Pyosyn: a new framework for conceptual design modeling and optimization

Qi Chen[a], Yunshan Liu[a], Grant Seastream[a], John D. Siirola[b], Ignacio E. Grossmann[a,*]

[a] *Center for Advanced Process Decision-Making, Carnegie Mellon University, Pittsburgh, PA 15213*
[b] *Sandia National Laboratories, Albuquerque, NM 87123*

Abstract

We present Pyosyn, an open-source framework for systematic superstructure-based process synthesis, including a new representation, superstructure generation approaches, modeling, and solution strategies. The new **P**yo**s**yn **G**raph (PSG) representation consists of units, ports, and streams, and includes support for nested units, including new "single-choice" units and modular superstructure construction. We introduce superstructure generation strategies based on both library-assisted and direct-hierarchical means-ends analysis. For the library-assisted approach, we describe generalized port annotations that describe conditions for compatibility between connected unit ports. We extend literature methods to present seven screening rules based on new material port annotations that categorize process chemical species as *primary*, *secondary*, or *residual*. We then describe high-level mathematical modeling of PSG representation elements using Pyomo.Network and Pyomo.GDP, including the automated handling of special cases. We also introduce the use of tailored logic-based decomposition algorithms to address "zero-flow" singularities characteristic of synthesis problems. Finally, we demonstrate the flexible use of Pyosyn tools on a set of diverse case studies.

Keywords: conceptual design, mathematical programming, generalized disjunctive programming, process synthesis, open source

1. Introduction

Recent developments have brought new opportunities as well as new challenges for the chemical process industry, including availability of inexpensive feedstocks from the shale gas revolution in the U.S., rising awareness of environmental impacts, evolving regulatory landscapes, and renewed volatility in

*Corresponding author
Email address: grossmann@cmu.edu (Ignacio E. Grossmann)

market conditions (Grossmann and Harjunkoski, 2019). The industry must decide how to best adapt its business practices to most effectively provide society with the fuels, polymers, pharmaceuticals, and other chemical goods necessary to sustain increasing standards of life across the world. Integral to these analyses are the process flowsheet designs themselves, both for the construction of new process plants and the retrofit of existing facilities. In crafting these new designs, decision-makers must choose between novel modular and/or intensified processing options that promise new operating modes and economic gains, or the time-tested efficiency and reliability of conventional large-scale processing plants.

Advances in conceptual process design aim to help decision-makers in assessing existing and prospective technology alternatives. Several broad reviews in overall flowsheet design may be found in literature (Tsay et al., 2018; Chen and Grossmann, 2017; Barnicki and Siirola, 2004; Westerberg, 2004), with other authors focusing on recent challenges such as sustainability/circular economy (Martín and Adams II, 2019; Avraamidou et al., 2020), process intensification (Sitter et al., 2019; Tula et al., 2019a; Tian et al., 2018a), and modular design (Baldea et al., 2017). Three main conceptual design approaches are described in the literature: evolutionary methods (Stephanopoulos and Westerberg, 1976), hierarchical decomposition (Douglas, 1985; Siirola and Rudd, 1971), and superstructure-based methods (Sargent and Gaminibandara, 1976).

Evolutionary methods involve iterative variations from a starting base case, utilizing various rules for permuting the design (Nishida et al., 1981; Neveux, 2018). Some recent authors also refer to these methods as "superstructure free" strategies (Boonstra et al., 2016). These techniques rely on effective permutation rules to cover desirable regions of design space. In general, convergence guarantees in finite time are not available, with termination criteria commonly defined in terms of a time limit or a stalling of progress between successive iterates.

Hierarchical decomposition involves a sequence of decisions made at progressively refined detail levels. Douglas (1988) defines these detail levels as:

1. Batch versus continuous
2. Input-output structure
3. Recycle structure and reactors
4. Separation systems
5. Heat exchanger network

The intuition is to first make those decisions at a high level that are most consequential, and then to refine the behaviors of flowsheet subsystems to arrive at an optimal flowsheet. Heuristics and rules-of-thumb guide decision-making at each decision level. These can be based on targeting techniques Horn (1964); Glasser et al. (1987) and pinch analysis Linnhoff and Hindmarsh (1983); Hohman (1971),

or other forms of engineering judgement. Through the use of these decision levels, hierarchical decomposition modularizes the decision-making process, creating an intuitive design procedure for decision makers. However, potential interactions between decision levels become more difficult to capture, e.g. between process design and heat integration (Lang et al., 1988; Duran and Grossmann, 1986b). More recent work in this area, e.g. by Tula et al. (2019b), combine library knowledge with thermodynamic insights to address these challenges.

Superstructure optimization aims to provide a systematic search over the entire relevant design space, by postulating all relevant permutations of the process alternatives and their interconnections (captured as a superstructure), then formulating a mathematical programming model and solving it to identify the optimal design. The use of mathematical programming confers one of the main advantages of superstructure-based synthesis—a mathematical optimality guarantee (with global optimization techniques) of the maximum gap with respect to the user-specified objective function, between a feasible flowsheet and the best possible design embedded in the superstructure. This approach relies on the success of three main steps:

1. Generation of a superstructure embedding the relevant flowsheet alternatives
2. Formulation of a tractable mathematical programming model
3. Solution with a mathematical programming code to obtain optimal flowsheet design

The first step is generation of a superstructure representation that embeds the optimal flowsheet. The optimal design must be a subgraph of the superstructure to be successfully identified (Westerberg, 2004; Agrawal, 1996). The choice of representation can also have an impact on the tractability of the later solution step (Yeomans and Grossmann, 1999). Mencarelli et al. (2020) provide a detailed review of superstructure representations and their respective generation techniques. Next, the superstructure must be translated into a mathematical model that captures the relevant decision logic and constraints. Even under ideal thermodynamic assumptions, a flowsheet design involving multiple chemical species will usually yield a Mixed-Integer Nonlinear Programming (MINLP) model with both continuous and discrete decision variables, and with non-convex variable relationships (Trespalacios and Grossmann, 2014). More recently, Generalized Disjunctive Programming (GDP) has also gained popularity as an alternative modeling approach. We discuss these modeling approaches in more detail in Section 5. Finally, given the difficult class of optimization problems that result from process design applications, advanced solution algorithms are often necessary to arrive at optimal or near-optimal candidate solutions and to converge the optimality within a given tolerance. Trespalacios and Grossmann (2014) and Kronqvist et al. (2019) provide recent reviews of MINLP and GDP solution algorithms. We also further elaborate on solution strategies in Section 6.

Despite these academic accomplishments, industrial practice largely remains partial enumeration, in which a set of prospective case studies are run to analyze specific feasible flowsheet configurations. As a result, a longstanding ambition of our community has been the development of effective computational tools to make state-of-the-art techniques accessible to both practitioners and fellow researchers. Software tools to support synthesis range from commercial process simulators to specialized tools for the synthesis of specific subsystems, for example, SYNHEAT (Yee and Grossmann, 1990) for heat exchanger network synthesis. Of particular interest are general purpose process synthesis tool sets. PROSYN (Schembecker et al., 1994) and ICAS (Gani et al., 1997) are the main tools that support a hierarchical decomposition design approach. For superstructure-based synthesis, MIPSYN (Kravanja and Grossmann, 1990), Super-O (Bertran et al., 2017), P-Graph Studio (Friedler et al., 2019), and SYNOPSIS (Tian et al., 2018b) are the actively developed platforms. Of superstructure-based frameworks, Super-O has made notable inroads among the industrial space. However, there remain hurdles to widespread adoption of these tools.

These hurdles include lack of awareness, a high cost of adoption, and the inherent difficulty of optimal process flowsheet design problems. Lack of awareness stems both in terms of the value of conceptual design, and in how to use these tools. The solution to this lies in education, both to students at our academic institutions and among our collaborators. The high cost of adoption also deters some potential users. This begins with availability. Though their frameworks and methods may be described in literature, none of the tools listed above are yet released on an open-source basis. Even when the tools are available, it may be a challenge to integrate pre-existing models and analysis workflows, which may be developed in custom software. Finally, it can be computationally difficult to solve the large mathematical programming problems associated with superstructure synthesis to obtain a feasible and optimal solution. Practitioners are often unaware of the scope of design that can be addressed with state-of-the-art tools.

In seeking solutions to the conceptual design problem, these tools must also balance a fundamental trade-off of process synthesis analyses: between generality, fidelity, and tractability (see Figure 1). We denote this the "central trade-

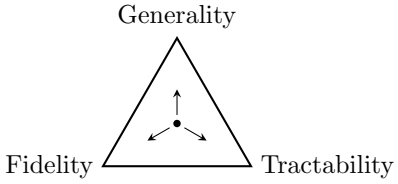


Figure 1: Central trade-off of process synthesis strategies

off" of synthesis. In the central trade-off, generality refers to the scope of design

space that a given synthesis technique is able to enumerate, whether implicitly or explicitly. With great recent interest in process intensification (Stankiewicz, 2018), promising great improvements to cost, equipment size, and process reliability, several authors have proposed design strategies with chemical/physical phenomena as the base-level flowsheet building blocks, rather than equipment or processing tasks (see review by Sitter et al. (2019)). At the same time, the models describing interconnection and selection of these phenomena must be sufficiently detailed to capture the synergistic effects of intensification. That is, they must have reasonably high fidelity. However, a design trade-off favoring generality and fidelity often struggles with tractability, as state-of-the-art solution strategies may not be able to identify good solutions, let alone potentially optimal ones. This trade-off is broadly relevant, as with enterprise-wide optimization (Grossmann, 2014), a process design may take place in the context of a larger petrochemical complex, or even a multi-site processing network; however, attempting to model all process elements at a high level of detail may be challenging to even simulate, a necessary prerequisite of optimization. As a result of the adoption hurdles and the inherent difficulties of synthesis, there remains wide scope for continued advances in process design.

