordered sets. In this case, the~~ the ordered sets S'_j. For each j, the set of positions is denoted as $1,2,\ldots,|S'_j|$, where the distance between consecutive elements ~~in the set of positions $\{1,2,...,|S'_j|\}$~~ is equal to ~~1. Consequently,~~ one. This change avoids potential issues with solutions defined by isolated discrete elements~~will not negatively affect the LD-SDA search, whose iterations are based on a~~ , since the nearest-neighborhood ~~exploration over external variables~~ search in the LD-SDA is now defined over positions ~~. To illustrate this potential issue, consider , for example,~~ instead of elements.

To illustrate the problem that may arise when considering the reformulation in terms of elements, consider an unevenly spaced set $S'_1 = \{0,1,2,7,10\}$, and its corresponding Boolean variables ~~$\mathbf{Y}_{\mathbf{R}1} = [Y_{R1,0}, Y_{R1,1}, Y_{R1,2}, Y_{R1,7}, Y_{R1,10}]$, where~~ $\mathbf{Y}_{\mathbf{R}1} = (Y_{R1,0}, Y_{R1,1}, Y_{R1,2}, Y_{R1,7}, Y_{R1,10})$, with the partitioning constraint $\texttt{Exactly}(1, \mathbf{Y}_{\mathbf{R}1}) = True$. Suppose the incumbent point is $z_{E,1} = 4$~~(~~, which corresponds to $Y_{R1,7}$ = *True*~~), thus its nearest neighbors are~~ . A neighborhood search around the incumbent would explore $z_{E,1} = 3$ and $z_{E,1} = 5$. ~~This stems from starting at 4 and searching its immediate neighboring points i.e., 3 and 5. For those neighbors, the definitions from previous work [33] would set one of their respective Boolean variables to *True* (~~According to previous definitions [33], this search would attempt to set $Y_{R1,3}$ = *True* or $Y_{R1,5}$ = *True*~~) and the rest of~~ , while assigning the remaining Boolean variables in $\mathbf{Y}_{\mathbf{R}1}$ to *False*. ~~Given that~~ Since $3 \notin S'_1$ and $5 \notin S'_1$, both ~~$z_{E,1} = 3$ and $z_{E,1} = 5$ would be declared as infeasible. Thus, the current incumbent $z_{E,1} = 4$ would be interpreted~~ neighboring points would be treated as infeasible, causing the search to stop prematurely, identifying $zE, 1 = 4$ as a local optimum~~, and the neighbor search will stop at this point. This problematic can be avoided using the proposed definition , where the reformulation is performed in terms of~~ . Our proposed definition resolves this issue by conducting the reformulation over set positions. In ~~our the example, the new definition would interpret~~ this case, the neighboring points $z_{E,1} = 3$ and $z_{E,1} = 5$ ~~as~~ would correspond to positions over $\mathbf{Y}_{\mathbf{R}1}$~~. This means~~ , meaning that either $Y_{R1,2}$ or $Y_{R1,10}$ would be set to *True*, instead of $Y_{R1,3}$ or $Y_{R1,5}$~~, and the discrete exploration would proceed~~. This allows the discrete search to continue without prematurely declaring a local optimum.

3.2 GDP Decomposition Using External Variables

The reformulation presented in the previous section allows to express some of the Boolean variables in the problem in terms of the external variables as $\mathbf{Y_R} = \mathbf{Y_R}(\mathbf{z_E})$, where $\mathbf{z_E}$ is a vector of n_R external variables. The core idea of the LD-SDA is to move these external variables to an upper-level problem (Upper) and the rest of the variables to a subproblem (7). This decomposition allows taking advantage of the special ordered structure of the *external variables* by using a Discrete-Steepest Descent Algorithm ~~(D-SDA) in the upper level~~ in the upper-level problem to explore their domain as explained in §3.3. Once an external variable configuration is determined by D-SDA, a subproblem is obtained by only ~~considered~~ considering the active disjuncts of that specific $\mathbf{z_E}$ configuration. The formal definition of both problems is given as:

$$\begin{aligned} &\min_{\mathbf{z_E}} f_{sub}(\mathbf{z_E}) \\ &\text{s.t. } \mathbf{z_E} \in \{\underline{\mathbf{z_E}}, \dots, \overline{\mathbf{z_E}}\} \subseteq \mathbb{Z}^{n_{z_E}} \qquad \text{(From Eq. (4))} \end{aligned} \tag{Upper}$$

$$s(\mathbf{z_E}) = \begin{cases} f_{sub}(\mathbf{z_E}) = & \min_{\mathbf{x},\mathbf{Y_N},\mathbf{z}} f(\mathbf{x},\mathbf{z}) \\ & \text{s.t. } \mathbf{Y_R} = \mathbf{Y_R}(\mathbf{z_E}) \quad \text{(Fixed as shown in Eq. (3))} \\ & \quad \mathbf{g}(\mathbf{x},\mathbf{z}) \leq \mathbf{0} \\ & \quad \Omega(\mathbf{Y}) = True \\ & \quad \underline{\bigvee}_{i\in D_k} \begin{bmatrix} Y_{ik} \\ \mathbf{h}_{ik}(\mathbf{x},\mathbf{z}) \leq \mathbf{0} \end{bmatrix} \quad k \in K \\ & \quad \mathbf{x} \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}] \subseteq \mathbb{R}^{n_x}; \mathbf{Y} \in \{False, True\}^{n_y}; \mathbf{z} \in \{\underline{\mathbf{z}}, \dots, \overline{\mathbf{z}}\} \subseteq \mathbb{Z}^{n_z} \end{cases} \tag{7}$$

In problem (Upper), a value for the objective function $f_{sub}(\mathbf{z_E})$ is obtained by the optimization of subproblem (7). Thus, $f_{sub}(\mathbf{z_E})$ is defined as ~~the~~ an optimal objective function value found by optimizing the subproblem $s(\mathbf{z_E})$, obtained by fixing external variables fixed at $\mathbf{z_E}$. If the subproblem (7) is infeasible, $f_{sub}(\mathbf{z_E})$ is set to ~~as~~ positive infinity by convention. Notably, the subproblems are reduced formulation given that they only consider the relevant constraints for the relevant external variable configuration $\mathbf{z_E}$.

A novel feature of the LD-SDA is its ability to handle various types of subproblems, extending previous versions [33], ~~which~~ that solely supported NLP subproblems. In its most general form, the ~~lower-layer~~ lower-level problem (7) is a (GDP) with continuous ($\mathbf{x}$), discrete ($\mathbf{z}$) and non-reformulated Boolean ($\mathbf{Y_N}$) variables. Consider the scenario where every Boolean variable can be reformulated (e.g., as shown in equations (5) and (6)) or every non-reformulated variable $\mathbf{Y_N}$ is equivalently expressed in terms of $\mathbf{Y_R}$. Note that the ~~later~~ latter situation may occur if all Boolean variables $\mathbf{Y_N}$ are determined within the subproblem upon fixing $\mathbf{Y_R}$, implying that logic constraints $\Omega(\mathbf{Y}) = True$ establish $\mathbf{Y_N}$ as functions of $\mathbf{Y_R}$. In such a scenario, the resulting subproblem becomes an (MINLP) with continuous ($\mathbf{x}$) and discrete variables ($\mathbf{z}$), or an NLP if there are no discrete variables ($\mathbf{z}$) in the formulation. In the following subsection, we introduce the LD-SDA as a decomposition algorithm that leverages the external variable reformulation and bi-level structure depicted so far.

3.3 ~~Logic-based~~ Logic-Based Discrete-Steepest Descent Algorithm

The ~~Logic-based~~ Logic-Based Discrete-Steepest Descent Algorithm~~(LD-SDA)~~, as described in Algorithm 1, solves a series of subproblems (7) until a stopping criterion is satisfied. The LD-SDA can only start once the external variable reformulation of the problem has been performed. The external variables $\mathbf{z_E}$ are handled in an upper optimization ~~layer~~ level where the algorithm is performed. To initialize, this method requires an initial fixed value of external variables $\mathbf{z_{E,0}}$, the value of the variables of its corresponding feasible solution $(\mathbf{x_0}, \mathbf{Y_0}, \mathbf{z_0})$, and its respective objective function value $f_{sub}(\mathbf{z_{E,0}})$. Finding a starting feasible solution is beyond the scope of this work; however, it would be enough to have $\mathbf{z_{E,0}}$ and solve the subproblem $s(\mathbf{z_{E,0}})$ to find the rest of the required initial solution. Note that problem-specific initialization strategies have been suggested in the literature, e.g., see [48].

The LD-SDA explores a neighborhood within the external variable domain; hence, the user must ~~to~~ determine the type of neighborhood k that will be studied. ~~In this paper~~As mentioned in §2.3, we only consider $k \in \{2, \infty\}$ ~~as shown in Figure 1~~(see Figure 1); nevertheless, other types of discrete neighborhoods can be considered [34]. Once k has been selected, the neighborhood N_k of ~~a given~~ the current point $\mathbf{z_E}$ is defined as $N_k(\mathbf{z_E}) = \{\alpha \in \mathbb{Z}^{n_{z_E}} : \|\alpha - \mathbf{z_E}\|_k \leq 1\}$ as given by Equation 1. Similarly, the set of ~~distances~~ directions Δ_k from the point $\mathbf{z_E}$ to each neighbor α is be calculated as $\Delta_k(\mathbf{z_E}) = \{\mathbf{d} : \alpha - \mathbf{z_E} = \mathbf{d}, \forall\, \alpha \in N_k(\mathbf{z_E})\}$ as dicatated by Equation 2.

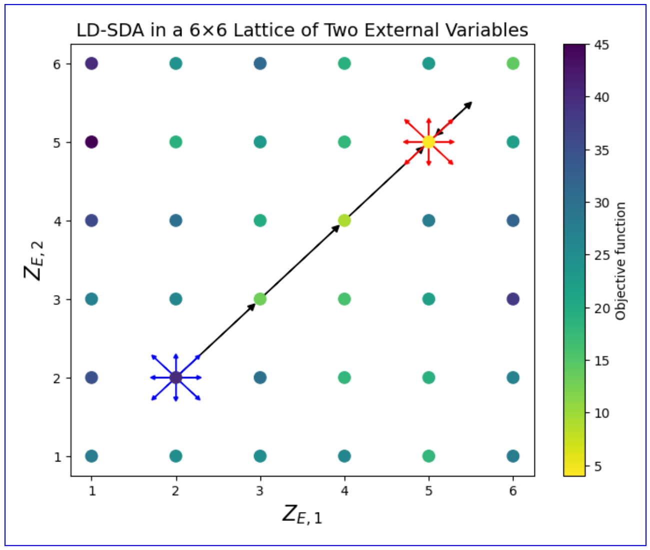


Figure 3: Visualization of the LD-SDA ~~with $k=\infty$~~algorithm using N_∞ in a two-variable discrete lattice example. ~~First,~~ In this example we initialize the algorithm ~~is initialized in [2, 2]~~begins at the initial point $(2,2)$. ~~Neighbor~~ A Neighborhood Search ~~in~~ within the neighborhood ~~$N_\infty([2,2])$ (~~$N_\infty((2,2))$, represented ~~with~~ by blue arrows~~) finds~~, identifies the best neighbor ~~is [3, 3]; hence~~as $(3,3)$, resulting in the steepest descent direction ~~$\mathbf{d}^*$ is [1, 1]~~$\mathbf{d}^*=(1,1)$. ~~Then,~~ A Line Search~~(represented~~, depicted with black arrows~~) is performed up to~~, follows this direction until reaching point ~~[5, 5] and stopped there because $f_{sub}([5,5]) < f_{sub}([5,5]+\mathbf{d}^*)$~~$(5,5)$, where it stops given that $f_{sub}((5,5)) < f_{sub}((5,5)+\mathbf{d}^*)$. A second ~~Neighbor~~ Neighborhood Search in ~~$N_\infty([5,5])$ is performed (represented in~~ $N_\infty((5,5))$, shown with red arrows~~) and concludes that point [5, 5]~~, determines $(5,5)$ is integrally local~~; therefore~~, terminating the algorithm~~terminates~~.

The next step is to perform ~~Neighbor~~ Neighborhood Search (see Algorithm ~~??), which~~ 2), that consists of a local search within the defined neighborhood. Essentially, this algorithm solves $s(\alpha)\ \forall\ \alpha \in N_k(\mathbf{z_E})$ and compares the solutions found with the best incumbent solution $f_{sub}(\mathbf{z_E})$. If, in a minimization problem, $f_{sub}(\mathbf{z_E}) \leq f_{sub}(\alpha)\ \forall\ \alpha \in N_k(\mathbf{z_E})$ then, the current solution in $\mathbf{z_E}$ is a discrete local minimum (*i-local* or *s-local* depending on the value of k); otherwise, the steepest descent direction $\mathbf{d}^* = \alpha^* - \mathbf{z_E}$ is computed, the algorithm moves to the best neighbor by letting $\mathbf{z_E} = \alpha^*$ and performs a Line Search in direction $\mathbf{d}^*$. Note that for a neighbor to be considered the *best neighbor* α^*, it must have a feasible subproblem, and a strictly better objective than both the incumbent solution and its corresponding neighborhood.

The Line Search (see Algorithm 3) determines a point in the direction of steepest descent $\beta = \mathbf{z_E} + \mathbf{d}^*$ and evaluates it. If the subproblem $s(\beta)$ is feasible and $f_{sub}(\beta) < f_{sub}(\mathbf{z_E})$ then, let $\mathbf{z_E} = \beta$ and perform the Line Search again until the search is unable to find a better feasible solution in direction $\mathbf{d}^*$. Once this occurs, the general algorithm should return to calculate $N_k(\mathbf{z_E})$ and $\Delta_k(\mathbf{z_E})$ to perform the ~~Neighbor~~ Neighborhood Search again in a new iteration.

The LD-SDA will terminate once the ~~Neighbor~~ Neighborhood Search is unable to find a neighbor α with a feasible subproblem $s(\alpha)$ that strictly improves the incumbent solution as $f_{sub}(\alpha) < f_{sub}(\mathbf{z_E})\ \forall\ \alpha \in N_k(\mathbf{z_E})$. In that case, the point is considered a discrete *i-local* or *s-local* minimum, and the algorithm will return the values of both variables $(\mathbf{x}, \mathbf{Y}, \mathbf{z}, \mathbf{z_E})$ and the objective function f^*_{sub} of the solution found. The stopping criterion employed indicates that the current point has the best objective function amongst its immediate discrete neighborhood mapping [34]. In rigorous terms, the integrally local optimality condition can only be guaranteed after an N_∞ exploration given that this is the neighborhood that considers the entire set of immediate neighbors (*i-local* optimality). Therefore, when using neighborhood N_2, it is up to the user to choose if the final solution $\hat{\mathbf{z_E}}$ is to be certified as integrally local by checking its $N_\infty(\hat{\mathbf{z_E}})$ neighborhood.

Figure 3 illustrates and explains the LD-SDA executed in its entirety on a 6×6 lattice of two external variables. For this example, the ∞-neighborhood is utilized and two different ~~Neighbor~~ Neighborhood Searches are required. Furthermore, a detailed pseudo-code for the LD-SDA is presented ~~below.~~ in the following section (see Algorithm 1). Additional efficiency improvements and other implementation details are presented in §4.

Algorithm 1: ~~Logic-based~~ Logic-Based Discrete-Steepest Descent Algorithm (LD-SDA)

Input: $k \in \{2, \infty\}$; An external variable feasible solution $\mathbf{z_{E,0}}$
Data: Variable values associated with feasible solution $\mathbf{x_0}, \mathbf{Y_0}, \mathbf{z_0}$
/* Initialize */
Set $\mathbf{x} \leftarrow \mathbf{x_0}$; $\mathbf{Y} \leftarrow \mathbf{Y_0}$; $\mathbf{z} \leftarrow \mathbf{z_0}$; $\mathbf{z_E} \leftarrow \mathbf{z_{E,0}}$
Solve subproblem: $f^*_{sub} \leftarrow f_{sub}(\mathbf{z_E})$
Set `neighborSearching` $\leftarrow True$
Generate initialization: $\gamma_{init} \leftarrow \{\mathbf{x}, \mathbf{Y}, \mathbf{z}, \mathbf{z_E}\}$ // Optional
Initialize set of explored point in lattice as $G \leftarrow \{\mathbf{z_E}\}$ // Optional
/* This cycle performs Neighborhood Search either when the algorithm starts (after initialization) or when Line Search does not improve the incumbent solution */
while `neighborSearching` is $True$ **do**
  /* Find the current neighborhood $N_k(\mathbf{z_E})$ and directions $\Delta_k(\mathbf{z_E})$ to execute Neighborhood Search */
  Compute $N_k(\mathbf{z_E}) = \{\alpha \in \mathbb{Z}^{n_{z_E}} : \|\alpha - \mathbf{z_E}\|_k \le 1\}$
  Compute $\Delta_k(\mathbf{z_E}) = \{\mathbf{d} : \alpha - \mathbf{z_E} = \mathbf{d}, \forall\, \alpha \in N_k(\mathbf{z_E})\}$
  /* Perform the Neighborhood Search by evaluating an comparing every $f_{sub}(\alpha)$ */
  f^*_{sub} ; $\mathbf{z_E}$; $\mathbf{d}^*$; `improvedDuringNS` ; $\gamma_{init} \leftarrow$ ~~**Neighbor**~~ **Neighborhood Search**$(f^*_{sub}, \mathbf{z_E}, N_k(\mathbf{z_E}), \Delta_k(\mathbf{z_E}))$
  /* Check for improvement during Neighborhood Search */
  if `improvedDuringNS` is $True$ **then**
    /* If so, perform Line Search in direction $\mathbf{d}^*$ until the incumbent does not improve */
    Set `lineSearching` $\leftarrow True$
    while `lineSearching` is $True$ **do**
      f^*_{sub} ; $\mathbf{z_E}$; `improvedDuringLS` ; $\gamma_{init} \leftarrow$ **Line Search**$(f^*_{sub}, \mathbf{z_E}, \mathbf{d}^*)$
      /* Check if the current solution was not improved during Line Search */
      if `improvedDuringLS` is $False$ **then**
        /* If so, stop Line Search */
        Set `lineSearching` $\leftarrow False$
  else
    /* If not, stop Neighborhood Search to terminate the algorithm and return the solution */
    Set `neighborSearching` $\leftarrow False$
Output: f^*_{sub} ; $\mathbf{x}$; $\mathbf{Y}$; $\mathbf{z}$; $\mathbf{z_E}$

3.3.1 ~~Neighbor~~ Neighborhood Search and Line Search

~~In this subsection we present detailed pseudo-codes and intuitions on implementing the Neighbor Search and the Line Search algorithms as presented in Algorithm ?? and Algorithm 3, respectively.~~

In a general sense, the ~~Neighbor~~ Neighborhood Search algorithm is a local search around the immediate neighborhood of discrete variables from a starting point $\mathbf{z_E}$. Therefore, the neighborhood $N_k(\mathbf{z_E})$ and the set of distances corresponding to each neighbor $\Delta_k(\mathbf{z_E})$ must be computed before starting the exploration. This algorithm solves the subproblems $s(\alpha)\ \forall\ \alpha \in N_k(\mathbf{z_E})$ and compares their objective function; if feasible, with the best incumbent solution found by ~~Neighbor~~ Neighborhood Search f^{NS}_{sub} so far.

~~Neighbor Search~~ The Neighborhood Search algorithm determines whether a new neighbor α improves upon the current solution $\mathbf{z_E}$ based on two criteria, which can be evaluated in relative or absolute terms. The first criterion employs a strict less than ($<$) comparison and is ~~utilized~~ applied when no neighbor has yet ~~improved upon~~ outperformed the current solution. This ensures the algorithm ~~does not transition~~ avoids transitioning to a neighbor with an identical objective ~~, thereby~~ value, preventing cycling between points with ~~identical objective functions. Further discussion~~ the same objective function. Further details on the non-cycling properties of the LD-SDA ~~is provided~~ are discussed in §3.4. ~~The second criterion ,~~

Once a neighbor α improves the incumbent solution, a second criterion (employing a less-than-or-equal-to ($\le$) comparison~~, becomes active once a single neighbor α improves upon the incumbent solution. This enables~~) is utilized. This allows the algorithm to consider multiple neighbors α's with the same objective ~~. Consequently, if~~ value. If more than one neighbor ~~shares the best current~~ achieves the best solution, a tie-breaking strategy based on a maximum Euclidean distance lexicographic heuristic ~~may be employed as a tie-break criterion. This heuristic calculates the Euclidean distance~~ is used. The Euclidean distance is computed as $\texttt{dist} = \|\alpha - \mathbf{z_E}\|_2 \; \forall \; \alpha \in N_k(\mathbf{z_E})$, favoring the first-found "most diagonal" ~~routes. Such routes, absent~~ path. These diagonal routes, which do not exist in N_2 neighborhoods, have proven effective in previous versions of the D-SDA [33, 35, 36].

Algorithm 2: ~~Neighbor~~ Neighborhood Search

Input: f^*_{sub} ; $\mathbf{z_E}$; $N_k(\mathbf{z_E})$; $\Delta_k(\mathbf{z_E})$
/* Initialize */
1 Relative tolerance ϵ
2 Set improvedDuringNS $\leftarrow False$; $\mathbf{d}^* \leftarrow \mathbf{0}$; $f^{NS}_{sub} \leftarrow \infty$
3 Set $\texttt{dist}^* \leftarrow 0$ // Optional
4 **for every** $\alpha_\mathbf{i} \in N_k(\mathbf{z_E})$ **do**
/* Optional: Check if the neighbor was already evaluated in a previous iteration */
5 **if** $\alpha_\mathbf{i} \in G$ **then**
6 Go to line 4 with $\alpha_\mathbf{i+1}$
7 **else**
8 Append $\alpha_\mathbf{i}$ to G
/* Optional: Check if the neighbor is within external variable domain */
9 **if** $\alpha_\mathbf{i} \notin \mathbf{Z_E} = \{\mathbf{1}, \ldots, \overline{\mathbf{z_E}}\}$ **then**
10 Go to line 4 with $\alpha_\mathbf{i+1}$
/* Create fixed subproblem */
11 Create subproblem and fix with external variables $s(\alpha_\mathbf{i})$
12 Initialize $s(\alpha_\mathbf{i})$ using γ_{init} // Optional
/* Optional: Check feasibility of fixed external variables in $s(\alpha_\mathbf{i})$ with FBBT */
13 **if** FBBT of $s(\alpha_\mathbf{i})$ detects infeasibility **then**
14 Go to line 4 with $\alpha_\mathbf{i+1}$
/* Solve subproblem */
15 Solve $s(\alpha_\mathbf{i})$
16 **if** $s(\alpha_\mathbf{i})$ is feasible **then**
17 Set $f^{NS}_{sub} \leftarrow f_{sub}(\alpha_\mathbf{i})$
18 Set $\texttt{dist}_i \leftarrow \|\alpha_\mathbf{i} - \mathbf{z_E}\|_2$ // Optional
/* Check if the algorithm has already improved the starting solution to choose the corresponding minimum improvement criterion */
19 **if** improvedDuringNS is $False$ **then**
/* Check if minimum improvement criterion is satisfied */
20 **if** $f^{NS}_{sub} < f^*_{sub}$ or $(f^*_{sub} - f^{NS}_{sub})/(|f^*_{sub}| + 10^{-10}) > \epsilon$ **then**
/* Update with new best solution */
21 Set $f^*_{sub} \leftarrow f^{NS}_{sub}$; $\mathbf{d}^* \leftarrow \Delta_k(\mathbf{z_E})_\mathbf{i}$; $\mathbf{z_E} \leftarrow \alpha_\mathbf{i}$
22 Set improvedDuringNS $\leftarrow True$
23 Set $\texttt{dist}^* \leftarrow \texttt{dist}_i$ // Optional
24 Generate initialization: $\gamma_{init} \leftarrow \{\mathbf{x}, \mathbf{Y}, \mathbf{z}, \mathbf{z_E}\}$ // Optional
25 **else**
/* Check if minimum improvement criterion is satisfied. There is an additional condition that implements the maximum Euclidean distance heuristic */
26 **if** $(f^{NS}_{sub} \le f^*_{sub}$ or $(f^*_{sub} - f^{NS}_{sub})/(|f^*_{sub}| + 10^{-10}) \ge \epsilon)$ **and** $\texttt{dist}_i \ge \texttt{dist}^*$ **then**
/* Update with new best solution */
27 Set $f^*_{sub} \leftarrow f^{NS}_{sub}$; $\mathbf{d}^* \leftarrow \Delta_k(\mathbf{z_E})_\mathbf{i}$; $\mathbf{z_E} \leftarrow \alpha_\mathbf{i}$
28 Set improvedDuringNS $\leftarrow True$
29 Set $\texttt{dist}^* \leftarrow \texttt{dist}_i$ // Optional
30 Generate initialization: $\gamma_{init} \leftarrow \{\mathbf{x}, \mathbf{Y}, \mathbf{z}, \mathbf{z_E}\}$ // Optional

Output: f^*_{sub} ; $\mathbf{z_E}$; $\mathbf{d}^*$; improvedDuringNS ; γ_{init}

The Line Search algorithm is a search in the steepest descent direction, determined by the direction of the best neighbor $\mathbf{d}^* = \alpha^* - \mathbf{z_E}$. This approach generates a point in the steepest descent direction $\beta = \mathbf{z_E} + \mathbf{d}^*$ and solves the optimization subproblem $s(\beta)$ to obtain f_{sub}^{LS}. The algorithm moves to the point β if and only if, $s(\beta)$ is feasible and $f_{sub}^{LS} < f_{sub}^{*}$, adhering to the strict ~~less-than~~ less than ($<$) improvement criterion. This criterion prevents revisiting previous points, thereby accelerating and ensuring convergence. Again, more insights into the convergence properties of the LD-SDA are stated in §3.4. The Line Search process continues until there is no feasible point in the direction $\mathbf{d}^*$ that improves upon the incumbent solution.

Algorithm 3: Line Search

Input: f_{sub}^{*} ; $\mathbf{z_E}$; $\mathbf{d}^*$

`/* Initialize */`

Relative tolerance ϵ

Set `improvedDuringLS` $\leftarrow False$; $\beta \leftarrow \mathbf{z_E} + \mathbf{d}^*$; $f_{sub}^{LS} \leftarrow \infty$

`/* Optional: Check if the moved point` β `was already evaluated in a previous iteration */`

if $\beta \in G$ **then**

  Terminate algorithm

else

  Append β to G

`/* Optional: Check if the moved point is within the external variable domain */`

if $\beta \notin \mathbf{Z_E} = \{1, \ldots, \overline{\mathbf{z_E}}\}$ **then**

  Terminate algorithm

`/* Create fixed subproblem */`

Create subproblem and fix with external variables $s(\beta)$

Initialize $s(\beta)$ using γ_{init} `// Optional`

`/* Optional: Check feasibility of fixed external variables in` $s(\beta)$ `with FBBT */`

if FBBT of $s(\beta)$ detects infeasibility **then**

  Terminate algorithm

`/* Solve subproblem */`

Solve $s(\beta)$

if $s(\beta)$ is feasible **then**

  Set $f_{sub}^{LS} \leftarrow f_{sub}(\beta)$

  `/* Check if minimum improvement criterion is satisfied */`

  if $f_{sub}^{LS} < f_{sub}^{*}$ or $(f_{sub}^{*} - f_{sub}^{LS})/(|f_{sub}^{*}| + 10^{-10}) > \epsilon$ **then**

    `/* Update with new best solution */`

    Set $f_{sub}^{*} \leftarrow f_{sub}^{LS}$; $\mathbf{z_E} \leftarrow \beta$

    Set `improvedDuringLS` $\leftarrow True$

    Generate initialization: $\gamma_{init} \leftarrow \{\mathbf{x}, \mathbf{Y}, \mathbf{z}, \mathbf{z_E}\}$ `// Optional`

Output: f_{sub}^{*} ; $\mathbf{z_E}$; `improvedDuringLS` ; γ_{init}

3.4 ~~Logic-based~~ Logic-Based Discrete-Steepest Descent Algorithm Properties

The LD-SDA algorithm is guaranteed not to cycle, ~~i.e., it will not re-evaluate~~ meaning it avoids re-evaluating the same solution candidates ~~when searching for the~~ while searching for an optimal solution. This is ~~avoiding revisiting~~ achieved by avoiding revisitation of previously solved subproblems, and ~~it can be accomplished carefully evaluating~~ carefully deciding when to move to ~~a next incumbent . In both Neighbor~~ the next incumbent solution. Both Neighborhood Search (Algorithm ~~??~~2) and Line Search (Algorithm 3) ~~, the algorithms ensure improvement in the solution of the next discrete point by following~~ adhere to a minimum improvement criterion ~~. As established~~ to ensure progress toward a better solution. As discussed in §3.3.1, this criterion is satisfied if and only if a strict less than ($<$) improvement is obtained. ~~Consequently, if this criterion~~ Once this is met, the algorithms update the incumbent with the ~~new best solutiondiscovered during the searches, ensuring~~ newly found solution, ensuring that only strictly better solutions are ~~considered. It is important to note that this criterion~~ accepted. An important aspect of this approach is that it excludes points with ~~the same objective value~~ identical objective values as the incumbent, ~~thereby~~ effectively preventing cycling between points with ~~identical objectives~~the same objective. By avoiding ~~revisiting~~ the reevaluation of points in the lattice, ~~the algorithms prevent the steepest descent from retracing its steps in either the Neighbor or Line Search steps. This verification not only~~ both Neighbor and Line Search avoid retracing steps, which guarantees convergence to a discrete local minimum ~~but also saves computational time , as the algorithm never re-evaluates the~~ while also saving computational time by not re-evaluating the same point.