In this paper, we introduce Pyosyn, a new flexible framework for conceptual process design, within the IDAES process systems engineering tool set (Miller et al., 2018). Pyosyn postulates an open-source collection of inter-operable methods and tools, interlinked using the Python general purpose programming language. Where stable versions of Pyosyn software components are available, we highlight their use and function; otherwise, we describe the methods and logic required to link framework elements together. Pyosyn may not be the final solution to all of the challenges listed above; however, we intend for it to serve as an open platform upon which the community may try new ideas, and expose new audiences to the benefits of systematic design. By adopting a modular design, we offer users the ability to select the components of Pyosyn that are most useful to their needs, and to integrate their own custom solution for other capabilities. In the following sections, we describe functionality present in the various components of Pyosyn, and how it may be adapted to suit different design applications. In Section 3, we introduce the **P**yo**s**yn **G**raph (PSG), a new superstructure representation for Pyosyn. In Section 4, we describe generation approaches for PSG. We then discuss translation of the PSG representation into a logical model in Section 5, followed by the suite of supported solution strategies in Section 6. We demonstrate the Pyosyn design approaches and its adaptability with case studies in Section 7. Finally, we conclude in Section 8.

2. Problem statement

In this paper, we define the conceptual design problem as follows: given a set of potential raw materials and desired end products, as well as a set of processing alternatives, identify the process flowsheet, equipment sizes, and operating conditions that optimize an economic, social, and/or environmental objective.

Several adaptations of this problem are possible to accommodate different applications. For example, alternative objectives can be adopted. For multi-objective design, the $\epsilon-$constraint method is popular (Ehrgott and Wiecek, 2005); we refer the reader to a review by Marler and Arora (2004) for more details. In retrofit design, existing equipment can be added as an alternative, and their costs can be adjusted to better reflect the trade-offs. In situations where multiple feed and/or product candidates exist, they can also be added as alternatives.

Conceptual design often takes place within the context of a broader analysis– for example, as part of a feasibility study for a new product launch. Its results may form the basis for a subsequent detailed design study, or may be used to guide the direction of related process technology research. That is to say, conceptual design is rarely a once-through exercise, and the analysis may be repeated to explore the Pareto-frontier between multiple objectives, or the constituent model assumptions may be revisited based on further analysis. A conceptual design framework must therefore be flexible enough to allow a decision-maker to pose the questions relevant to their analysis.

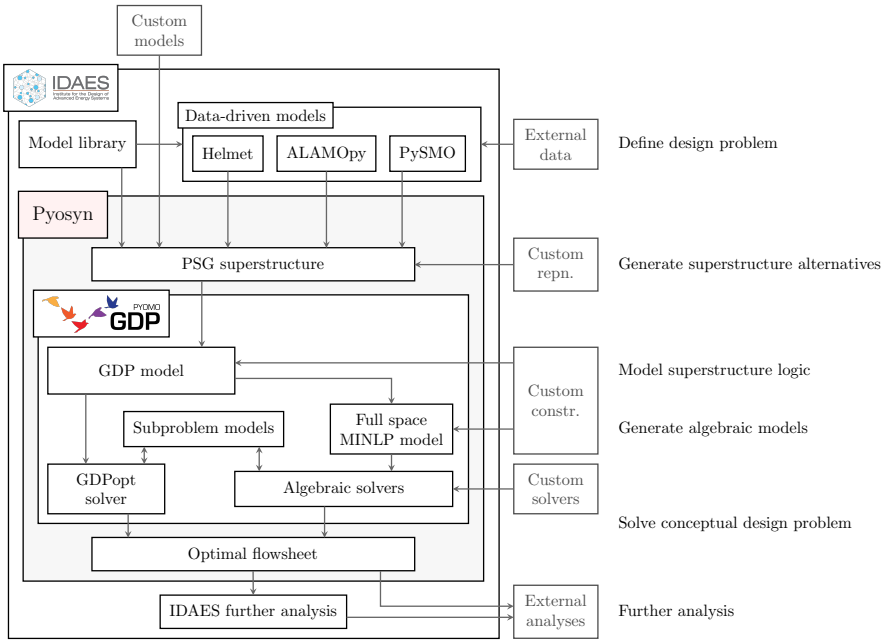


Figure 2: Pyosyn framework conceptual flow diagram, illustrating key components and flexible integration points. Abbreviations: repn = representation; constr = constraints.

Based on these challenges, we adopt the following guiding principles in designing Pyosyn:

1. Intuitive representations

2. Systematic transformations to algebraic models
3. Flexible solution strategies
4. Open architecture

Intuitive representations for both the superstructure topology—through an inherently modular implementation—and the high-level decision logic—through an explicit syntax—facilitate ease-of-use for Pyosyn users. Both help manage complexity as iterative adjustments are made to design assumptions or design goals. Systematic, automated reformulations from the logical model to algebraic forms facilitate the ability to make model adjustments using higher-level intuitive representations, without the need to propagate these changes manually to solver-compatible syntax. Flexible solution strategies without the need for model adjustments in Pyosyn allows for more robust solution of synthesis problems, as one approach may be able to provide a solution when others fail. And finally, the open-source implementation of Pyosyn gives transparency into its functionality, facilitates future research development, instils confidence in the software, and allows for arbitrary customization.

Figure 2 gives a broad overview of the components of Pyosyn, as well as its potential interfaces to other IDAES tools (Miller et al., 2016) and external data/specifications. To define the design problem, alternatives from the IDAES unit model library (Lee et al., 2018) may be used, in addition to data-driven models built using IDAES surrogate model-building tools. Superstructure units based on custom models may also be used with Pyosyn, providing that they define the appropriate flowsheet interfaces using unit ports (see Section 3.1.2).

Next, Pyosyn supports the generation of a superstructure representation via Pyomo.Network and the underlying graph representation using the Python package networkx (Hagberg et al., 2008). In Section 3, we describe the **P**yo**s**yn **G**raph representation, and generation strategies in Section 4. However, a custom representation (e.g. a superstructure for HEN or utility plant synthesis) may also be used with Pyosyn, provided that it can be translated to a logical or algebraic mathematical programming model.

Flexibility is also provided for modeling superstructure logic and translating it to a mathematical form. Here, Pyosyn employs Pyomo.GDP (Chen et al., 2018) to offer high-level logical syntax for describing superstructure logic. Advanced solvers in Pyomo.GDP such as GDPopt (Chen et al., 2018) can directly address these logic-based models, but systematic reformulations are also possible to conventional algebraic forms. Custom model specifications may be introduced into Pyosyn through the addition of custom constraints. Likewise, custom solution routines can also be implemented for a given Pyosyn design problem. Finally, the optimal flowsheet design can be passed to other IDAES or external tools for further analysis.

3. Superstructure representation

The choice of superstucture representation can impact the central trade-off of synthesis. As previously mentioned, a representation can affect the tractability of the resulting algebraic model (Yeomans and Grossmann, 1999). Indeed, the R-graph representation (Farkas et al., 2005) was conceived in large part to improve tractability compared to the State-Equipment Network (SEN) representation (Smith and Pantelides, 1995), or the State-Task Network with One Task, One Equipment (STN-OTOE) (Kondili et al., 1993). More recently, superstructures have been introduced for process intensification with a greater emphasis on generality (Demirel et al., 2017; Lutze et al., 2013; Kuhlmann and Skiborowski, 2016), but sometimes to the detriment of tractability. Specialized representations for subsystems have also long existed, e.g. for distillation sequencing (Agrawal, 1996) and heat exchanger networks (Yee and Grossmann, 1990). We refer the reader to a recent critical review of existing superstructure representations (Mencarelli et al., 2020).

The ideal superstructure representation is defined by three characteristics: generality, ease of use, and tractability. Generality and tractability considerations here are the same as described for the central trade-off of synthesis. The superstructure must embed the optimal configuration, and its complexity impacts tractability. Ease of use is less well-defined in literature. However, its characteristics include visual appeal, the existence of systematic approaches and tools to support its generation, and the ease of modeling—elements that make it more intuitive to the process engineer and facilitate its adoption.

In this work, we introduce the **P**yo**s**yn **G**raph (PSG) representation with these goals in mind. Among the sources of inspiration for the PSG, we highlight the P-Graph (Friedler et al., 1992), STN (Kondili et al., 1993), SEN (Smith and Pantelides, 1995), Unit-Operation-Port State Superstructure (UOPSS) (Kelly, 2004), Unit-Port-Conditioning Stream (UPCS) (Wu et al., 2016), and Processing Step-Interval Network (PSIN) (Bertran et al., 2017) representations. The P-Graph and the UPCS representations both describe clear, graph-based systematic generation strategies, aiding in ease of use. The STN and SEN representations offer visual representations similar to the familiar process flow diagram (PFD). These both describe a directed bipartite graph between tasks (or equipment), with states as the connecting arcs. From the PSIN, we draw inspiration from its nesting of processing phenomena within each processing step-interval "unit". We also draw inspiration from the Pyomo algebraic modeling language (Hart et al., 2017)—upon which we build Pyosyn—and its support for "Block"-centric hierarchical modeling (Friedman et al., 2013).