The primary advantage of the LD-SDA over previous iterations of the D-SDA ~~lies in its utilization of~~ in how it leverages the structure of ordered Boolean variables for external variable reformulation, ~~as opposed to~~ rather than ordered binary variables. In the LD-SDA, the solution to the upper-level problem is the same as the one used in the D-SDA, involving a series of Neighbor and Line searches over the external variable lattice. However, the key distinction is that each lattice point in the LD-SDA upper-level problem corresponds to a reduced space GDP or (MI)NLP, obtained by fixing ~~Booleans~~ Boolean variables, and thereby disjunctions (see §3.2). This ~~approach results in~~ leads to a reduced subproblem that ~~considers only~~ only considers relevant constraints, ~~effectively circumventing~~ avoiding zero-flow issues and improving both numerical stability and computational ~~tractability~~efficiency. In contrast, previous versions of the D-SDA fixed binary variables to obtain NLP subproblems, ~~which~~ that could potentially contain irrelevant constraints with respect to the current configuration of the Boolean variables yielding and ill-posed ~~behavior~~problem. Furthermore, additional algorithmic improvements with respect to previous versions of the D-SDA were added to the LD-SDA as discussed in §4.

3.5 Equivalence to Other Generalized Disjunctive Programming Algorithms

While LD-SDA exhibits different features compared to other GDP algorithms, certain aspects of it remain equivalent to them. Notably, akin to other logic-based approaches, LD-SDA addresses (MI)NLP subproblems containing only the constraints of active disjunctions, thereby excluding irrelevant nonlinear constraints. Each method employs a mechanism for selecting the subsequent (MI)NLP subproblem, typically based on a search procedure. In LOA, this mechanism involves solving a MILP problem subsequent to reformulating Problem (Main l-GDP). On the other hand, LBB determines a sequence of branched disjunctions for each ~~layer~~ level l, KB_l, based on a predetermined rule known as *branching rule*. In contrast, LD-SDA utilizes Neighbor and Line Search algorithms to make this decision, solving Problem (Upper) locally.

The LD-SDA employs an external variable reformulation to map Boolean variables into a lower-dimensional representation of discrete variables. While LD-SDA solves the upper-level problem through steepest descent optimization, this problem essentially constitutes a discrete optimization problem without access to the functional form of the objective. Hence, in principle, this problem could be addressed using black-box optimization methods. ~~Moving~~

When transitioning from one point to another ~~within~~ in the discrete external variable lattice involves changing the configuration of Boolean variables in the original problem, often modifying multiple Boolean variables simultaneously. Consequently, the LD-SDA can be viewed as a variant of LBB, where Neighbor and Line Searches act as sophisticated branching rules for obtaining Boolean configurations to fix and evaluate. ~~Additionally~~Furthermore, improvements to this problem could be achieved by leveraging information from the original GDP problem. For instance, linear approximations of the nonlinear constraints of the GDP could be provided, although this would necessitate employing ~~a~~ an MILP solver. By constructing such linearizations around the solutions of the subproblem (7), one could recover Problem (Main l-GDP) from LOA.

4 Implementation Details

The LD-SDA, as a solution method for GDP, was implemented in Python using Pyomo [40] as an open-source algebraic modeling language. Pyomo.GDP [41] was used to implement the GDP models and use their data structures for the LD-SDA. The code implementation allows the automatic reformulation of the Boolean variables in the GDP into external variables and provides an efficient implementation of the search algorithms over the lattice of external variables.

4.1 Automatic Reformulation

In contrast to previous works for MINLP models [33], the reformulation in (3) and (4) provides a generalized framework that is automated in the Python implementation developed in this work. Minimal user input is required for the reformulation process, with only the Boolean variables in $\mathbf{Y}$ defined over ordered sets $S_j, \forall\, j \in \{1, 2, ..., n_R\}$ needing specification. This reformulation allows fixing Boolean variables based on the values of *external variables*. Moreover, additional Boolean variables can be fixed based on the values of the external variables, as users can specify those Boolean variables in $\mathbf{Y_N}$ that are equivalent to expressions of the independent Boolean variables $\mathbf{Y_R}$ through logic constraints $\Omega(\mathbf{Y}) = True$.

4.2 Algorithmic Efficiency Improvements

This section presents the four major efficiency improvements that are included in the algorithm and are indicated throughout the pseudo-codes in §3.3 as `Optional`.

4.2.1 Globally Visited Set Verification

Due to the alternating dynamic between Line Search and ~~Neighbor~~ Neighborhood Search, the LD-SDA often queues discrete points that were previously visited and evaluated. An example of this issue can be observed in Figure 3 where the second ~~Neighbor Search in $N_\infty([5,5])$~~Neighborhood Search in $N_\infty((5,5))$, depicted in red, visits points ~~$[4,4]$ and $[6,6]$~~ $(4,4)$ and $(6,6)$ that had already been evaluated during Line Search (shown in black). The number of ~~reevaluated~~ re-evaluated points depends on how close to the ~~Neighbor~~ Neighborhood Search the Line Search stops, increasing proportionally with the number of external variables.

Although re-evaluating points does not affect the convergence of the algorithm as discussed in §3.4, it results in unnecessary additional computation that can be avoided. This redundant evaluation existed in the previous versions of the D-SDA [33, 35, 36] and can be rectified by maintaining a globally visited set G (line 5 of Algorithm 1). ~~Now~~Furthermore, before solving the optimization model for a particular point α (lines 5 to 8 of Algorithm ~~??~~2) or β (lines 3 to 6 of Algorithm 3), the algorithm verifies if the point has already been visited. If so, the algorithm disregards that point and either proceeds to the next α in the ~~Neighbor~~ Neighborhood Search or terminates the Line Search algorithm.

4.2.2 External Variable Domain Verification

All external variables must be defined over a constrained box $\mathbf{Z_E} = \{\mathbf{1}, \dots, \overline{\mathbf{z_E}}\}$ (as shown in Eq. (4)) that depends on the problem. For superstructure problems, this domain is bounded by the size of the superstructure, such as the number of potential trays in a distillation column, or the maximum number of available parallel units in a process. Similarly, for scheduling problems, the external variable domain can be given by the scheduling horizon.

External variables with non-positive values or exceeding the potential size of the problem, resulting in a lack of physical sense, should not be considered in the explorations. To prevent unnecessary presolve computations, the algorithm verifies if the incumbent point (α or β) belongs to $\mathbf{Z_E}$ before solving the optimization model, effectively avoiding consideration of infeasible subproblems. If during ~~Neighbor~~ Neighborhood Search $\alpha \notin \mathbf{Z_E}$, the neighbor α can be ignored, and the algorithm proceeds to explore the next neighbor. Similarly, if $\beta \notin \mathbf{Z_E}$ while performing the Line Search, the algorithm should return to $\mathbf{z_E} = \beta - \mathbf{d}^*$ and terminate. Returning to the example shown in Figure 3, note that, for instance, ~~$N_\infty([1,6]) = \{[2,6],[2,5],[1,5]\}$~~ $N_\infty((1,6)) = \{(2,6),(2,5),(1,5)\}$ given that points ~~$\{[1,7],[2,7],[0,5],[0,6],[0,7]\}$~~ $\{(1,7),(2,7),(0,5),(0,6),(0,7)\}$ can be automatically discarded and considered infeasible since $\mathbf{z_{E,1}}, \mathbf{z_{E,2}} \in \{1,2,...,6\}$.

4.2.3 Fixed External Variable Feasibility Verification via FBBT

The existence of external variables $\mathbf{z_E}$ within their respective bounds does not ensure feasibility in the subproblem $s(\mathbf{z_E})$. While external variables can encode a physical interpretation of the problem by representing specific positions within a well-ordered set, constraints concerning the rest of the problem must align with spatial information to achieve a feasible subproblem. For instance, consider the distillation column (discussed in §5.2) that has two external variables: one determining the reflux position $z_{E,R}$ and another determining the boil-up position $z_{E,B}$. The problem has an implicit positional constraint $z_{E,B} < z_{E,R}$, indicating that the boil-up stage must be above the reflux stage when counting trays from top to bottom.

Throughout the algorithm, this type of discrete positional constraint, which relates external variables, is frequently violated when a particular $\mathbf{z_E}$ is fixed in a subproblem $s(\mathbf{z_E})$. This violation arises because these constraints are specified in the original GDP model in terms of Boolean variables. Consequently, after the external variable reformulation, fixed points in the discrete lattice may overlook the original logical constraints.

In previous works [33, 35, 36], users were tasked with manually re-specifying these constraints in the domain of external variables. However, this work aims to automate this requirement. Instead of solving infeasible models that consume computation time and may generate errors ~~terminating~~ that terminate the algorithm, we used ~~Feasibility-based Bound Tightening (FBBT), which is~~ the Feasibility-Based Bound Tightening routine available in Pyomo. FBBT rapidly verifies feasibility over the fixed Boolean constraints, enabling the algorithm to identify subproblem infeasibility without executing a more resource-intensive MINLP or GDP presolve algorithm. Now, if FBBT determines that a subproblem $s(\mathbf{z_E})$ is infeasible, the point $\mathbf{z_E}$ can be instantly disregarded.

4.2.4 ~~Re-initialization~~ Re-Initialization Scheme

The LD-SDA method incorporates an efficiency improvement that involves reinitializing from the best solution $\gamma_{init} = \{\mathbf{x}, \mathbf{Y}, \mathbf{z}, \mathbf{z_E}\}$. Effective model initialization is crucial for achieving faster convergence, particularly as problems increase in size and complexity. Initiating a discrete point with the solution of a neighboring point is

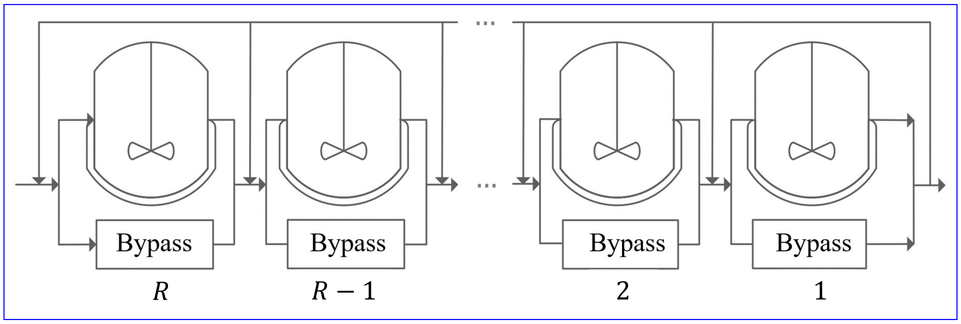


Figure 4: Reactor series Visualization of a superstructure with N_T consisting of R potential continuously stirred reactor tanks (CSTRs). The reactors are numbered starting from the product stream and counted in reverse. At each position, a reactor can either be present or replaced by a bypass. The configuration must be continuous, meaning no bypasses are allowed between two active reactors.

intuitively reasonable. Since points in the external variable lattice are derived from Boolean configurations following ordered sets, adjacent points are expected to yield very similar subproblems (e.g., adding an extra tray in a distillation column or starting a process one time step later). Therefore, initializing from an adjacent neighbor can offer an advantage of discrete-steepest descent optimization over black-box methods that search the lattice.

During Neighbor Neighborhood Search with $\alpha \in N_k(\mathbf{z_E})$, all subproblems are initialized using the solved variable values from the best incumbent solution $s(\mathbf{z_E})$. Similarly, in Line Search each subproblem from the moved point $s(\beta)$ is initialized with the variable values of the best incumbent solution $s(\mathbf{z_E} = \beta - \mathbf{d}^*)$. This reinitialization re-initialization methodology proved very efficient when integrated into the MINLP D-SDA in the rigorous design of a catalytic distillation column using a rate-based model [36].

5 Results

The LD-SDA is implemented as an open-source code using Python. The case studies, such as reactors, chemical batch processing, and binary distillation column design, are modeled using Python 3.7.7 and Pyomo 5.7.3 [40]. The catalytic distillation column design case study was modeled using GAMS 36.2.0. All the solvers used for the subproblems are available in that version of GAMS and were solved using a Linux cluster with 48 AMD EPYC 7643 2.3GHz CPU processors base clock frequency and 1.0 TB RAM. Although the Neighborhood Search can be trivially parallelized, this study limited experiments to a single thread. All the codes are available at `https://github.com/SECQUOIA/dsda-gdp`. The solvers used for the MINLP optimization are BARON [53], SCIP [54], ANTIGONE [55], DICOPT [56], SBB [57], and KNITRO [58]. KNITRO, BARON, and CONOPT [59], and IPOPT [60] are used to solve the NLP problems. The GDP reformulations and algorithms are implemented in GDPOpt [41].

5.1 Series of Continuously Stirred Tank Reactors (CSTRs)

Consider a reactor network adapted from [33], consisting of a superstructure of N_T R reactors in series (depicted in Figure 4), where N_T R represents the total number of potential reactors to install. The objective is to minimize the sum of reactor volumes. The network involves an autocatalytic reaction $A + B \rightarrow 2B$ with a first-order reaction rate, along with mass balances and reaction equations for each reactor. Logical constraints define the recycle flow location and the number of CSTRs in series to install. All installed reactors must have the same volume and a single recycle stream can feed any of them. Interestingly, as the number of reactors increases and the recycle is placed in the first reactor, the system approximates to a plug-flow reactor, minimizing the total volume and providing an asymptotic analytical solution. We investigate this feature by varying the number of potential reactors N_T R. For a detailed formulation of the reactor series superstructure, refer to Appendix Supplementary Material A.1.

The external variables $\mathbf{z_E} = [z_{E,1}$: No. of reactors (related with YF), $z_{E,2}$: Recycle position (related with YR)] $\mathbf{z_E} = (z_{E,1}$: No. of reactors (related with $\mathbf{YF}$), $z_{E,2}$: Recycle position (related with $\mathbf{YR}$)) as shown in Figure 5 are the result of a complete reformulation of logic variables into external integer variables (detailed in Appendix Supplementary Material B.1). This figure shows the binaries associated with the values of the ordered Boolean variables and their corresponding external variable mapping for an illustrative feasible solution, effectively indicating the reformulation $YF_4 = True \Leftrightarrow z_{E,1} = 4$ and $YR_2 = True \Leftrightarrow z_{E,2} = 2$.

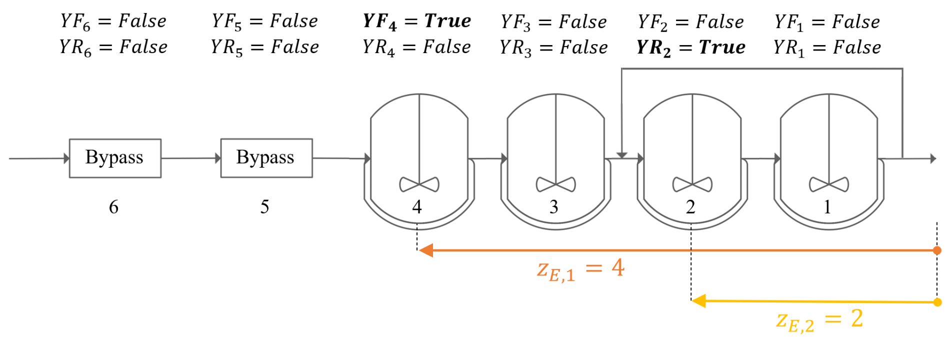


Figure 5: Reformulation Visualization of a potential configuration for an illustrative feasible the CSTR superstructure with $R = 6$. The figure explicitly displays the Boolean variables for the position of the feed ($\mathbf{YF}$) and recycle ($\mathbf{YR}$) streams. In this configuration, the feed enters at the fourth reactor, meaning $YF_4 = True$, indicating the presence of four reactors in the superstructure. The recycle stream is positioned before the second reactorseries with $N_T = 6$, meaning $YR_2 = True$.

We analyze the paths and solutions generated by the LD-SDA under varying neighborhood selections, as depicted in Figure 6. Given N_T R series of potential reactors, the problem is initialized with one reactor and its recycle flow. This initialization is represented in the integer variables lattice as a single reactor with a reflux position immediately behind it ($\mathbf{z_E} = [1, 1]$). $\mathbf{z_E} = (1, 1)$).

For LD-SDA employing a $k = 2$ neighborhood search, the algorithm identifies $[2, 1]$ $(2, 1)$ is locally optimal and proceeds with the line search in the $\mathbf{d}^* = [1, 0]$ $\mathbf{d}^* = (1, 0)$ direction. The algorithm continues the line search until $[5, 1]$ as $[6, 1]$ $(5, 1)$ as $(6, 1)$ exhibits a worse objective. It searches among its neighbors, eventually moving and converging to the a local optimal solution $[5, 1]$$(5, 1)$. In contrast, with LD-SDA utilizing a $k = \infty$ neighborhood search, the algorithm finds that both $[2, 1]$ and $[2, 2]$ $(2, 1)$ and $(2, 2)$ yield the best solution within the first neighborhood explored. Employing the maximum Euclidean distance heuristic as a tie-breakcriterion selects $[2, 2]$, this criterion selects $(2, 2)$ as the new incumbent. Subsequently, a line search in the steepest direction $\mathbf{d}^* = [1, 1]$ $\mathbf{d}^* = (1, 1)$ proceeds until reaching $[N_T, N_T]$$(R, R)$, representing the global optimal solution as it approximates the plug-flow reactor.

For a reactor series superstructure with $N_T = 30$$R = 30$, we performed external variable reformulation and fully enumerated the discrete points in a 30×30 lattice. Notably, only the lower right lower-right triangle of the lattice is depicted, as points outside this region yield infeasible Boolean configurations. More specifically, these points indicate superstructures that have their recycle previous to an uninstalled reactor. Local optimality was verified with both neighborhoods, revealing local minima for $k = 2$ neighborhood at $[5, 1]$, $[5, 3]$ and $[N, N]$ $(5, 1)$, $(5, 3)$ and (r, r) $\forall$ $N \in \{5, \ldots, 29\}$$r \in \{5, \ldots, 29\}$, whereas the only locally optimal point for $k = \infty$ was $[30, 30]$$(30, 30)$. Figure 6 illustrates the LD-SDA process for the 30 CSTR series, showing trajectories and local minimum points for all neighborhoods. The presence of multiple local optima with respect to both the 2-neighborhood and the ∞-neighborhood suggests that this problem is neither separably convex, nor integrally convex.

For the CSTR series, various solver approaches are applied across different numbers of potential reactors (N_T R ranging from 5 to 30). The solution approaches include MINLP reformulations, LBB, LOA, GLOA, and LD-SDA with two different neighborhoods. Figure 7 illustrates the comparison of solution times for each reactor superstructure size with different solvers. Notably,

LD-SDA with $k = 2$ neighborhoods failed to achieve optimal solutions within 0.1% of the global optimum for any superstructure size, converging instead to the $[5, 1]$ $(5, 1)$ solution, as explained above. In contrast, methods that employed $k = \infty$ attained the global minimum. KNITRO was computationally more efficient than BARON when using $k = \infty$, as BARON, being a global solver, incurred higher in greater computational costs certifying global optimality for each NLP subproblem. InterestinglyRemarkably, even when using local solvers like KNITRO for the subproblem, $k = \infty$ allowed LD-SDA to converge to global optimal solutions.

Among the logic-based methods in GDPopt, LBB for $N_T = 5$ $R = 5$ achieved the global optimum. GLOA reached globally global optimal solutions up to $N_T = 14$ $R = 14$, but only when paired with the global NLP solver BARON. Comparing MINLP reformulations, HR outperformed BM, with KNITRO being the most efficient subsolver. While some MINLP reformulations exhibited faster solution times for smaller superstructures (up to 9 nine reactors), LD-SDA, particularly with KNITRO, surpassed them for larger networks (from 15 reactors onwards). This trend suggests

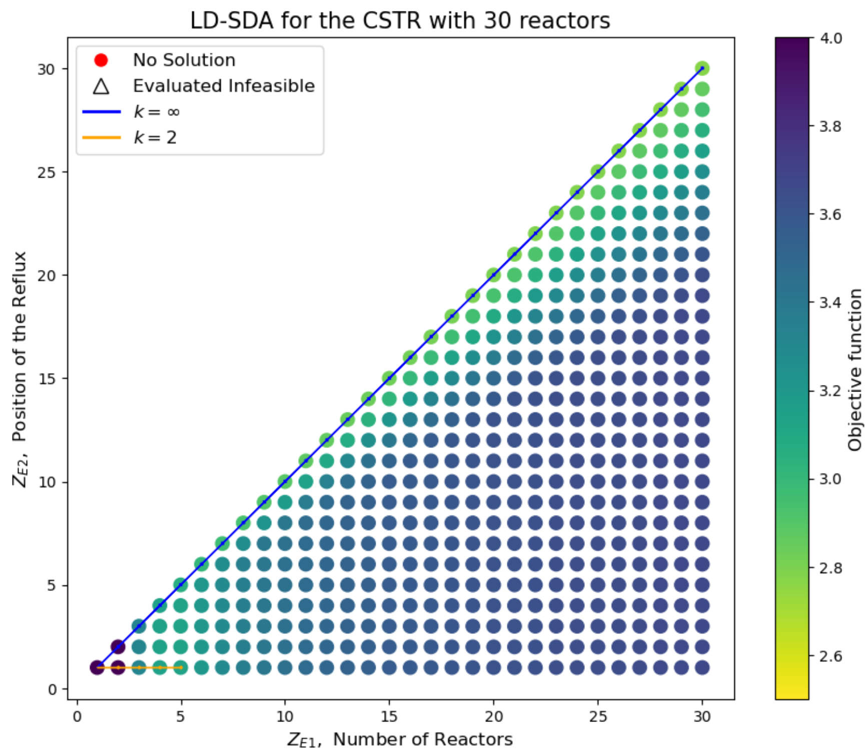


Figure 6: ~~Paths~~ Visualization of the paths traversed by LD-SDA and the solutions found ~~by the LD-SDA~~ using both neighborhoods for a ~~$N_T = 30$ CSTR series;~~ superstructure with $R = 30$ CSTRs. LD-SDA with ~~$k = 2$~~ N_2 converged ~~on [5, 1] which is the local optimal point~~ to $(5, 1)$, a discrete s-local optimum in the lattice~~;~~. In contrast, LD-SDA with ~~$k = \infty$ proceeded~~ N_∞ continued to ~~[30, 30]~~ $(30, 30)$, achieving the global optimal solution. This figure highlights the difference in convergence behavior between the two neighborhood strategies.

that LD-SDA methods are particularly well-suited for solving larger optimization problems, where solving reduced subproblems offers a significant advantage over monolithic GDP-MINLP approaches.

Figure 8 compares different algorithmic alternatives derived from LD-SDA. These include the algorithm discussed so far (referred in this example as NLP LD-SDA) where Boolean variables are fixed from external variables, leading to NLP subproblems considering only relevant constraints. Another approach, which we refer to as MIP LD-SDA, is where inactive disjunctions are retained in subproblems, and mixed-binary reformulations (e.g., HR or BM) are applied to unresolved disjunctions, resulting in MINLP subproblems. The third alternative is Enumeration, which involves reformulating external variables, fixing (or not) Boolean variables, and enumerating all lattice points instead of traversing them via steepest descent optimization.

LD-SDA and Enumeration methods exhibited faster performance when the mixed-binary reformulation was omitted. The inclusion of MIP transformations led to additional solution time, emphasizing the efficiency of solving GDP problems directly where reduced subproblems with solely relevant constraints are considered. As anticipated, the Enumeration of external variables coupled with ~~a proficient~~ an efficent local solver like KNITRO achieved the global optimum. However, employing LD-SDA yielded the same result in significantly less time, showcasing the importance of navigating the lattice intelligently, like via discrete-steepest descent.

Among the LD-SDA approaches, 2-neighborhood search methods were only effective when the tolerance gap between solutions was 10%, while ∞-neighborhood search methods performed consistently across both gap thresholds. The LD-SDA using $k = 2$ neighborhood converges to a local minimum ~~, which~~ that is more than 0.1% away from the global optimal solution~~and,~~ , and for larger instances, is beyond the 10% optimality gap. Although LD-SDA with the $k = \infty$ neighborhood search required more time compared to $k = 2$, it consistently converged to the global optimal point regardless of the superstructure size.

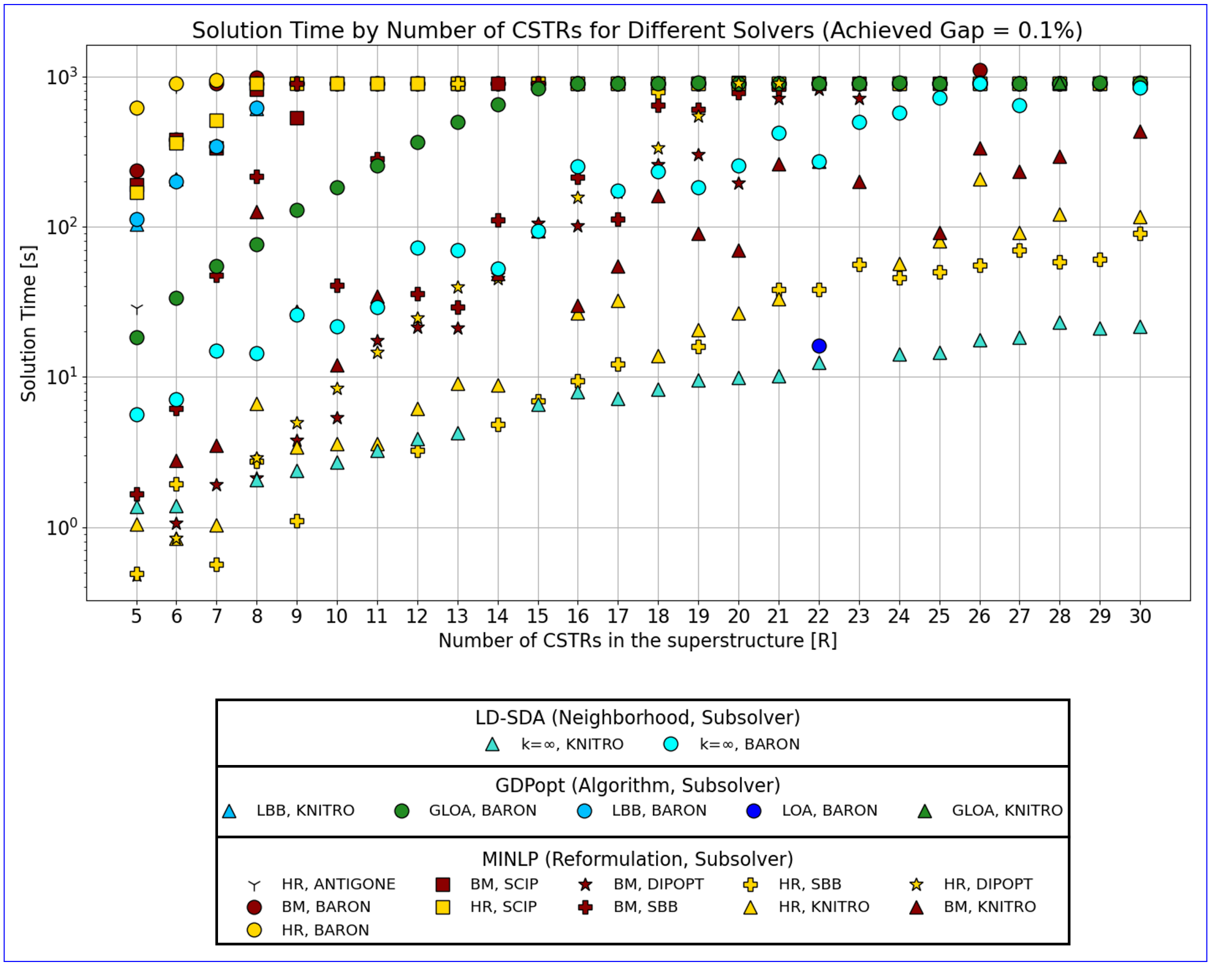


Figure 7: Computational time solution times for different sizes of reactor network N_T for various GDP solution strategies and solvers as the size of the superstructure R increases. Solutions The figure includes only accounted for being the analytically guaranteed solutions that achieved a global optimal solution (equivalent optimum, which corresponds to the installing N_T reactors since the minimum total volume corresponds to $PFR = \lim_{N_T \to \infty} CSTR$)of the superstructure, analytically derived as the volume of a plug flow reactor as $R \to \infty$. It can be observed that LD-SDA with N_∞ using KNITRO consistently reaches the global optimum the fastest for instances with more than 15 CSTRs.

5.2 Distillation Column Design for a Binary Mixture

We consider the single-unit operation design of an example distillation column in Ghouse, Chen, Zamarripa, Lee, Burgard, Grossmann, and Miller [61], which that implements the simplified model provided by Jackson and Grossmann [62]. The objective is to design a distillation column to separate Toluene and Benzene while minimizing cost. There is , which include both a fixed cost associated with for tray installation and an operational cost related to operational costs for the condenser and reboilerduties. The feed conditions are .