3.1. PSG Representation

The **P**yo**s**yn **G**raph (PSG) representation consists of three main representation elements: (1) units, representing potential system control volumes; (2) unit ports, corresponding to boundaries through which flows of material and/or energy may take place; and (3) streams, which represent these flows between control volumes.

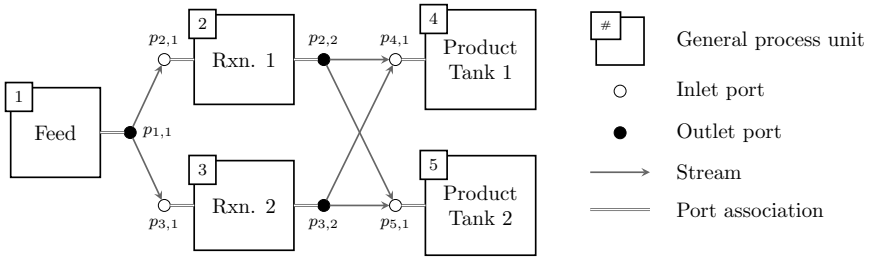


Figure 3: Demonstration of PSG representation elements.

3.1.1. Units

The processing units in the PSG are the central elements of the representation. We denote the set of PSG units $\mathbf{U}$. Traditionally, superstructure units have been understood to represent alternative processing tasks (STN) (Kondili et al., 1993) or equipment (SEN) (Smith and Pantelides, 1995) in the flowsheet. Here, we adopt a more general definition in which units can represent any predetermined operation on a control volume. Many other existing representations adopt this more general definition (Papalexandri and Pistikopoulos, 1994; Friedler et al., 1992; Wu et al., 2016; Bagajewicz and Manousiouthakis, 1992). In many cases, the PSG unit will correspond to a physical piece of equipment, as in the SEN or STN-OTOE (Yeomans and Grossmann, 1999). However, as we illustrate below, a PSG unit can also represent a process subsystem, a processing task, or a control volume within a potential piece of equipment. Like with the UOPSS (Kelly, 2004) and UPCS (Wu et al., 2016), the PSG introduces two special types of units: source and sink units. Source and sink units govern the material/energy in and out flows, respectively, for their parent unit (or at the top-most level, the overall flowsheet). Specifications on feed or product qualities may be found here.

Unlike previous representations, PSG units can be nested. We term the containing unit as the "parent" unit, and its "children" are the units nested within. The set $\mathbf{U}_u$ denotes the children of unit $u \in \mathbf{U}$. This feature enables the PSG to more readily capture the hierarchical mental model inherent in most flowsheets, as well as to facilitate features for improved tractability. An example of nested units can be found in Figure 4, where unit 2 represents the entire reaction section, including both reaction alternatives. PSG supports unit nesting to an arbitrary depth, as a nested unit may also contain its own child units.

When units are nested, source and sink nodes are created to correspond to each of the parent unit's ports. We highlight the relationship between these nodes and their parent unit's ports in Figure 4, and subsequently, with a dashed box. Adding these source and sink nodes increases the total number of units and ports required for a PSG representation. In modern computational tools, these extra representational aids add little complexity, and can be readily pre-

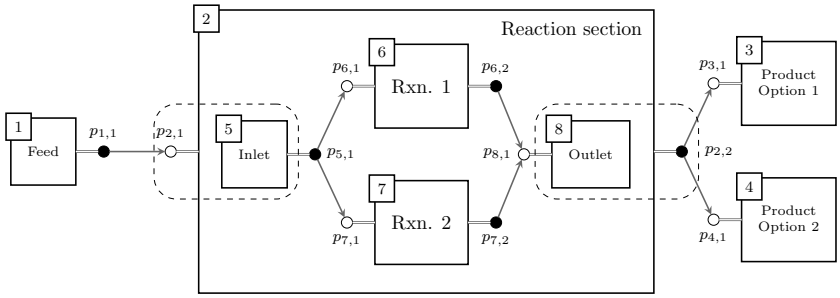
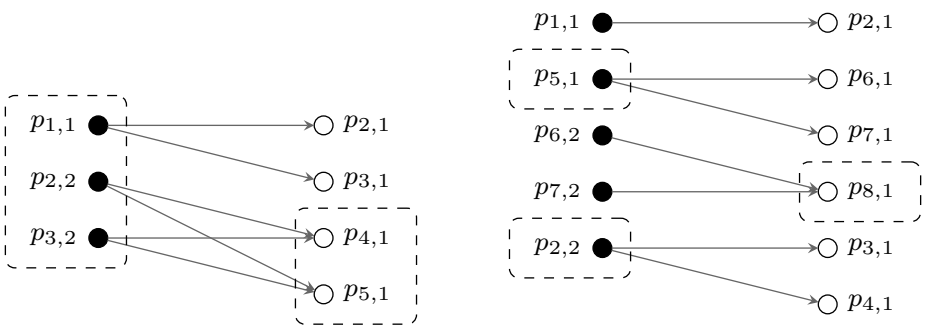


Figure 4: PSG representation with nested flowsheet units. Units 5, 6, 7, and 8 are children of parent unit 2, representing the reaction section of the flowsheet. Dashed boxes indicate the association between source unit 5 with parent port $p_{2,1}$, and sink unit 8 with parent port $p_{2,2}$, respectively.

processed. In fact, we can even observe in Figure 5 that the bipartite graph corresponding to the nested representation (Figure 5b) has fewer high-degree ports than the unnested representation (Figure 5a). Due to automated simplifications possible with degree 1 ports (see Section 5.4), unit nesting reduces complexity. Support for unit nesting also affords the PSG representation the ability to conceptually group major decisions. Moreover, using these auxiliary units (source/sink nodes associated with parent ports), streams never cross unit boundaries, maintaining clearer interface boundaries. This nesting also allows us to introduce "single-choice" units, describing process sections in which the optimal flowsheet involves selection of only one unit among a set of alternatives, as may be the case with reactors in Figure 4. These features play into tractability considerations for the PSG, as explained later in Section 5.2, where we also elaborate further on single-choice units.



(a) PSG without unit nesting (see Figure 3).

(b) PSG with unit nesting (see Figure 4).

Figure 5: Bipartite representation of the outlet and inlet ports in the illustrative superstructures. Ports with degree > 1 are highlighted, demonstrating reduced complexity through unit nesting.

In general, control volumes are understood to abide by material and energy conservation. However, in superstructure flowsheet representations, some material and energy flows of lesser significance are often abbreviated away, implicitly handled in the eventual mathematical model for the unit or simply ignored. For example, a separation task in the STN (Kondili et al., 1993) does not explicitly show the energy flows necessary for a distillation. Energy flows are also implicit in the UPCS (Wu et al., 2016), both in the units and the conditioning streams. The PSG representation also supports this mode of use, with units permitted to internally define handling of these quantities. In nested units, these relationships can manifest as source and/or sink units not associated with a port on their parent unit, creating implicit ports. However, in keeping with the principles of Pyosyn, we encourage an explicit characterization of unit interfaces, when possible.

3.1.2. Ports and streams

Ports define the physical or conceptual interfaces on a unit's control volume, through which material and/or energy flow may take place. We denote the set of ports $\mathbf{P}$. The enumeration of these interfaces for each unit u is given by the set of port numbers $\mathbf{PN}_u$. A port is uniquely defined by both its associated unit u and its port number $pn \in \mathbf{PN}_u$. In continuous interfaces where an infinite number of ports might be justified (e.g. membrane systems), PSG ports may be introduced at collocation points, or the interface may be manually defined using custom constraints in a membrane unit.

We define two main types of ports: inlet ports $\mathbf{P}_{in}$ and outlet ports $\mathbf{P}_{out}$, corresponding to flows into a unit and out from a unit, respectively. An inlet port functions as a multi-stream mixer, and an outlet port functions as a multi-stream splitter. We do not consider bidirectional flow in this work, so these sets are disjoint: $\mathbf{P}_{in} \cap \mathbf{P}_{out} = \varnothing$. For convenience, we also define the set of inlet ports connected to outlet port $p \in \mathbf{P}_{out}$ as $\mathbf{P}_p$, and the set of the outlet ports connected to inlet port $p' \in \mathbf{P}_{in}$ as $\mathbf{P}_{p'}$. Among these, we also distinguish between ports for material flow $\mathbf{P}^m$ and those for purely energy flow $\mathbf{P}^e$.

In the PSG representation, streams serve only the conceptual function of defining feasible connections between outlet ports and inlet ports. That is, streams are implicitly defined by the set of feasible pairings between outlet ports and inlet ports, $s \in \mathbf{S} \subseteq \mathbf{P}_{out} \times \mathbf{P}_{in}$. If a stream does not exist, no flow may take place between the corresponding outlet-inlet port pair. The combination of ports and streams in the PSG creates a bipartite graph, from the outlet ports to the inlet ports, similar to that of the UOPSS (Kelly, 2004) or UPCS (Wu et al., 2016) representations.