The column processes 100 molgmol/s of an equimolar benzene-toluene mixture, with aiming to achieve a minimum mole fraction of 0.95 for benzene in the distillate and 0.95 for toluene in the bottom . The constraints are the product. To meet these requirements, the design must satisfy mass, equilibrium, summation, and heat (MESH) equations for each tray. Each column stage is modeled with thermodynamic stage of the column is modeled using thermodynamic principles and vapor-liquid equilibriumvia , applying Raoult's law and Antoine 's equation. equation.

The continuous variables of this model are in the model include the flow rates of each component in both the liquid and vapor phase and the temperatures in phases, the temperatures of each tray, the reflux and boil-up ratioratios, and the heat duties of the condenser and reboilerheat duties. The logical variables are account for the existence of trays and the position of the positions of reflux and boil-up flows. Furthermore, the existence of trays expressed only in terms of the position of the reflux and boil-up flows can be posed with a logical constraint. The reformulation of the Boolean

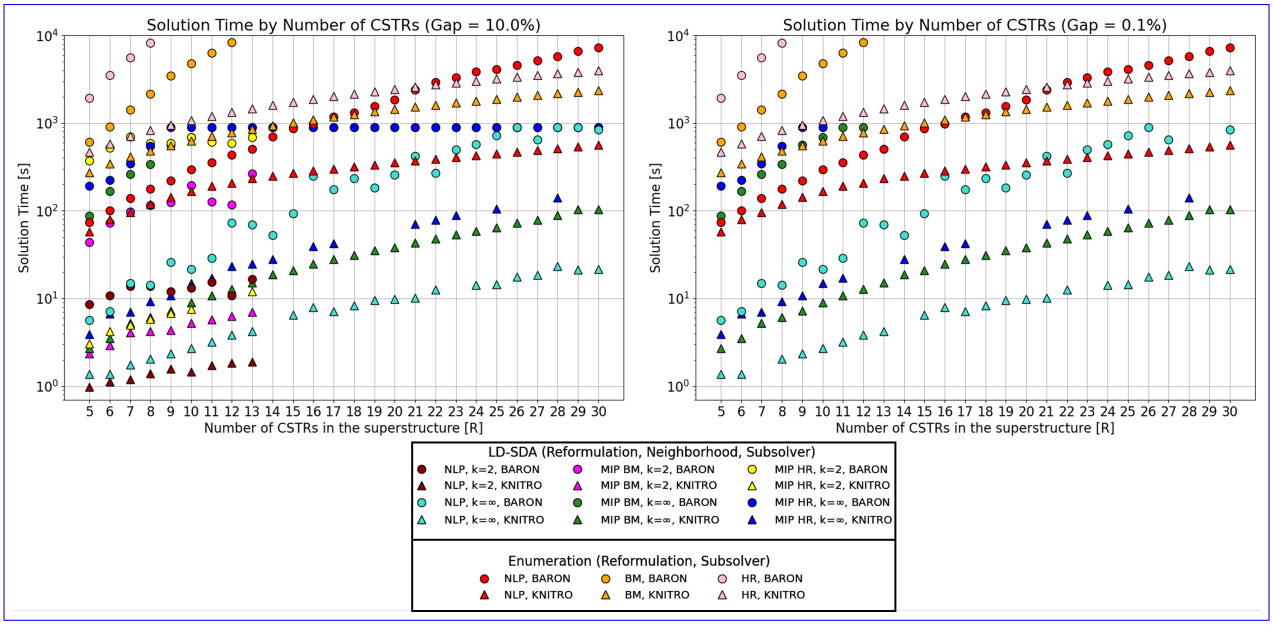


Figure 8: ~~Solution~~ Computational solution times for ~~CSTR superstructures using~~ different solution approaches derived from LD-SDA ~~and Enumeration~~ (including external variable enumeration) using various solvers as the size of the superstructure R increases. The left subfigure shows methods ~~at different~~ that achieved a discrete local minimum with a 10% optimality gap~~: (left) 10%~~, ~~(~~while the right ~~) 0.1%)~~subfigure shows methods that reached an analytically proven global optimum. LD-SDA with N_2 and the KNITRO solver is the fastest method for smaller superstructures but fails to meet the 10% optimality gap for structures larger than 13 CSTRs. For larger superstructures, LD-SDA with N_∞ and KNITRO consistently achieves a global optimum and becomes the fastest approach among LD-SDA methods.

~~variables to external variables is described in the figure shown in Appendix B.2~~, with tray existence modeled using logical constraints related to these flow positions. Previous studies from the literature ~~-~~[61] set a maximum number of 17 potential trays and provide the initial position of the feed tray in the ninth stage (tray ~~number 9~~nine).

The distillation column optimization uses the LD-SDA method with different neighborhoods for the search. The problem is initialized with a column that has the reflux at tray 16 (top to bottom numbering, condenser being tray 17 and the reboiler being tray 1) and the boil-up at the second tray, which we represent as ~~[15, 1]~~(15, 1). The configuration for initialization is shown in Figure 9a, which corresponds to all possible trays being installed.

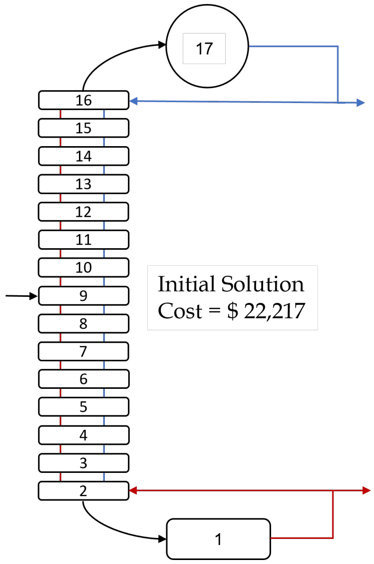
For the 2-neighborhood, the LD-SDA converges to external variable configuration ~~[12, 3]~~(12, 3), with an objective value of \$19,450 in only 6.3 seconds using KNITRO as subsolver, resulting in the design shown in Figure 9b. Note this is the exact same solution reported by the GDP model from the literature [61], ~~which~~ that was solved using the LOA method.

Regarding the ∞-neighborhood, the algorithm terminates at ~~[13, 4]~~ (13, 4) with an objective of \$19,346 after 8.6 seconds using KNITRO as subsolver, yielding the column design shown in Figure 9c. In this case, the LD-SDA found the best-known solution to this problem, also found through a complete enumeration over the external variables, ~~which~~ that took 42.7 seconds using KNITRO as a subsolver. The same best-known solution could be found using GLOA with KNITRO as the NLP subsolver, but after 161.6 seconds. In our results of the binary mixture distillation column design, we successfully identified a better solution by applying the LD-SDA to the GDP model, surpassing the optimal values previously ~~documented~~ reported in the literature. This achievement highlights the efficacy of our approach, especially considering the limitations of the NLP formulation in guaranteeing global optimality, which we effectively navigated by employing the GDP framework.

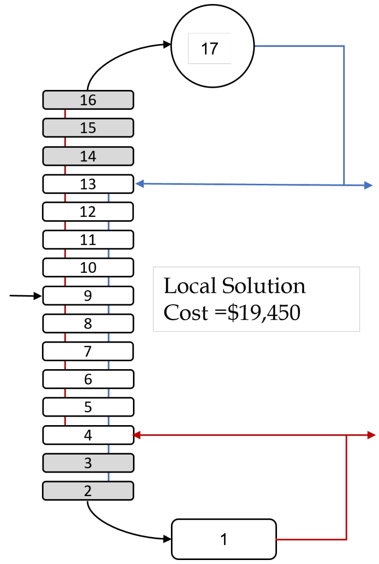
The trajectories traversed by the LD-SDA with both neighborhoods mentioned are depicted in Figure 10. Similarly, Table 1 summarizes the previous design from the literature as well as the different columns obtained by the LD-SDA.

5.3 Catalytic Distillation Column Design

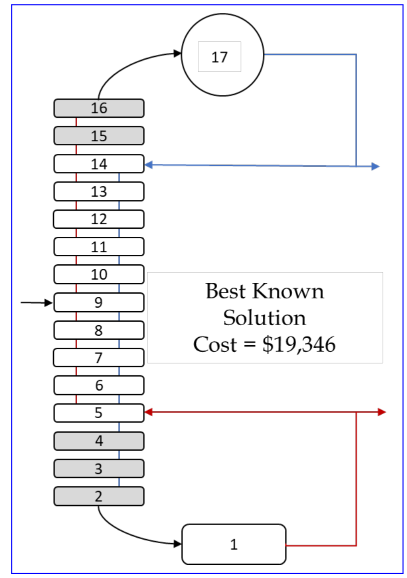
Consider a catalytic distillation column design for the production of Ethyl tert-butyl ether (ETBE) from isobutene and ethanol. In this work, two models are considered: one that uses equilibrium-based modeling in each of the separation



(a) ~~Initial solution for the~~ Distillation ~~Column~~column configuration of the initial solution.



(b) Distillation ~~Column Solution by~~ column configuration obtained with LD-SDA ~~$k = 2$. Reference~~ and N_2, which matches the solution ~~by~~ reported in [61] ~~using GDP model~~.



(c) Distillation ~~Column Solution by~~ column configuration obtained with LD-SDA ~~$k = \infty$~~and N_∞, which yields a new best solution for the problem.

Figure 9: ~~Distillation Column design comparison between~~ Visualization of the distillation columns obtained from the initial solution, ~~local~~the solution found using LD-SDA with N_2, and ~~global solutions~~the solution using LD-SDA with N_∞. The feed tray is fixed at stage nine, with existing trays displayed in white and bypassed trays shaded in gray. The boil-up position is highlighted in red, while the reflux position is marked in blue. Each subfigure shows its corresponding objective function.

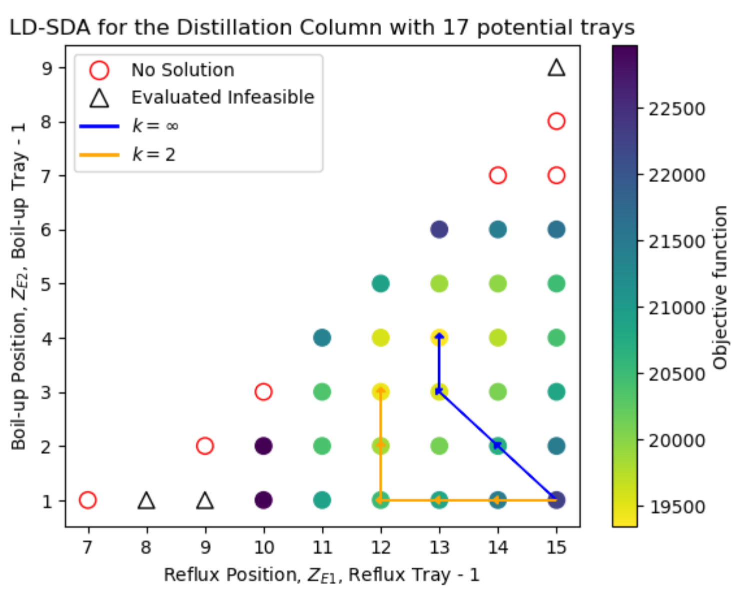


Figure 10: ~~Algorithmic steps~~ Visualization of the paths traversed by LD-SDA ~~when solving~~ and the solutions found using both neighborhoods for the distillation column ~~example~~superstructure. The lattice shows points proven to be infeasible with a global solver (black triangle) and points where no feasible solution was found before timing out (red circle). Both solutions were initialized with the largest column configuration. LD-SDA with $k = 2$ converged to $(13, 3)$, a discrete s-local optimal point in the lattice. Meanwhile, LD-SDA with $k = \infty$ advanced to $(13, 4)$, yielding the best-known solution for the problem.

Table 1: Comparison of the optimal ~~solution with~~ solutions found in the literature~~results~~ , using LD-SDA with N_2 and LDSDA with N_∞. The LD-SDA with N_2 identified the same configuration as previously reported in the literature (with a minor numerical difference in the objective). In contrast, the LD-SDA with N_∞ discovered a new best solution, effectively improving upon the existing results in the literature. This table illustrates the improvements gained by using more expansive neighborhoods in LD-SDA.

Solution Method	**LOA ~~[61]~~Ghouse et al. [61]**	**LD-SDA k = 2**	**LD-SDA k = ∞**
Objective [$]	19,450	19,449	19,346
Number of Trays	10	10	10
Feed Tray	6	6	5
Reflux ratio	2.45	2.45	2.01
Reboil ratio	2.39	2.39	2.00

Table 2: ~~Execution time~~ Comparison of ~~catalytic distillation~~ the optimal ~~design~~ solutions and respective computational times using KNITRO for the catalytic distillation column and rate-based ~~model presented~~ catalytic distillation column case studies, using D-SDA (from the literature) and LD-SDA with both N_2 and N_∞. For the catalytic distillation column case, both neighborhoods in ~~Liñán, Bernal, Gómez, and Ricardez-Sandoval [36]~~ LD-SDA successfully found the same optimal solution as D-SDA but in one-third of the computational time. In the rate-based catalytic distillation column case, D-SDA was unable to find a solution, while LD-SDA, using ~~KNITRO~~both neighborhoods, found distinct optimal solutions.

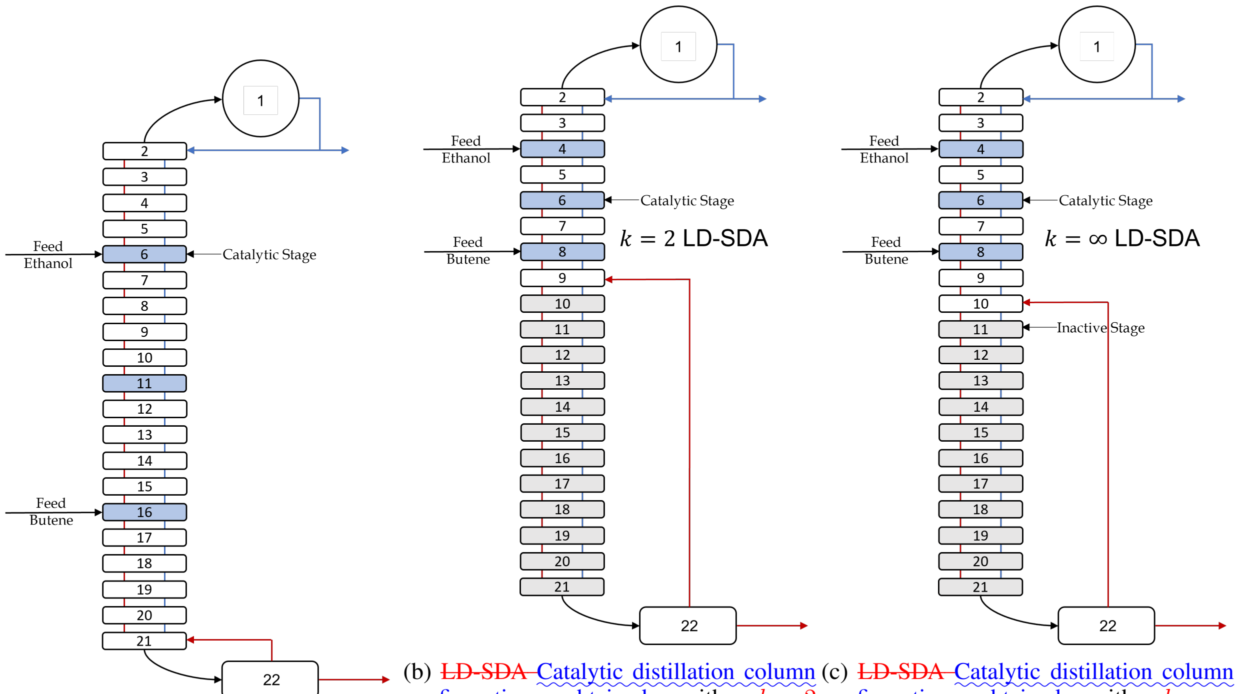
	Catalytic Distillation Column				**Rate-Based Catalytic Distillation Column**			
Solution Method	D-SDA: [36]		LD-SDA: This work		D-SDA: [36]		LD-SDA: This work	
Neighborhood	$k=2$	$k=\infty$	$k=2$	$k=\infty$	$k=2$	$k=\infty$	$k=2$	$k=\infty$
Objective [$/year]	22,410	22,410	22,410	22,410	–	–	23,443.2	23,443.2
Time [s]	12.49	12.52	4.29	4.25	–	–	1089.31	1061.18

and reactive stages, and another one that includes a rate-based description of the mass and energy transfer in all the stages [35]. These models maximize an economic objective by determining the position of separation and catalytic stages along the column, together with a Langmuir-Hinshelwood-Hougen-Watson kinetic model for the chemical reaction, MESH equations for each one of the stages, and hydraulic constraints for the column operation. The goal is to determine optimal operational variables such as reboiler and condenser heat duties and reflux ratio. Similarly, design variables such as column diameter, tray height, and downcomer specifications need to be defined. Finally, discrete design choices, meaning feed locations and positions of catalytic stages, must be selected. A detailed description of the models is given in [36, 63].

Previously in the literature, the economic annualized profit objective maximization of a catalytic distillation column to produce ETBE from butenes and ethanol was solved using a D-SDA [36]. Here, the authors demonstrated the difficulty of this design problem as several traditional optimization methods fail ~~even to obtain~~ to obtain even a feasible solution [33, 35]. In these papers, the D-SDA was used to solve the problem as ~~a~~ an MINLP by fixing binary variables and including constraints of the form $y_{ik}\mathbf{h}_{ik}(x) \leq 0$ to enforce the logic constraints. In this work, we demonstrate that approaching the problem disjunctively and employing LD-SDA leads to a faster solution of subproblems (as in Eq. (Sub)) as our method neglects the irrelevant and numerically challenging nonlinear constraints.

~~These~~ The models were implemented in GAMS~~, hence,~~. Hence, for this problem, the reformulation and implementations of the algorithms were custom-made, as they did not rely on our implementation of LD-SDA in Python. Given that only the relevant constraints were included for each problem, we could more efficiently obtain the same solution to each subproblem. More specifically, as shown in Table 2, the proposed LD-SDA method leads to speedups of up to ~~3x~~ three-fold in this problem when using KNITRO as a subsolver. D-SDA was unable to even initialize the rate-based catalytic distillation column with KNITRO, while LD-SDA could find ~~the~~ an optimal solution. Moreover, note that the previous results using the D-SDA were already beating state-of-the-art MINLP solution methods, further demonstrating the advantages of the LD-SDA.

An important distinction for the LD-SDA is that it does not include all the constraints in each iteration, given that subproblems are reduced after the disjunctions are fixed. This implies that not all variables are present in all iterations,



(a) Catalytic column superstructure.

(b) ~~LD-SDA~~ Catalytic distillation column configuration obtained with ~~$k = 2$~~ ~~solution~~LD-SDA and N_2.

(c) ~~LD-SDA~~ Catalytic distillation column configuration obtained with ~~$k = \infty$~~ ~~solution~~LD-SDA and N_∞.

Figure 11: ~~Solution design comparison~~ Visualization of the catalytic distillation ~~column problem between~~ columns obtained using ~~the~~ LD-SDA with N_2 and N_∞. The feed trays for Ethanol and Butane are shaded in blue, with existing trays shown in white, while bypass trays are shaded in gray. The boil-up position is marked in red, and the reflux position is depicted in blue.

preventing a complete variable initialization as the algorithm progresses. These missing values for the variables might make converging these complex NLP problems challenging, ~~explaining~~ which explains why the D-SDA and the LD-SDA sometimes yield different solutions. Moreover, the solver KNITRO reported that the initial point was infeasible for the more complex NLP problem involving rate-based transfer equations. Using that same initialization, yet using the logic-based D-SDA, the model could not only be started, but it converged to the same optimal solution reported in [35].

5.4 Optimal Design for Chemical Batch Processing

Consider an instance of the optimal design for chemical batch processing from Kocis and Grossmann [64] formulated as a GDP. This is a convexified GDP that aims to find ~~the~~ an optimal design for multiproduct batch plants that minimizes the sum of exponential costs. In our example, the process has ~~3~~ three processing stages where fixed amounts of q_i of ~~2~~ two products must be produced. The goal of the problem is to determine the number of parallel units n_j, the volume v_j of each stage j, the batch sizes b_i, and the cycle time tl_i of each product i. The given parameters of the problem are the time horizon h, cost coefficients α_j, β_j for each stage j, size factors s_{ij}, and processing time t_{ij} for product i in stage j. The optimization model employs Boolean variables Y_{kj} to indicate the presence of a stage, potentially representing three unit types: mixers, reactors, and centrifuges. The formulation of the model can be found in ~~Appendix A.3~~Supplementary Material A.3, and the external variable reformulation of the Boolean variables is described in ~~Appendix~~ Supplementary Material B.4.

The problem was initialized by setting the maximum number of units, i.e., ~~$[3,3,3]$~~$(3,3,3)$, for the number of mixers, reactors, and centrifuges, respectively. The algorithm terminates on a solution with objective \$167,427 with external variables ~~$[2,2,1]$~~ $(2,2,1)$ for both $k = 2$ or $k = \infty$ neighborhood alternatives in LD-SDA. The trajectories taken by both searches of the LD-SDA for the small batch problem are shown in Figure 12. This solution corresponds to ~~the~~ a global optimal solution of the problem, ~~hinting~~ suggesting that convergence to global optimal solutions in convex GDP might be achieved even with $k = 2$ in the ~~Neighbor~~ Neighborhood Search step. For this small problem, the solution

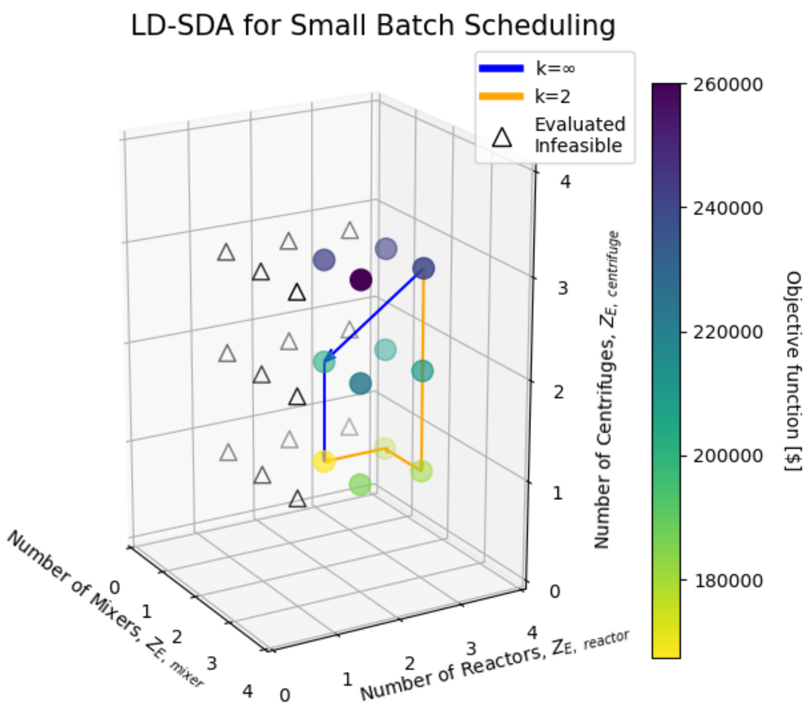


Figure 12: ~~LD-SDA~~ Visualization of the paths ~~to optimal solution in~~ traversed by LD-SDA and the solutions found using both neighborhoods for the small batch scheduling case study. The lattice includes points proven to be infeasible with a solver (black triangle). Both solutions were initialized with a configuration containing all mixers, reactors, and centrifuges. LD-SDA with both $k = 2$ and $k = \infty$ converged to the same solution $(2, 2, 1)$, corresponding to a global optimal solution. This example highlights convergence to the same solution in a convex GDP problem using different neighborhoods.

times were negligible (~~< 2~~ less than two seconds). Still, this example is included to observe convergence to the same solution in a convex GDP problem using different neighborhoods.

6 Conclusions and Final Remarks

This work ~~presented the Logic-based~~ has presented the Logic-Based Discrete-Steepest Descent Algorithm ~~(LD-SDA)~~ as an optimization method for GDP problems with ordered Boolean variables, which often appear in process superstructure and single-unit design problems. The unique characteristics of the LD-SDA are highlighted, and its similarities with other existing logic-based methods are discussed. To verify the performance of the LD-SDA, we solved various GDP problems with applications in process systems engineering, such as reactor series volume minimization, binary distillation column design, rate-based catalytic distillation column design, and chemical batch process design. The LD-SDA has demonstrated an efficient convergence toward high-quality solutions that outperformed state-of-the-art MINLP solvers and GDP solution techniques ~~.~~ for the problems studied. The results show that LD-SDA is a valuable tool for solving GDP ~~models with ordered Boolean variables.~~ problems with the special ordered structure considered in this work. Nonetheless, the scalability of the LD-SDA still needs to be evaluated for larger superstructure problems, e.g., those resulting in more than 7 external variables. The limitations of the LD-SDA include the lack of guarantee for a globally optimal solution due to its local search nature. Additionally, the exponential growth of neighbors with increasing reformulated variables can make neighborhood evaluation prohibitively expensive for large-scale problems.

Future research directions include utilizing the LD-SDA to solve larger and more challenging ordered GDPs. Similarly, we propose exploring theoretical convergence guarantees of the LD-SDA method, with a special focus on convex GDP problems and their relation to integrally convex problems in discrete analysis. Moreover, ~~part of the future work~~ future work also involves the integration of the LD-SDA into the GDPOpt solver in Pyomo.GDP, making ~~it available to more practitioners~~this algorithm available to a wider audience. Finally, we will study the parallelization of NLP solutions in the neighborhood search. The ~~neighbor search~~ Neighborhood Search can be faster by dividing the computation involved in solving NLP problems into multiple tasks that can be executed simultaneously, eventually improving ~~LD-SDA performance~~ the performance of the LD-SDA.

~~Acknowledgements~~Acknowledgments

D.B.N. was supported by the NASA Academic Mission Services, Contract No. NNA16BD14C. D.B.N. and A.L. acknowledge the support of the startup grant of the Davidson School of Chemical Engineering at Purdue University.

References

[1] Ignacio Grossmann. "Enterprise-wide optimization: A new frontier in process systems engineering". *AIChE Journal* 51.7 (2005), pp. 1846–1857.

[2] Oswaldo Andrés-Martínez and Luis A Ricardez-Sandoval. "Integration of planning, scheduling, and control: A review and new perspectives". *The Canadian Journal of Chemical Engineering* 100.9 (2022), pp. 2057–2070.

[3] Luca Mencarelli, Qi Chen, Alexandre Pagot, and Ignacio E Grossmann. "A review on superstructure optimization approaches in process system engineering". *Computers & Chemical Engineering* 136 (2020), p. 106808.

[4] Yingjie Ma, Zekun Yang, Aline El-Khoruy, Nan Zhang, Jie Li, Bingjian Zhang, and Li Sun. "Simultaneous Synthesis and Design of Reaction–Separation–Recycle Processes Using Rigorous Models". *Industrial & Engineering Chemistry Research* 60.19 (2021), pp. 7275–7290.

[5] Adrian Caspari, Calvin Tsay, Adel Mhamdi, Michael Baldea, and Alexander Mitsos. "The integration of scheduling and control: Top-down vs. bottom-up". *Journal of Process Control* 91 (2020), pp. 50–62. ISSN: 0959-1524.

[6] Leo Liberti. "Undecidability and hardness in mixed-integer nonlinear programming". *RAIRO-Operations Research* 53.1 (2019), pp. 81–109.

[7] Ignacio E Grossmann. "Review of nonlinear mixed-integer and disjunctive programming techniques". *Optimization and engineering* 3 (2002), pp. 227–252.

[8] Jon Lee and Sven Leyffer. *Mixed integer nonlinear programming*. Vol. 154. Springer Science & Business Media, 2011.

[9] Francisco Trespalacios and Ignacio E Grossmann. "Review of mixed-integer nonlinear and generalized disjunctive programming methods". *Chemie Ingenieur Technik* 86.7 (2014), pp. 991–1012.

[10] Hector D Perez and Ignacio E Grossmann. "Modeling hierarchical systems via nested generalized disjunctive programming". *Computer Aided Chemical Engineering*. Vol. 52. Elsevier, 2023, pp. 1373–1378.

[11] Laurence A Wolsey and George L Nemhauser. *Integer and combinatorial optimization*. Vol. 55. John Wiley & Sons, 1999.

[12] Ramesh Raman and Ignacio E Grossmann. "Modelling and computational techniques for logic based integer programming". *Computers & Chemical Engineering* 18.7 (1994), pp. 563–578.

[13] Ignacio E Grossmann and Sangbum Lee. "Generalized convex disjunctive programming: Nonlinear convex hull relaxation". *Computational optimization and applications* 26.1 (2003), pp. 83–100.

[14] Egon Balas. *Disjunctive programming*. Springer, 2018.

[15] Metin Türkay and Ignacio E Grossmann. "Logic-based MINLP algorithms for the optimal synthesis of process networks". *Computers & Chemical Engineering* 20.8 (1996), pp. 959–978.

[16] Sangbum Lee and Ignacio E Grossmann. "New algorithms for nonlinear generalized disjunctive programming". *Computers & Chemical Engineering* 24.9-10 (2000), pp. 2125–2141.

[17] Hector Alejandro Pedrozo, SB Rodriguez Reartes, Aldo R Vecchietti, Marıéa Soledad Dıéaz, and Ignacio E Grossmann. "Optimal design of ethylene and propylene coproduction plants with generalized disjunctive programming and state equipment network models". *Computers & Chemical Engineering* 149 (2021), p. 107295.