4. Superstructure generation

Means-ends analysis (MEA) forms the basis for most superstructure generation approaches. First introduced by Newell and Simon (1961), and applied to chemical process synthesis problems by Siirola et al. (1971), MEA describes the

recursive action of identifying tasks that bring an initial state (e.g. raw materials) to a final state (e.g. desired products). MEA sees two main modes of use in superstructure generation: library-assisted or standalone. PSG is compatible with both modes of generation. In the library assisted mode, MEA is used in conjunction with a library of unit models, potentially augmented with the inclusion of custom models. MEA selects the library units (and, if applicable, their respective number of occurrences) relevant to the design problem. From this set of library units, the most straightforward generation approach is to form the fully-connected graph between all outlet ports to all inlet ports, then to screen based on connectivity rules. This strategy is adopted by Wu et al. (2016) for the UPCS superstructure, and we describe the equivalent adaptation in Pyosyn for the PSG below. The P-Graph also uses MEA to determine the appropriate connectivity for the superstructure, applying the algorithm detailed by Friedler et al. (1993).

Pyosyn leverages the IDAES unit model library (Lee et al., 2018). Knowledge libraries (ontologies) in concert with model libraries have the potential to automate parts of the superstructure generation procedure (Morbach et al., 2007). However, as Pyosyn does not yet link to any such knowledge libraries, we rely on user input for to select the appropriate library and custom superstructure units. Using this model library and custom user models, the following steps describe generation of the desired PSG superstructure.

1. Generate all relevant major processing alternatives (classically, reaction and separation units)
2. Generate all possible streams
3. Screen possible streams based on port annotations
4. Generate supplementary conditioning alternatives (classically, heat exchangers, compressors, pumps, and turbines)
5. Generate necessary subsystems

First, we select all library units that describe major processing tasks needed. We also group any top-level units that participate in mutually exclusive relationships by nesting them in single-choice units. We then generate candidate streams between all top-level outlet and inlet ports. Note that unit nesting reduces the number of streams in the top-level flowsheet, as observed in the comparison between Figure 3, with 6 top-level streams, and Figure 4, with 3 top-level streams.

We next screen the set of candidate streams based on compatibility of the port annotations between their outlet and inlet ports. The P-Graph approach defines screening based on material state identities (Friedler et al., 1993). The UPCS introduces an approach based on material species identities, defining *minimal* and *feasible* components for each port (Wu et al., 2016). Minimal components are known to be present if the port exists, and feasible components are those that are admissible, but not necessary. We generalize this approach

based on the broader functionality of ports in the PSG representation. We denote port annotations as general specifications placed on a port based on species categorizations, port conditions, and port/unit identity. Other annotations may be defined, with the understanding that port existence is conditional on satisfaction of the annotations.

For material flow ports, we define three subsets in the set of chemical components $\mathbf{C}_p$ (also referred to as "species") that may be present at a port $p \in \mathbf{P}^m$. An example of these classifications in use may be found later in Section 7.2.

1. *Primary* ($\mathbf{C}_p^I$): species that must be present at the port when it is active. An example would be the reactants for a reactor inlet port.

2. *Secondary* ($\mathbf{C}_p^{II}$): species that may optionally be present in significant concentrations, but are not critical to the proper function of a port for its associated unit. An example would be an inert compound at a reactor inlet port.

3. *Residual* ($\mathbf{C}_p^X$): species that may be present in very small concentrations at a port, but do not serve a useful purpose for the port or its associated unit.

We also define the set $\mathbf{C}_p^\star = \mathbf{C}_p^I \cup \mathbf{C}_p^{II}$ as the set of *useful* species at port $p \in \mathbf{P}^m$, and the set $\mathbf{C}_p^F = \mathbf{C}_p^\star \cup \mathbf{C}_p^X$ as the set of *feasible* species at port $p \in \mathbf{P}^m$. Useful species at a port often serve some tangible function for the associated unit. For example, even if an inert does not participate in a reaction, it may function to moderate the temperature change from the heat of reaction. Infeasible chemical species $c \notin \mathbf{C}^F$ are those that should not be present in detectable quantities, for example, due to safety or material incompatibility concerns. Our material port species category annotations are compatible with those given for UPCS ports (Wu et al., 2016). We regard *minimal* UPCS components as *primary* PSG species, and *feasible* UPCS components are also regarded as *feasible* for PSG.

Based on these definitions, we propose the following set of port connectivity rules for the PSG. The initial four are adapted from (Wu et al., 2016), with additional rules added to exploit the nuance between *feasible* and *useful* species in the PSG, and support other PSG extensions.

Rule 1. All species feasible in outlet port $p \in \mathbf{P}_{p'}$ must be feasible in inlet port $p' \in \mathbf{P}_{in}^m$, if there exists a stream connecting p and p'.

$$\mathbf{C}_p^F \subseteq \mathbf{C}_{p'}^F, \qquad \forall (p, p') \in \mathbf{S}^m \tag{1}$$

Rule 2. All primary species in inlet port $p' \in \mathbf{P}_{in}$ must exist in the union of useful species among connected outlet ports $p \in \mathbf{P}_{p'}$.

$$\mathbf{C}_{p'}^I \subseteq \bigcup_{p \in \mathbf{P}_{p'}^m} \mathbf{C}_p^\star, \qquad \forall p' \in \mathbf{P}_{in}^m \tag{2}$$

Rule 3. All outlet ports $p \in \mathbf{P}^m_{p'}$ connected to a reactor inlet port $p' \in \mathbf{P}^m_u, u \in$ {reaction units} must provide at least one of the inlet port's *useful* species.

$$\mathbf{C}^\star_p \cap \mathbf{C}^\star_{p'} \neq \varnothing, \qquad \forall p' \in \mathbf{P}^m_u \cap \mathbf{P}^m_{in}, \forall u \in \text{reaction units}, \forall p \in \mathbf{P}^m_{p'} \tag{3}$$

Rule 4. All outlet ports $p \in \mathbf{P}^m_{p'}$ connected to a separator inlet port $p' \in \mathbf{P}^m_u, u \in$ {separation units} must provide all primary inlet port species.

$$\mathbf{C}^\star_p \cap \mathbf{C}^I_{p'} = \mathbf{C}^I_{p'}, \qquad \forall p' \in \mathbf{P}^m_u \cap \mathbf{P}^m_{in}, \forall u \in \text{separation units}, \forall p \in \mathbf{P}^m_{p'} \tag{4}$$

Rule 5. All primary components of outlet ports $p \in \mathbf{P}^m_{p'}$ connected to inlet port $p' \in \mathbf{P}^m_{in}$ must be *useful* inlet port species.

$$\mathbf{C}^I_p \subseteq \mathbf{C}^\star_{p'}, \qquad \forall (p, p') \in \mathbf{S}^m \tag{5}$$

Rule 6. Only connect material ports $p \in \mathbf{P}^m$ to other material ports.

$$p \in \mathbf{P}^m \iff p' \in \mathbf{P}^m, \qquad \forall (p, p') \in \mathbf{S} \tag{6}$$

Rule 7. All material ports $p \in \mathbf{P}^m$ must have at least one *useful* species.

$$\mathbf{C}^\star_p \neq \varnothing, \qquad \forall p \in \mathbf{P}^m \tag{7}$$

Note that Rules 1 and 3-6 primarily serve to screen out infeasible or impractical streams, while Rules 2 and 7 screen out entire ports, and their associated units. Rules 2 and 7 therefore primarily serve an error-checking purpose, as incorrect or insufficient units may have been identified in Step 1 of generation. In this work, we do not address additional rules that may govern energy ports $\mathbf{P}^e$.

For additional error checking, note that the generality of port annotations allows for the definition of feasible flow quantity ranges for each species. Consider the extensive flow quantity, $f_{p,c}$, of component c from port p. A *primary* species would have a flow lower bound, $f^{LB}_{p,c}$, greater than zero in the PSG, while a *feasible* component would have a flow upper bound, $f^{UB}_{p,c}$, greater than zero. Through the use of interval arithmetic, a second screening layer is possible. Each inlet/outlet port $p \in \mathbf{P}$ in the superstructure must satisfy the following relationships in Equations (8) and (9) with respect to its corresponding connected outlet/inlet ports $p' \in \mathbf{P}_p$:

$$f^{LB}_{p,c} \leq \sum_{p' \in \mathbf{P}_p} f^{UB}_{p',c}, \qquad \forall p \in \mathbf{P}, \forall c \in \mathbf{C}_p \tag{8}$$

$$f^{UB}_{p,c} \geq \sum_{p' \in \mathbf{P}_p} f^{LB}_{p',c}, \qquad \forall p \in \mathbf{P}, \forall c \in \mathbf{C}_p \tag{9}$$

That is, the lower bound for flow of species c at port p must be less than the sum over the flow upper bounds for species c at its connected ports. If p is an outlet port, this means that the maximum capacity across all of its outlet

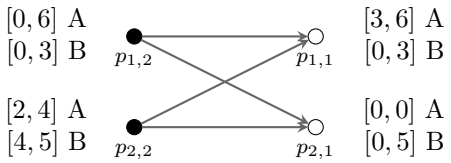


Figure 6: Bipartite graph representation of streams between outlet ports (on left) and inlet ports (on right), with annotations indicating feasible flow ranges for each species.

streams must be sufficient to satisfy its minimum flow. Similarly, if p is an inlet port, maximum capacity across inlet streams must satisfy minimum flow. As an illustration, consider the simple scenario depicted in Figure 6. Here, the stream between $p_{2,2}$ and $p_{2,1}$ would be eliminated by Rule 1, since *feasible* component A in $p_{2,2}$ is not *feasible* for $p_{2,1}$, after which applying Equation (8) on component B for port $p_{2,2}$ would flag the port as infeasible. By itself, this serves mostly as an error-checking tool, highlighting units in the superstructure that may be implausible due to limitations on species flow rates. However, in concert with simple input-output functions approximating the process units, bounds-tightening (Puranik and Sahinidis, 2017) can be performed on the resulting mass balance model of the superstructure as an additional screening step. Port condition annotations are also possible, such as admissible temperature and pressure ranges. Enforcement of these conditions are usually deferred until the next step, since multiple copies of conditioning unit types often occur in a flowsheet. However, if the conditioning units are pre-specified, then these condition ranges may also be used in the screening step. Finally, we also describe annotations for port/unit identity specifications. These specifications hold, for example, that material ports should connect to other material ports, rather than direct to energy ports.