[18] Xiang Zhang, Zhen Song, and Teng Zhou. "Rigorous design of reaction-separation processes using disjunctive programming models". *Computers & Chemical Engineering* 111 (2018), pp. 16–26.

[19] Hend Ben Ali, Rubén Ruiz-Femenia, and Ammar Ben Brahim. "Design of once-through multistage flash process under the Generalized Disjunctive Programming framework". *2017 International Conference on Green Energy Conversion Systems (GECS)*. IEEE. 2017, pp. 1–7.

[20] Qi Chen and Ignacio E Grossmann. "Effective generalized disjunctive programming models for modular process synthesis". *Industrial & Engineering Chemistry Research* 58.15 (2019), pp. 5873–5886.

[21] Danlei Chen, Xue Ma, Yiqing Luo, Yingjie Ma, and Xigang Yuan. “Synthesis of refrigeration system based on generalized disjunctive programming model”. *Chinese journal of chemical engineering* 26.8 (2018), pp. 1613–1620.

[22] Juan I Manassaldi, Miguel C Mussati, Nicolás J Scenna, and Sergio F Mussati. “Optimization of triple-pressure combined-cycle power plants by generalized disjunctive programming and extrinsic functions”. *Computers & Chemical Engineering* 146 (2021), p. 107190.

[23] Jingyue Cui, Suela Jonuzaj, and Claire S Adjiman. “A Comprehensive Approach for the Design of Solvent-based Adhesive Products using Generalized Disjunctive Programming”. *Computer Aided Chemical Engineering*. Vol. 44. Elsevier, 2018, pp. 427–432.

[24] Suela Jonuzaj, Aparana Gupta, and Claire S Adjiman. “The design of optimal mixtures from atom groups using Generalized Disjunctive Programming”. *Computers & Chemical Engineering* 116 (2018), pp. 401–421.

[25] Suela Jonuzaj and Claire S Adjiman. “Designing optimal mixtures using generalized disjunctive programming: Hull relaxations”. *Chemical Engineering Science* 159 (2017), pp. 106–130.

[26] Ouyang Wu, Giancarlo Dalle Ave, Iiro Harjunkoski, and Lars Imsland. “A rolling horizon approach for scheduling of multiproduct batch production and maintenance using generalized disjunctive programming models”. *Computers & Chemical Engineering* (2021), p. 107268.

[27] Daniel Ovalle, Joshua L Pulsipher, Camilo Gomez, Jorge M Gomez, Carl D Laird, Markus G Drouven, and Ignacio E Grossmann. “Study of Different Formulations for the Multiperiod Blending Problem Applied to Lithium Recovery from Produced Water”. *Computer Aided Chemical Engineering*. Vol. 52. Elsevier, 2023, pp. 1861–1866.

[28] Ming Li. “Multi-periodic Refinery Scheduling Based on Generalized Disjunctive Programming”. *Journal of Physics: Conference Series*. Vol. 1575. 1. IOP Publishing. 2020, p. 012195.

[29] Lijie Su, Lixin Tang, David E Bernal, Ignacio E Grossmann, and Bowen Wang. “Integrated scheduling of on-line blending and distribution of oil products in refinery operation”. *Computer Aided Chemical Engineering*. Vol. 44. Elsevier, 2018, pp. 1213–1218.

[30] Maryam Mohammadi, Edgar Martıén-Hernández, Mariano Martıén, and Iiro Harjunkoski. “Modeling and Analysis of Organic Waste Management Systems in Centralized and Decentralized Supply Chains Using Generalized Disjunctive Programming”. *Industrial & Engineering Chemistry Research* 60.4 (2021), pp. 1719–1745.

[31] Cristiana Lara and Ignacio Grossmann. “Global optimization algorithm for multi-period design and planning of centralized and distributed manufacturing networks”. *Computer Aided Chemical Engineering*. Vol. 43. Elsevier, 2018, pp. 1261–1262.

[32] Maria Analia Rodriguez, Jorge Marcelo Montagna, Aldo Vecchietti, and Gabriela Corsano. “Generalized disjunctive programming model for the multi-period production planning optimization: An application in a polyurethane foam manufacturing plant”. *Computers & Chemical Engineering* 103 (2017), pp. 69–80.

[33] David A Liñán, David E Bernal, Luis A Ricardez-Sandoval, and Jorge M Gómez. “Optimal design of superstructures for placing units and streams with multiple and ordered available locations. Part I: A new mathematical framework”. *Computers & Chemical Engineering* 137 (2020), p. 106794.

[34] Kazuo Murota. “Discrete convex analysis”. *Mathematical Programming* 83.1 (1998), pp. 313–371.

[35] David A Liñán, David E Bernal, Luis A Ricardez-Sandoval, and Jorge M Gómez. “Optimal design of superstructures for placing units and streams with multiple and ordered available locations. Part II: Rigorous design of catalytic distillation columns”. *Computers & Chemical Engineering* 139 (2020), p. 106845.

[36] David A Liñán, David E Bernal, Jorge M Gómez, and Luis A Ricardez-Sandoval. “Optimal synthesis and design of catalytic distillation columns: A rate-based modeling approach”. *Chemical Engineering Science* 231 (2021), p. 116294.

[37] David E Bernal, Daniel Ovalle, David A Liñán, Luis A Ricardez-Sandoval, Jorge M Gómez, and Ignacio E Grossmann. “Process Superstructure Optimization through Discrete Steepest Descent Optimization: a GDP Analysis and Applications in Process Intensification”. *Computer Aided Chemical Engineering*. Vol. 49. Elsevier, 2022, pp. 1279–1284.

[38] David A Liñán and Luis A Ricardez-Sandoval. “Discrete-Time Network Scheduling and Dynamic Optimization of Batch Processes with Variable Processing Times through Discrete-Steepest Descent Optimization”. *Industrial & Engineering Chemistry Research* (2024).

[39] Paola Favati. “Convexity in nonlinear interger programming”. *Ricerca operativa* 53 (1990), pp. 3–44.

[40] Michael L Bynum, Gabriel A Hackebeil, William E Hart, Carl D Laird, Bethany L Nicholson, John D Siirola, Jean-Paul Watson, David L Woodruff, et al. *Pyomo-optimization modeling in python*. Vol. 67. Springer, 2021.

[41] Qi Chen, Emma S Johnson, David E Bernal, Romeo Valentin, Sunjeev Kale, Johnny Bates, John D Siirola, and Ignacio E Grossmann. "Pyomo. GDP: an ecosystem for logic based modeling and optimization development". *Optimization and Engineering* (2021), pp. 1–36.

[42] Ignacio E Grossmann and Francisco Trespalacios. "Systematic modeling of discrete-continuous optimization models through generalized disjunctive programming". *AIChE Journal* 59.9 (2013), pp. 3276–3295.

[43] David E Bernal Neira and Ignacio E Grossmann. "Convex mixed-integer nonlinear programs derived from generalized disjunctive programming using cones". *Computational Optimization and Applications* (2024), pp. 1–62.

[44] Kevin C Furman, Nicolas W Sawaya, and Ignacio E Grossmann. "A computationally useful algebraic representation of nonlinear disjunctive convex sets using the perspective function". *Computational Optimization and Applications* 76.2 (2020), pp. 589–614.

[45] Marco A Duran and Ignacio E Grossmann. "An outer-approximation algorithm for a class of mixed-integer nonlinear programs". *Mathematical programming* 36 (1986), pp. 307–339.

[46] Angelos Tsoukalas and Alexander Mitsos. "Multivariate mccormick relaxations". *Journal of Global Optimization* 59.2-3 (2014), pp. 633–662.

[47] Kazuo Murota and Akihisa Tamura. "Recent progress on integrally convex functions". *Japan Journal of Industrial and Applied Mathematics* (2023), pp. 1–55.

[48] David A Liñán and Luis A Ricardez-Sandoval. "Optimal design and dynamic transitions of multitask catalytic distillation columns: A Discrete-Steepest Descend Framework". *Chemical Engineering and Processing-Process Intensification* 180 (2022), p. 108655.

[49] András Frank and Kazuo Murota. "Decreasing minimization on M-convex sets: algorithms and applications". *Mathematical Programming* 195.1-2 (2022), pp. 1027–1068.

[50] Krzysztof Ciesielski. *Set theory for the working mathematician*. 39. Cambridge University Press, 1997.

[51] Scott P Stevens and Susan W Palocsay. "Teaching use of binary variables in integer linear programs: Formulating logical conditions". *INFORMS Transactions on Education* 18.1 (2017), pp. 28–36.

[52] Balasubramanian Ram and Sanjiv Sarin. "An algorithm for the 0-1 equality knapsack problem". *Journal of the Operational Research Society* 39.11 (1988), pp. 1045–1049.

[53] Nikolaos V Sahinidis. "BARON: A general purpose global optimization software package". *Journal of global optimization* 8 (1996), pp. 201–205.

[54] Tobias Achterberg. "SCIP: solving constraint integer programs". *Mathematical Programming Computation* 1 (2009), pp. 1–41.

[55] Ruth Misener and Christodoulos A Floudas. "ANTIGONE: algorithms for continuous/integer global optimization of nonlinear equations". *Journal of Global Optimization* 59.2-3 (2014), pp. 503–526.

[56] David E Bernal, Stefan Vigerske, Francisco Trespalacios, and Ignacio E Grossmann. "Improving the performance of DICOPT in convex MINLP problems using a feasibility pump". *Optimization Methods and Software* 35.1 (2020), pp. 171–190.

[57] Michael R Bussieck and Arne Drud. "SBB: A new solver for mixed integer nonlinear programming". *Talk, OR* (2001).

[58] Richard H Byrd, Jorge Nocedal, and Richard A Waltz. "Knitro: An integrated package for nonlinear optimization". *Large-scale nonlinear optimization* (2006), pp. 35–59.

[59] Arne Stolbjerg Drud. "CONOPT—a large-scale GRG code". *ORSA Journal on computing* 6.2 (1994), pp. 207–216.

[60] Andreas Wächter and Lorenz T Biegler. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". *Mathematical programming* 106 (2006), pp. 25–57.

[61] Jaffer H Ghouse, Qi Chen, Miguel A Zamarripa, Andrew Lee, Anthony P Burgard, Ignacio E Grossmann, and David C Miller. "A comparative study between GDP and NLP formulations for conceptual design of distillation columns". *Computer Aided Chemical Engineering*. Vol. 44. Elsevier, 2018, pp. 865–870.

[62] Jennifer R Jackson and Ignacio E Grossmann. "A disjunctive programming approach for the optimal design of reactive distillation columns". *Computers & Chemical Engineering* 25.11-12 (2001), pp. 1661–1673.

[63] David E Bernal, Carolina Carrillo-Diaz, Jorge M Gómez, and Luis A Ricardez-Sandoval. "Simultaneous design and control of catalytic distillation columns using comprehensive rigorous dynamic models". *Industrial & Engineering Chemistry Research* 57.7 (2018), pp. 2587–2608.

[64] Gary R Kocis and Ignacio E Grossmann. "Global optimization of nonconvex mixed-integer nonlinear programming (MINLP) problems in process synthesis". *Industrial & engineering chemistry research* 27.8 (1988), pp. 1407–1421.

The appendices show how the

Supplementary Material

The supplementary material provides detailed formulations of GDP models, such as including reactors and chemical batch processing , are formulated. The systems. It outlines the objective functions and their presents both the algebraic and logical constraints are described for the model. Also, the reformulations of the Boolean Variables into External variables for the models are given in this section. There are some models which are described as figures for each model. Additionally, this section explains how Boolean variables are reformulated into external variables. Several models are also illustrated through figures for clarity.

A Generalized Disjunctive Programming formulations

This appendix includes the formulations of the examples of the problems solved in this manuscript as Generalized Disjunctive Programs(GDP).

A.1 Series of Reactors

Set of components (index *i*)

$$I = \{A, B\} \tag{8}$$

Set of units in the superstructure (index *n, j*)

$$N = \{1, ..., NT\} \tag{9}$$

Existence of an unreacted feed in unit *n*

$$\begin{gathered} YF_n \in \{\textit{True, False}\} \;\; \forall\, n \in N \\ \text{If } YF_n = \textit{True} \implies \text{There is unreacted feed in reactor } n \end{gathered} \tag{10}$$

Existence of a recycle flow in unit *n*

$$\begin{gathered} YR_n \in \{\textit{True, False}\} \;\; \forall\, n \in N \\ \text{If } YR_n = \textit{True} \implies \text{There is recycle in reactor } n \end{gathered} \tag{11}$$

Unit operation in *n*: If at the current unit *n* **every** unit after it (from 1 one to *n*) **is not** an unreacted feed **or** if the current unit *n* has the unreacted feed, then the unit is a CSTR (the opposite is also true)

$$\begin{gathered} YP_n \iff \left(\bigwedge_{j \in \{1,2,..n\}} \neg YF_j \right) \vee YF_n \;\; \forall\, n \in N \\ \text{If } YP_n = \textit{True} \implies \text{Unit } n \text{ is a CSTR} \\ \text{If } YP_n = \textit{False} \implies \text{Unit } n \text{ is a bypass} \end{gathered} \tag{12}$$

The unit must be a CSTR to include a recycle at *n*

$$YR_n \implies YP_n \;\; \forall\, n \in N \tag{13}$$

There is only one unreacted feed

$$\bigvee_{n \in N} YF_n \tag{14}$$

There is only one recycling stream.

$$\bigvee_{n \in N} YR_n \tag{15}$$

Unreacted feed unit: Partial mole balance

$$0 = F0_i + FR_{i,NT} - F_{i,NT} + r_{i,NT} V_{NT} \;\; \forall\, i \in I \tag{16}$$

Unreacted feed unit: Continuity

$$0 = Q_{F0} + Q_{FR,NT} - Q_{NT} \tag{17}$$

Reactor Sequence: Partial mole balance

$$0 = F_{i,n+1} + FR_{i,n} - F_{i,n} + r_{i,n}V_n \;\; \forall\, n \in N \setminus \{NT\}, \forall\, i \in I \tag{18}$$

Reactor Sequence: Continuity

$$0 = Q_{n+1} + Q_{FR,n} - Q_n \;\; \forall\, n \in N \setminus \{NT\} \tag{19}$$

If unit *n* is a CSTR or a bypass

$$\begin{bmatrix} YP_n \\ r_{A,n}Q_n^2 = -kF_{A,n}F_{B,n} \\ r_{B,n} = -r_{A,n} \\ c_n = V_n \end{bmatrix} \underline{\bigvee} \begin{bmatrix} \neg YP_n \\ FR_{i,n} = 0 \;\; \forall\, i \in I \\ r_{i,n} = 0 \;\; \forall\, i \in I \\ Q_{FR,n} = 0 \\ c_n = 0 \end{bmatrix} \forall\, n \in N \tag{20}$$

If there is recycle in before reactor *n*

$$\begin{bmatrix} YR_n \\ FR_{i,n} = R_i \;\; \forall\, i \in I \\ Q_{FR,n} = Q_R \end{bmatrix} \underline{\bigvee} \begin{bmatrix} \neg YR_n \\ FR_{i,n} = 0 \;\; \forall\, i \in I \\ Q_{FR,n} = 0 \end{bmatrix} \forall\, n \in N \tag{21}$$

Splitting point: Partial mole balance

$$0 = F_{i,1} - P_i - R_i \;\; \forall\, i \in I \tag{22}$$

Splitting point: Continuity

$$0 = Q_1 - Q_P - Q_R \tag{23}$$

Splitting point: Additional constraint

$$0 = P_i Q_1 - F_{i,1} Q_P \;\; \forall\, i \in I \tag{24}$$

Product specification constraint

$$0.95 Q_P = P_B \tag{25}$$

Volume constraint

$$V_n = V_{n-1} \;\; \forall\, n \in N \setminus \{1\} \tag{26}$$

Objective Function: Total reactor network volume

$$f_{OBJ} = \min \sum_{n \in N} c_n \tag{27}$$

A.2 Distillation Column Design

Set of trays (index *t*)

$$T = \{2, 3, \ldots, 16\} \tag{28}$$

Set of composition (index *c*)

$$C = \{\text{Benzene, Toluene}\} \tag{29}$$

Existence of tray *t*

$$\begin{gathered} Y_t \in \{\textit{True, False}\} \;\; \forall\, t \in T \\ \text{If } Y_i = \textit{True} \implies \text{There exist a tray in stage } t \end{gathered} \tag{30}$$

Existence of boil-up flow in tray *t*

$$\begin{gathered} YB_t \in \{\textit{True, False}\} \;\; \forall\, t \in T \\ \text{If } YB_t = \textit{True} \implies \text{There is a boil-up flow in tray } t \end{gathered} \tag{31}$$

Existence of reflux flow in tray *t*

$$YR_t \in \{\textit{True, False}\} \;\; \forall\, t \in T$$
$$\text{If } YR_i = \textit{True} \implies \text{There is a reflux flow in tray } t \tag{32}$$

There is only one boil-up flow in the distillation column.

$$\underline{\bigvee}_{t \in T} YB_t \tag{33}$$

There is only one reflux flow in the distillation column.

$$\underline{\bigvee}_{t \in T} YR_t \tag{34}$$

Tray t is an equilibrium stage or a bypass, where $\mathbf{g}_1(\mathbf{x})$ contains equilibrium mass and energy balances, while $\mathbf{g}_2(\mathbf{x})$ contains a bypass material balance

$$\begin{bmatrix} Y_t \\ \mathbf{g}_1(\mathbf{x}) = \mathbf{0} \\ y_{t,\text{active}} = 1 \end{bmatrix} \underline{\bigvee} \begin{bmatrix} \neg Y_t \\ \mathbf{g}_2(\mathbf{x}) = \mathbf{0} \\ y_{t,\text{active}} = 0 \end{bmatrix}, \quad \forall\, t \in T \tag{35}$$

If the reflux flow is on or above tray t and the boil-up flow is on or below tray t, then tray t is an equilibrium stage (the opposite is also $True$).

$$\left(\bigvee_{\forall\, \tau \in \{t, t+1, \ldots, 16\}} YR_\tau \right) \wedge \left(\bigwedge_{\forall\, \tau \in \{t, t+1, \ldots, 16\}} \neg YB_\tau \vee YB_t \right) \iff Y_t, \quad \forall\, t \in T \tag{36}$$

The reflux flow stage is not below the feed tray.

$$\bigwedge_{\forall\, t \in \{2, 3, \ldots, 8\}} \neg YR_t \tag{37}$$

The boil-up flow is not above the feed tray.

$$\bigwedge_{\forall\, t \in \{10, 11, \ldots, 17\}} \neg YB_t \tag{38}$$

The column has at least eight active trays.

$$\sum_{t \in T} y_{t,active} \geq 8 \tag{39}$$

Tray 1 (reboiler), tray 9 (feed), and tray 17 (condenser) are equilibrium stages.

$$Y_1 = True,\; Y_9 = True,\; Y_{17} = True, \tag{40}$$

Benzene concentration constraint (distillate product)

$$X_{D,\text{Benzene}} \geq 0.95 \tag{41}$$

Toluene concentration constraint (bottom product)

$$X_{B,\text{Toluene}} \geq 0.95 \tag{42}$$

Bounds imposed over the reflux ratio RF

$$0.5 \leq RF \leq 4 \tag{43}$$

Bounds imposed over the Reboil ratio RB

$$1.3 \leq RB \leq 4 \tag{44}$$

Objective Function: Sum of the capital cost (number of active trays) and operating cost (reboiler and condenser duty), where Q_R is the reboiler duty, Q_C is the condenser duty, and the sum term over binary variables $y_{t,\text{active}}$ represents the total number of active trays.

$$f_{OBJ} = \min_{RF,RB,feedtray,N_{trays}} 10^3 \left((Q_R + Q_C) + \sum_{t \in T}^{N_{trays}} y_{t,\text{active}} \right) \tag{45}$$

A.3 Small Batch Problem

Set of components (index *i*)

$$I = \{A, B\} \tag{46}$$

Set of stages (index *j*)

$$J = \{\text{mixer, reactor, centrifuge}\} \tag{47}$$

Set of potential number of parallel units for each stage (index *k*)

$$K = \{1, 2, 3\} \tag{48}$$

Existence of the parallel units for each stage *j*

$$\begin{gathered} Y_{kj} \in \{\textit{True, False}\} \ \forall\, k \in K,\ j \in J \\ \text{If } Y_{kj} = \textit{True} \implies \text{There are } k \text{ parallel units in stage } j \end{gathered} \tag{49}$$

Only one of the parallel unit existence is $True$

$$\underline{\bigvee_{j \in J}} Y_{kj} \quad \forall\, k \in K \tag{50}$$

Volume requirement in stage *j*

$$v_j \geq \ln(s_{ij}) + b_i \quad \forall\, i \in I,\ j \in J \tag{51}$$

Cycle time for each product *i*

$$n_j + t_{Li} \geq \ln(t_{ij}) \quad \forall\, i \in I,\ j \in J \tag{52}$$

Constraint for production time(horizon constraint)

$$\sum_{i \in I} Q_i \exp\left(t_{Li} - b_i\right) \leq H \tag{53}$$

Relating number of units to $0-1$ variables

$$n_j = \sum_{k \in K} \gamma_{kj} \quad \forall\, j \in J \tag{54}$$

If only *k* parallel units exist in stage *j*

$$\begin{bmatrix} Y_{kj} \\ \gamma_{kj} = \ln(k) \end{bmatrix} \underline{\bigvee} \begin{bmatrix} \neg Y_{kj} \\ \gamma_{kj} = 0 \end{bmatrix} \quad \forall\, k \in K,\ j \in J \tag{55}$$

Objective Function: the investment cost for setting the small batch system [$]

$$f_{OBJ} = \min \sum_{j \in J} \alpha_j (\exp(n_j + \beta_j v_j)) \tag{56}$$

B External variable reformulation for example problems

This appendix presents the external variable reformulation of the Boolean variables in the examples considered in this manuscript.

B.1 Series of Reactors Problem

$$YF_n = \begin{cases} True, & \mathbf{z}_{E,1} = n \\ False, & \text{otherwise} \end{cases} \quad \forall\, n \in N \tag{B.1.57}$$

$$YR_n = \begin{cases} True, & \mathbf{z}_{E,2} = n \\ False, & \text{otherwise} \end{cases} \quad \forall\, n \in N \tag{B.1.58}$$

$$X_1 = \left\{ \mathbf{z_E} \in \mathbb{Z}^2 : \begin{matrix} 1 \le \mathbf{z_{E,1}} \le NT \\ 1 \le \mathbf{z_{E,2}} \le NT \end{matrix} \right\} \tag{B.1.59}$$

$$X_2 = \left\{ \mathbf{z_E} \in \mathbb{Z}^2 : \mathbf{z_{E,2}} - \mathbf{z_{E,1}} \le 0 \right\} \tag{B.1.60}$$

$$X = X_1 \cap X_2 = \left\{ x_E \in \mathbb{Z}^2 : \begin{matrix} 1 - \mathbf{z_{E,1}} \le 0 \\ \mathbf{z_{E,1}} - NT \le 0 \\ 1 - \mathbf{z_{E,2}} \le 0 \\ \mathbf{z_{E,2}} - \mathbf{z_{E,1}} \le 0 \end{matrix} \right\} \tag{B.1.61}$$

B.2 External Variable Reformulation for Distillation Column Problem

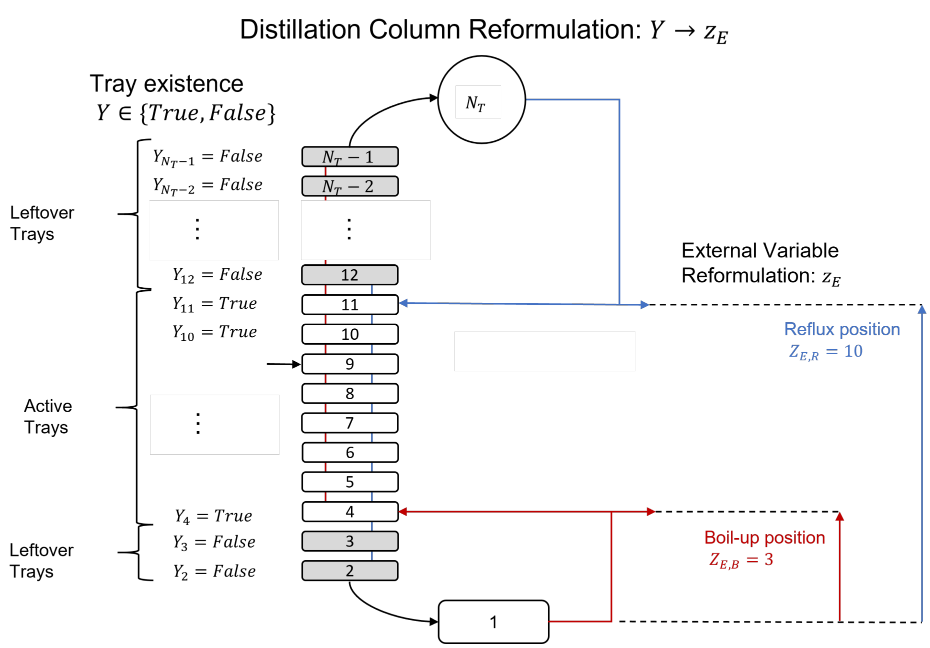


Figure 13: Example of distillation column external variable reformulation

$$YR_t = \begin{cases} True, & \mathbf{z}_{\mathbf{E},\text{reflux}} = t - 1 \\ False, & \text{otherwise} \end{cases} \quad \forall\, t \in T \tag{B.2.1}$$

$$YB_t = \begin{cases} True, & \mathbf{z}_{\mathbf{E},\text{boil-up}} = t - 1 \\ False, & \text{otherwise} \end{cases} \quad \forall\, t \in T \tag{B.2.2}$$

$$X_1 = \left\{ \mathbf{z_E} \in \mathbb{Z}^2 : \begin{matrix} 1 \le \mathbf{z}_{\mathbf{E},\text{reflux}} \le 15 \\ 1 \le \mathbf{z}_{\mathbf{E},\text{boil-up}} \le 15 \end{matrix} \right\} \tag{B.2.3}$$

B.3 Catalytic Distillation Column Problem

The external variable reformulation is equivalent to the one presented in [35] with Boolean variables instead of binary variables. We highlight in Figure 14 how these external variables are interpretable as relative positions of the ethanol feed, the butene feed, the catalytic stages and the boil-up.

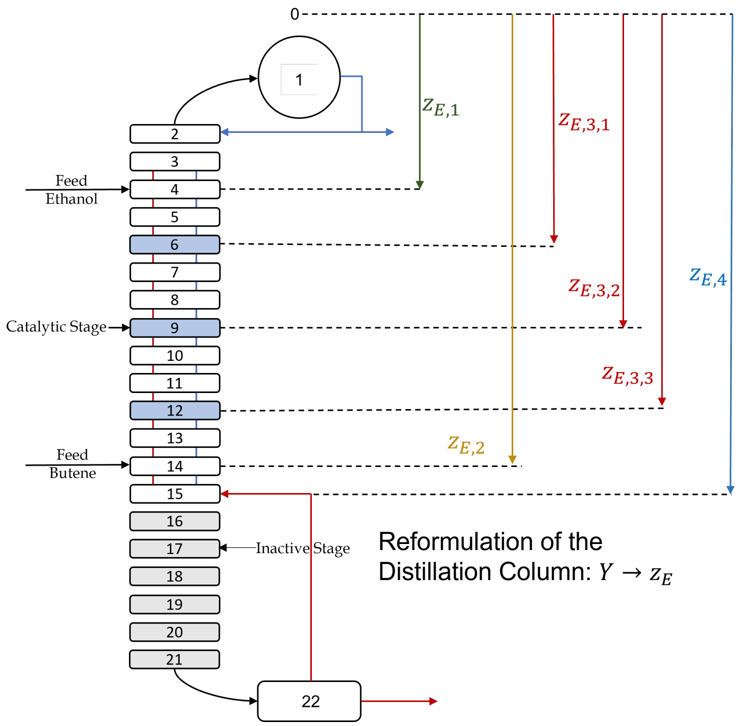


Figure 14: Example of catalytic distillation column external variable reformulation

B.4 Small Batch Problem

$$Y_{k,\text{mixer}} = \begin{cases} True, & \mathbf{z_E}_{,\text{mixer}} = k \\ False, & \text{otherwise} \end{cases} \quad \forall\, k \in K \tag{B.4.1}$$

$$Y_{k,\text{reactor}} = \begin{cases} True, & \mathbf{z_E}_{,\text{reactor}} = k \\ False, & \text{otherwise} \end{cases} \quad \forall\, k \in K \tag{B.4.2}$$

$$Y_{k,\text{centrifuge}} = \begin{cases} True, & \mathbf{z_E}_{,\text{centrifuge}} = k \\ False, & \text{otherwise} \end{cases} \quad \forall\, k \in K \tag{B.4.3}$$

$$X_1 = \left\{ \mathbf{z_E} \in \mathbb{Z}^3 : \begin{array}{c} 1 \leq \mathbf{z_E}_{,\text{mixer}} \leq K \\ 1 \leq \mathbf{z_E}_{,\text{reactor}} \leq K \\ 1 \leq \mathbf{z_E}_{,\text{centrifuge}} \leq K \end{array} \right\} \tag{B.4.4}$$