Generation of the PSG superstructure can be implemented using the networkx package (Hagberg et al., 2008) from graph representations in Python, therefore facilitating easy data transfer with other Pyosyn components in Python. Each unit port would appear in the graph as a node, and the streams would be arcs between the nodes. Networkx node attributes may be used to store port annotations and the reference to a port's parent unit. Release-quality framework code to support use of these connectivity rules remains an open opportunity, but provided the appropriate species classifications as input, these rules may be seen as sequentially applied filters on a list of valid streams. Exclusion of a port would imply its exclusion from all of its relevant streams.

After screening, we insert temperature and/or pressure conditioning by applying MEA based on conditioning annotations of the outlet and inlet ports. This modifies the superstructure graph, as the ports will no longer be directly connected; instead, they will be connected through one or more intermediate conditioning units. The final valid candidate streams, with appropriate inser-

tion of conditioning operations, thus form the high-level superstructure for the process. This procedure is then recursively applied to any blocks containing nested units.

Note that this class of generation approaches—postulating all connections and then screening—contains the special case of omitting the screening procedure. In the interest of preserving generality of the representation in the central trade-off, some authors prefer this approach. However, this often comes at great cost to tractability. As a result, generality is often lost later in the design process, when the solution approach is unable to guarantee an implicit enumeration over the entire design space described by the superstructure. As a result, the generation step still often requires manual input of engineering knowledge in the screening phase to be effective. Wu et al. (2016) introduce manually specified "non-conditioning" streams, "single-stream" ports, and conditioning reduction due to these considerations. Expert systems tools can aid in screening, with the caveat that the optimal configuration may be prematurely excluded, impacting generality. And eventually, solution algorithms may advance to the point that the screening step is no longer necessary for a broad range of problems. Until that time, however, we also provide decision-maker flexibility to augment the screening stage for PSG, in addition to systematic screening based on port annotations.

The other proposed superstructure generation approach is a hierarchical strategy to directly use MEA to postulate a PSG superstructure. In this strategy, we adapt levels 2-5 of the hierarchical decomposition approach given by Douglas (1985), and derive process data and problem specifications from user knowledge. Rather than fixing certain choices at each decision level, we introduce the described alternatives as units in the superstructure. Note that common criticisms of hierarchical decomposition still apply—this technique may not identify integrated solutions such as reactive distillation, unless they are manually specified. However, for systems in which decision makers are knowledgeable of the relevant alternatives, the direct MEA approach offers a straightforward alternative.

We demonstrate a comparison of these generation approaches in Section 7.2. The end result of the superstructure generation phase is a graph representation, describing potential connections between alternative flowsheet units.

5. Logical/Algebraic model formulation

From the superstructure representation, a mathematical programming model must be formulated to solve for the optimal flowsheet design. While the choice of superstructure has the greatest impact on generality in terms of the central trade-off of process synthesis, the modeling step often involves making a trade-off between model fidelity and tractability.

Due to the prevalence of nonlinear relationships, and both continuous and discrete decisions variables, the process design problem is historically formulated

as a Mixed-Integer Nonlinear Programming (MINLP) model (Trespalacios and Grossmann, 2014). The general form is as follows:

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

Here, we minimize an objective function $f(x,y)$, subject to inequality constraints (e.g. from process specifications) $g(x,y) \leq 0$ and equality constraints (e.g. from material, energy balances and thermodynamic relationships) $h(x,y) = 0$. Note that maximization may be achieved by minimizing the negative of the objective function. x are the continuous decision variables (e.g. flows and equipment sizes), while y are the discrete decision variables, usually corresponding to selection or omission of units and/or interconnections in the superstructure. Long-time practitioners of modeling for superstructure synthesis may be accustomed to expressing logic as an MINLP. Many legacy models may also exist as MINLPs. Furthermore, a wide variety of solver codes linked to algebraic modeling platforms exist for MINLP (e.g. DICOPT (Viswanathan and Grossmann, 1990), SBB (Bussieck and Drud, 2001), BARON (Tawarmalani and Sahinidis, 2005), SCIP (Vigerske and Gleixner, 2018), ANTIGONE (Misener and Floudas, 2014)). As a result, Pyosyn accommodates specification of specific units or even the whole superstructure as an MINLP. In fact, custom MINLP solvers, such as MindtPy (Bernal et al., 2018), may also be employed for Pyosyn synthesis problems (see Section 6).

However, in the interest of providing high-level, intuitive representations, the recommended modeling approach for Pyosyn is using Generalized Disjunctive Programming (GDP), the extension of disjunctive programming (Balas, 1985) for nonlinear functions (Grossmann and Trespalacios, 2013; Trespalacios and Grossmann, 2014). The general form for a GDP model can be found below:

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

Here, we minimize an objective function $f(x, z)$. Variables x again describe continuous decisions (e.g. flows and equipment sizes). Global constraints $g(x, z) \leq 0$ describe specifications and physical relationships that apply for all feasible configurations in the superstructure. However, with GDP, disjunctions K (corresponding to logical-OR relationships) describe selection among process alternatives. Selection of a given alternative i in disjunction k is indicated by the Boolean variable Y_{ik}. When an alternative is selected, its corresponding constraints $r_{ik}(x, z) \leq 0$ are enforced. In most cases, we want to select one alternative within a given set of options; therefore, we can enforce a logical-XOR relationship such that at most one disjunct Y_{ik} in each disjunction K has value $True$. Finally, other types of logical relationships may be described using logical propositions $\Omega(Y) = True$.

The form of GDP that we adopt for Pyosyn also allows for auxiliary discrete variables z, which may be used to describe the count of a particular operation within a process unit. The impact of these variables is explored in (Chen et al., 2020). Conveniently, this also allows us to regard MINLP as a special case of GDP.

Due to this expressiveness, GDP is well-suited for synthesis problems (Chen and Grossmann, 2019). Explicit modeling of disjunctions (logical-OR relationships) in GDP describe existence and absence of a superstructure unit. Furthermore, through the use of logical propositions, other logical conditions governing unit selection may be described, for example, that selection of a particular reaction technology $Y_{rxn,A}$ implies the selection of a corresponding separation approach $Y_{sep,A}$: $Y_{rxn,A} \implies Y_{sep,A}$. Pyosyn utilizes Pyomo.GDP (Chen et al., 2018, 2020) to provide a software platform with first-class support for GDP modeling.

5.1. Modeling PSG units

External interfaces for each PSG unit to other flowsheet units are defined by its unit ports. Therefore, the PSG unit model must define the relationship between its unit-facing port variables both when the superstructure unit exists, and when it is absent from the selected flowsheet. Using GDP, a very intuitive formulation for the PSG unit is provided by using disjunctions. In fact, for process synthesis problems, the GDP model can also be expressed in the form (GDP').

$$
\begin{aligned}
\min\, & obj = f(x, z) \\
s.t.\quad & g(x, z) \leq 0 \\
& \begin{bmatrix} Y_k \\ r_k(x, z) \leq 0 \end{bmatrix} \vee \begin{bmatrix} \neg Y_k \\ B^k x = 0 \end{bmatrix}, \quad \forall k \in K \\
& \Omega(Y) = True \\
& x \in X \subseteq \mathbb{R}^n \\
& Y_k \in \{True, False\} \quad \forall k \in K \\
& z \in Z \subseteq \mathbb{Z}^m
\end{aligned}
\tag{GDP'}
$$

Here, the disjunctions K are posed in terms of existence or absence of superstructure elements. $Y_k = True$ can denote existence of a unit, and $Y_k = False$ its absence. If a unit exists, the constraints $r_k(x, z) \leq 0$ enforce the relevant mass and energy balances, thermodynamics, kinetics, or other physical/chemical phenomena taking place within the unit. Otherwise, constraints $B^k x = 0$ describe port variable relationships when the unit is absent. Two common approaches exist for modeling an absent unit: bypass and no-flow. In a bypass model, the unit exit state is set equal to the inlet state. However, Farkas et al. (2005) show that this can lead to unnecessary structural redundancy. Therefore, in Pyosyn, these constraints usually enforce no-flow; that is, extensive port variables corresponding to the unit are set equal to zero. However, in some circumstances, the bypass model can be advantageous (see section 7.3). Therefore, in keeping with the principle of flexibility, Pyosyn allows for both approaches.

Note that support for nested units in Pyosyn, coupled with the use of unit ports to define interfaces, allows a very intuitive modular construction for flowsheets, helping to manage complexity (Friedman et al., 2013). With clearly defined interfaces, surrogate models are also easier to integrate, as they must simply define the relationships between the port variables of a unit. Furthermore, Pyosyn's parent framework, IDAES (Miller et al., 2018), provides several tools that generate surrogate models expressed in the Pyomo (Hart et al., 2017) algebraic modeling language, which are therefore readily compatible with Pyosyn models in Pyomo.GDP. Selection of the level of modeling detail is left to the user's discretion.

5.2. Single-choice units

Support for nested units in Pyosyn also allows for definition of single-choice unit. A central goal in conceptual design is the selection of the optimal processing equipment/technology from among a set of alternatives. For example, in Figure 4, the reaction section involves the selection of reaction technology 1 or 2. In most cases, it is desirable to select only one of the processing alternatives for a given flowsheet subsection. When this is true, Kocis and Grossmann (1989) identify ports $p_{5,1}$ and $p_{8,1}$ as "single-choice" interconnection nodes, facilitating the use of linear material and energy balances at these ports. With the PSG, we leverage support for nested units to define single-choice units (e.g. Unit 2 in Figure 4), in which this single choice relationship governs selection among its non-auxiliary child units. All ports on auxiliary source and sink units (units 5 and 8 in Figure 4) in the single choice unit can thus be automatically identified as single-choice interconnection nodes. Visually, the single-choice units also make clearer where major decision elements exist, and group together superstructure elements that likely have similar functions.

5.3. Modeling PSG unit ports

Unit ports in PSG act as general purpose mixers and splitters at the unit interface. Existence or absence of the unit port $p_{u,pn}$ is typically logically equivalent to existence or absence of its associated unit u, and thus can share the

same Boolean variable Y_u. Handling of mixing/splitting calculations when the port exists is included within the constraints $r_u(x, z) \leq 0$, and port absence in the linear constraints $B^u x = 0$.

For both inlet and outlet ports, we distinguish handling of intensive and extensive flow variables. Intensive variables describe properties such as temperature and pressure, which do not vary with the rate of flow; extensive variables such as the species molar flowrate, on the other hand, are dependent on the flow quantity (Biegler et al., 1997). Intensive variables are easy to handle for splitters, but may involve more complex relationships for mixers. On the other hand, extensive variables require special handling for splitters, but are simple for mixers. As a result, a multi-component material balance will always involve complexity at the mixers or the splitters, depending on the choice of flow representation between total flow and species compositions, or species component flows (Quesada and Grossmann, 1995).

Inlet ports represent general purpose mixers at the unit interface with other process units. Extensive inlet port variable $x_{e,p'}$ is calculated simply as the sum of flows from all connected outlet ports, as in Equation (10).

$$x_{e,p'} = \sum_{p \in \mathbf{P}_{p'}} x_{e,p,p'}, \qquad \forall p' \in \mathbf{P}_{in}, \forall e \in \text{extensive port variables} \tag{10}$$

Intensive inlet port variable $x_{i,p'}$, however, must be calculated based on a general function $f_{i,p'}$ of both intensive and extensive stream variables.

$$x_{i,p'} = f_{i,p'}(x_{p,p'}), \qquad \forall p' \in \mathbf{P}_{in}, \forall i \in \text{intensive port variables} \tag{11}$$

For example, temperature must be calculated based on an energy balance that may involve stream temperatures as well as the enthalpy of mixing. Note that an isobaric assumption is often made for mixers in conceptual design. Equation (11) for the pressure $P_{p'}$ of inlet port p' would then simplify to

$$P_{p'} = P_p, \qquad \forall p' \in \mathbf{P}_{in}, \forall p \in \mathbf{P}_{p'} \tag{12}$$

Outlet ports represent general purpose splitters at the unit interface with other units. We assume that flow through the outlet port is well-mixed. Therefore, intensive outlet port variables $x_{i,p}$ are equal to their corresponding flow variables to all connected inlet ports, as in Equation (13).

$$x_{i,p} = x_{i,p'}, \qquad \forall p \in \mathbf{P}_{out}, \forall p' \in \mathbf{P}_p, \forall i \in \text{intensive port variables} \tag{13}$$

The extensive outlet port variables $x_{e,p}$, however, must be linked to the flow variables through the use of a split fraction $SF_{p,p'}$, denoting the fraction of flow directed towards each connected inlet port.

$$x_{e,p} SF_{p,p'} = x_{e,p,p'}, \qquad \forall p \in \mathbf{P}_{out}, \forall p' \in \mathbf{P}_p, \forall e \in \text{extensive port variables} \tag{14}$$

This split fraction must also sum to unity.

$$\sum_{p' \in P} SF_{p,p'} = 1 \tag{15}$$

Note that we present the component molar flow formulation with split fractions here, but total flow and species compositions are also supported in Pyosyn (Quesada and Grossmann, 1995).

5.4. Special cases

Two special cases are relevant in modeling unit ports: when only one extensive flow variable exists, and when only one stream is connected to the port.

When there exists only one extensive variable, the split fraction in Equations (14) and (15) is not necessary. Therefore, the two equations can be replaced by

$$x_{e,p} = \sum_{p' \in \mathbf{P}_p} x_{e,p,p'}, \qquad \forall p \in \mathbf{P}_{out}, e = \text{single extensive port variable} \tag{16}$$

This can occur for material ports in single-component systems, for example, in utility plant design (Bruno et al., 1998).

The second case is when only one stream is connected to the port. In this case, because no state change occurs along PSG streams, the port variables are simply equal to their counterparts among the stream variables.

$$\begin{aligned} x_p &= x_{p,p'}, \qquad \forall p \in \mathbf{P}_{out}, p' \in \mathbf{P}_p, |\mathbf{P}_p| = 1 \\ x_{p'} &= x_{p,p'}, \qquad \forall p' \in \mathbf{P}_{in}, p \in \mathbf{P}_{p'}, |\mathbf{P}_{p'}| = 1 \end{aligned} \tag{17}$$

In Pyosyn, detection of these special cases can be automated through the use of Pyomo.Network, which provides both the ability to distinguish between intensive and extensive variables, and the ability to adjust the port model based on the number of connected streams.

6. Pyosyn solution strategies

Due to the presence of nonlinearities arising from physical property calculations and mixing/splitting, along with discrete flowsheet topology decisions, advanced solution strategies are required to solve mathematical programming models for process synthesis. One of the most challenging characteristics of flowsheet synthesis problems for modern optimization solvers arises from "zero flow" singularities, discussed in further detail in Appendix A.

For process design problems, tailored algorithms can offer improved performance compared to general purpose solution strategies; however, they may not always be appropriate, as commercial general purpose solvers may be better optimized (from a software engineering perspective), allowing them to execute more quickly. Pyosyn therefore implements a wide range of customizable solution strategies (see Figure 7).

The traditional approach is a reformulation of the GDP into an MINLP model, followed by the use of an MINLP solver (Trespalacios and Grossmann, 2014). Two canonical forms are described in literature: the Big-M (BM) reformulation (Nemhauser and Wolsey, 1988; Raman and Grossmann, 1994) and the

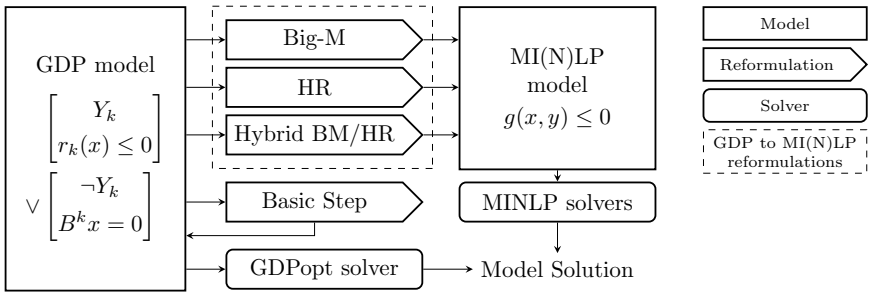


Figure 7: Pyosyn flexible solution strategies. Figure adapted from (Chen et al., 2018).

Hull Reformulation (HR) (Grossmann and Lee, 2003). BM results in a smaller formulation, since it does not require the introduction of additional variables, which may help reduce the required solution time. On the other hand, the HR gives a tighter continuous relaxation, which may reduce the number of iterations needed for a solution algorithm to converge. This comes at the expense of additional disaggregated variables and their corresponding constraints. The appropriate choice of reformulation is problem-specific, and difficult to determine a priori. One formulation may be tractable, while the other is not. Therefore, Pyosyn—via Pyomo.GDP (Chen et al., 2020)—provides the capability to automatically reformulate a GDP with either BM or HR. In addition, advanced hybrid techniques based on a cutting plane algorithm (Trespalacios and Grossmann, 2016) are also implemented (Chen et al., 2018).

Once reformulated as an MINLP model, traditional mathematical programming solvers may be used, including several commercial alternatives (Tawarmalani and Sahinidis, 2005; Vigerske and Gleixner, 2018; Misener and Floudas, 2014; Viswanathan and Grossmann, 1990). However, specialized algorithms, custom implementations, and experimental prototypes may also be directly used (Bernal et al., 2018; Muts et al., 2020; Mitsos et al., 2018).

Tightening reformulations in the logical space can also be preformed as a preprocessing step to other solution paths, via an operation referred to as a "basic step" (Ruiz and Grossmann, 2012). This brings the GDP formulation closer to disjunctive normal form and improves its continuous relaxation, at the expense of increasing the number of disjuncts.

Finally, direct logic-based solution approaches are possible via the GDPopt solver (Chen et al., 2020). GDPopt provides modern implementations of the logic-based outer approximation (LOA) (Türkay and Grossmann, 1996) and logic-based branch-and-bound (LBB) (Lee and Grossmann, 2000) decomposition algorithms. For LOA, GDPopt also implements the global optimization extension (GLOA) (Bergamini et al., 2005). These logic-based decomposition algorithms are particularly advantageous for synthesis problems, due to their ability to solve nonlinear subproblems in reduced space, thereby avoiding the zero flow numerical difficulties described in Appendix A.

Indeed, for flowsheet synthesis problems, the LOA algorithm may be broadly understood as the following iterative procedure:

0. Initialize: Based on the solution of NLP subproblems corresponding to different flowsheets that "cover" all of the disjunctions, construct a linear approximation of the full superstructure.
1. Master problem: solve this linear approximation to determine a new candidate flowsheet topology (unit and stream existence).
2. Subproblem: solve a reduced space nonlinear programming model corresponding to only the selected candidate flowsheet, obtaining accurate equipment sizes, operating conditions, and objective value.
3. Cut generation: generate additional linear inequalities (cuts) based on the subproblem solution values.
4. Iterate: repeat steps 1–3 above, adding the new cuts to the master problem, until the master problem and subproblem objective values converge.

Note that selecting the fewest number of flowsheets needed to generate the initial linearization of the master problem can be formulated as a set covering problem (Türkay and Grossmann, 1996). This set covering is distinct from a complete enumeration, as we only need to evaluate each nonlinear disjunct; we do not need to evaluate all possible *combinations* of the disjuncts. Alternative initialization approaches are also possible, but not explored here. Since the complicating nonlinear "zero flow" functions are only present in the subproblems, where complete flowsheets are optimized (and thus units should have non-zero flow), the LOA algorithm avoids zero flow numerical difficulties. In the master problem, when units and streams may "disappear" from the superstructure during the solution procedure, the complicating "zero flow" functions are not present. This results in a more robust strategy than the full-space MINLP approach. However, for problems where zero flow issues are not as pronounced, conventional MINLP approaches may be faster.

Therefore, given the range of solution strategies that may be preferable to others for a particular application, Pyosyn supports a flexible and extensible suite of options, as shown in Figure 7. Crucially, the user can also select among these solution strategies without needing to rewrite their GDP model, allowing them to quickly explore different options for their specific application.

7. Case studies

In the following case studies, we illustrate the benefits of Pyosyn's flexible implementation, applied to various use cases, with differing levels of customization. Note that computational results in these case studies reflect our ability to reproduce with Pyosyn approaches previously reported solutions by the examples' original authors and are not the emphasis in this work.

7.1. Eight Process Problem (8PP)

The Eight Process Problem (8PP) is a literature case study (Duran and Grossmann, 1986a, Example 3) involving the synthesis of a flowsheet from a superstructure of eight potential processing units. A diagram of the original STN superstructure can be found in (Türkay and Grossmann, 1996, Figure 3). In this case study, we use the 8PP to demonstrate the translation of an STN superstructure into the PSG representation, and to demonstrate logical modeling functionality in Pyosyn.

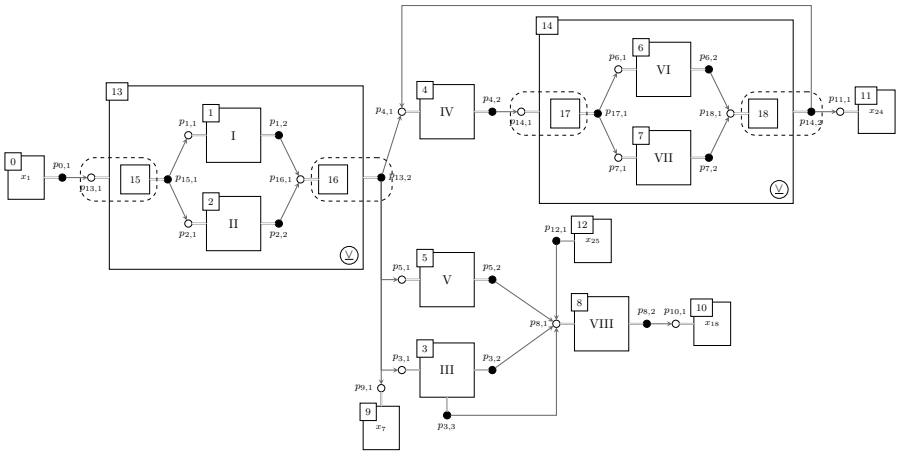


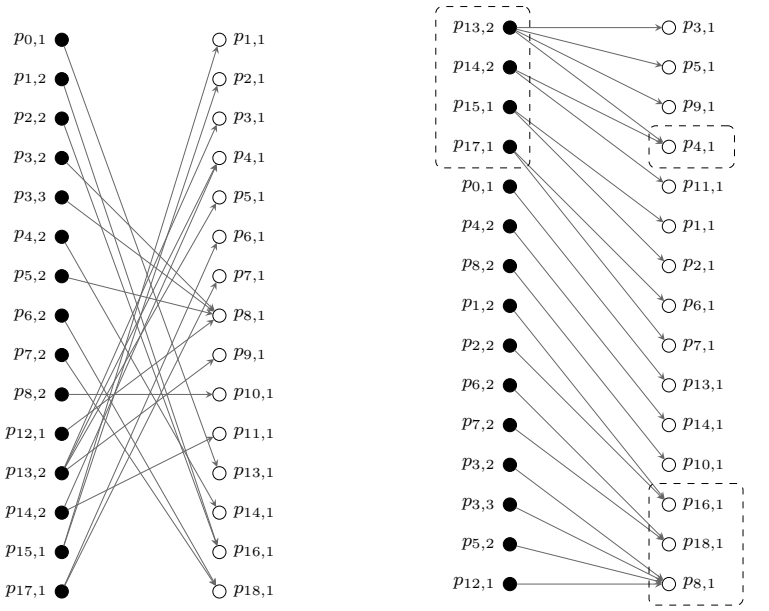
Figure 8: PSG superstructure for the 8PP.

We generate a custom PSG representation of the 8PP (see Figure 8), using the original STN as a reference. Unit numbering on the PSG superstructure is adjusted so that units 1-8 match the original eight processes. Note that the use of PSG highlights the single-choice relationship between processes 1 and 2, as well as between processes 6 and 7.

Given the graph structure of PSG, we can also visualize the superstructure as a bipartite graph of the outlet and inlet ports (see Figure 9). Notice that most ports have degree 1 (connected to only one other port) in this superstructure. Therefore, the automatic model simplifications implemented in Pyosyn (see Section 5.4) are relevant here. Moreover, since the 8PP is treated as a single-component system, with only one extensive variable per port, the other simplification described in Section 5.4 also applies.

The logical formulation of the 8PP results in a small GDP with 44 variables (12 Boolean, 32 continuous) and 52 constraints with 5 convex nonlinear functions (Türkay and Grossmann, 1996, Appendix A). We implement this model using the Pyomo.GDP component of Pyosyn, allowing us to directly specify propositional logic such as Equation (18) instead of manually converting it to algebraic form, as given in Equation (19).

$$(Y_1 \vee Y_2) \implies (Y_3 \vee Y_4 \vee Y_5) \tag{18}$$



(a) Ports sorted by unit number.

(b) Ports sorted by connections. Ports with degree > 1 are highlighted.

Figure 9: Bipartite representation of the outlet and inlet ports in the 8PP superstructure.

$$\begin{aligned} -y_1 + y_3 + y_4 + y_5 \geq 0 \\ -y_2 + y_3 + y_4 + y_5 \geq 0 \end{aligned} \tag{19}$$

This improves both the readability of the model and the ease with which modifications may be made in the future. Note that modelers experienced with building MINLP models may continue to do so from PSG representations, and their models may be solved either directly using commercial solvers, experimental implementations (Bernal et al., 2018), or custom solvers. Code corresponding to both GDP and MINLP models of the 8PP may be found in the examples of the public Pyomo Github repository (`https://github.com/Pyomo/pyomo`).

However, modeling in GDP also confers the advantage of a broader range of flexible solution alternatives (see Figure 7). Table 1 shows the solution times for the 8PP using a range of different solvers, performed on machine running Ubuntu 16.04 LTE with 12 cores across two sockets (Intel Xeon X5650 @ 2.66 GHz), with 128GB physical memory. All solvers converge to the optimal solution in a short amount of time. DICOPT (Viswanathan and Grossmann, 1990), BARON (Tawarmalani and Sahinidis, 2005), and SCIP (Vigerske and Gleixner, 2018) are all commercially available MINLP solvers, accessible to Pyosyn through an programmatic interface between Pyomo and GAMS (Brook et al., 1988). GAMS version 30.1.0 was used in this case study. Prior to solving

Solver	Time (s)
DICOPT	0.05
BARON	0.11
SCIP	0.35
LOA	1.39
GLOA	1.53
LBB	12.92

Table 1: Solution times for the 8PP using different solution strategies.

with these, the GDP model is automatically converted to MINLP using the BM reformulation in Pyomo.GDP. LOA (Türkay and Grossmann, 1996), GLOA (Lee and Grossmann, 2001), and LBB (Lee and Grossmann, 2000) correspond to their respective logic-based decomposition algorithm implementations in the GDPopt direct GDP solver (Chen et al., 2020). For the simple problems like the 8PP, the commercial MINLP solver implementations perform faster than their logic-based counterparts, since GDPopt is a meta-solver implemented in Python. Therefore, Pyosyn offers the capability to use these solvers as part of its flexible solution framework.

7.2. Methanol synthesis

A common use for syngas is the production of methanol, another common chemical intermediate. Here, we examine the methanol process synthesis problem defined in (Türkay and Grossmann, 1996, Example 3). We use the methanol synthesis problem to demonstrate superstructure generation in Pyosyn, as well as the value of logic-based decomposition algorithms made available in GDPopt.

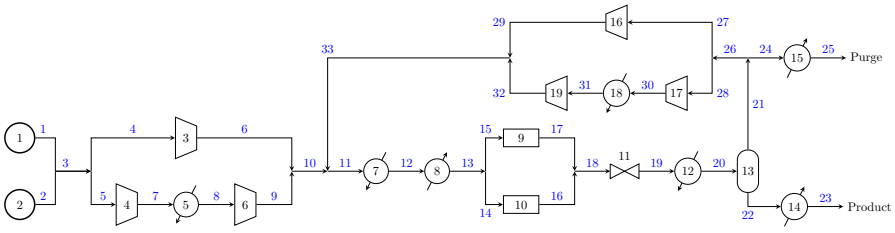


Figure 10: Methanol synthesis from syngas (traditional view). Figure adapted from (Chen and Grossmann, 2019, Figure 3).

The traditional STN superstructure for the methanol process is shown in Figure 10. We first generate a custom PSG representation based on the STN (see Figure 11). Four major structural decisions are present in the superstructure, highlighted by the use of single-choice units 1, 2, 4, and 8. Two alternative feed grades of syngas are available, as well as two different reactor sizes. The options for each set of alternatives are denoted by their relative costs as "cheap" and "expensive", respectively. The feed and recycle compressors can also each be

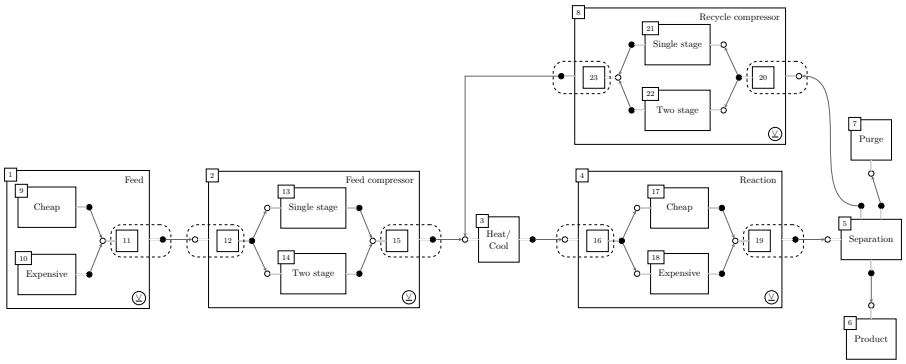


Figure 11: Methanol synthesis from syngas

designed as single-stage or two-stage compressors. The reactor conditioning unit is given in Figure 12, and the flash separation with product/purge conditioning is given in Figure 13.

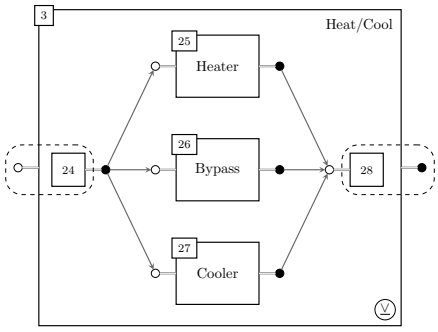


Figure 12: Single choice temperature conditioning block from methanol synthesis flowsheet

As described in Section 4, systematic generation of the methanol flowsheet can be achieved using means-ends analysis (MEA) assisted by a unit model library, or using an approach based on hierarchical decomposition. First, we examine the process for unit model library-assisted generation. Step 1 involves the identification of major processing alternatives. Here, we identify units corresponding to 2 feeds, 2 reactors, a flash separation, a product, and a purge. Since the two feeds and reactors participate in single-choice relationships with each other, we nest them in corresponding single-choice units. We list the resulting units and their associated processing tasks in Table 2 (see also Figure 14). Their unit ports, with their associated annotations, are given in Table 3.

In Step 2, we generate all possible streams in the fully connected graph between outlet and inlet ports, $\mathbf{P}_{out} \times \mathbf{P}_{in}$. We then screen the streams in Step 3 based on connectivity rules (see Section 4) using the port annotations given

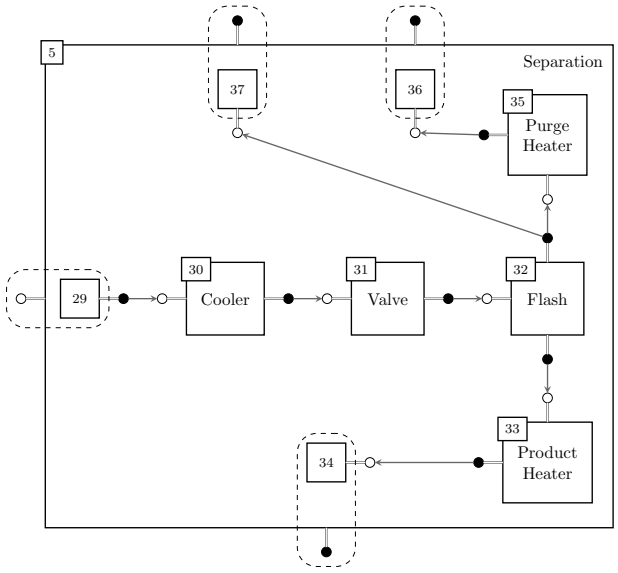


Figure 13: Separation block from methanol synthesis flowsheet

in Table 3. Note that all four components are annotated as "primary" for the reactor outlet, as the reaction is equilibrium-limited. The full list of considered streams, along with their rationale for exclusion, are given in Table 4. Note the use of engineering judgment, denoted by "ENGR" to exclude the direct feeding of raw material to the purge stream. The bipartite graph representation of this step is illustrated in Figure 15.

From the reduced set of candidate streams not excluded in Table 4, we apply means-ends analysis to identify temperature and pressure conditioning needs based on the annotations in Table 3. When the source and destination ports require temperature/pressure adjustment between two ranges, then the appropriate conditioning must be added. Of the six candidate streams, four need both temperature and pressure conditioning and two need only temperature conditioning (see Table 5). Based on required temperature and pressure ranges given by the port annotations, compressors and heaters are postulated between the feed and the reactor (1,1 $\rightarrow$ 2,1), as well as between the flash vapor outlet and the reactor (3,3 $\rightarrow$ 2,1). From the model library, a single-choice unit for compression is selected, which contains a nested decision between single- or two-stage compression. Between the reactor and the flash (2,2 $\rightarrow$ 3,1), a valve and cooler are postulated. Finally, as the product and purge do not have required pressure values, and the flash outlet streams are always at the same or lower temperature, heaters are postulated between the flash and product (3,2 $\rightarrow$ 4,1), and between the flash and purge (3,3 $\rightarrow$ 5,1).

A useful point is the ordering of the temperature/pressure conditioning operations. In the UPCS, these are fixed to an arrangement with temperature

Unit #	Name	Task
1	Single-Choice Feed	Supply raw material syngas
2	Single-Choice Reactor	Equilibrium-limited reaction $2H_2 + CO \rightarrow CH_3OH$
3	Flash	Separate liquid CH_3OH from vapor H_2, CO, CH_4
4	Product	Collect primary product CH_3OH
5	Purge	Collect purge CH_4

Table 2: Methanol synthesis: identified major processing units

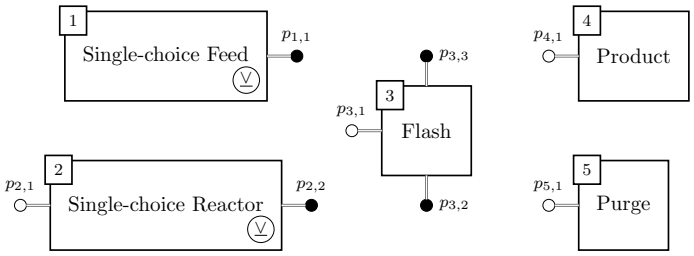


Figure 14: Methanol synthesis: identified major processing units

changes preceding pressure changes (Wu et al., 2016). This can be advantageous when there is a change of state required (e.g. prior to a compressor); however, this arrangement is not always ideal. With PSG, use of explicit conditioning units allows the determination to be made based on the means-ends analysis. For example, between the reactor outlet and the flash inlet, it may be advantageous to allow for pressure reduction through a valve before the cooler due to material cost and/or safety considerations. Furthermore, it may be advantageous to position the heater after compression for the feed and recycle streams to the reactor, as the two adjacent heaters corresponding to each stream can be identified via a graph adjacency search, and consolidated into a single reactor pre-heater. After this step, subsystem representations in nested blocks are generated by a similar procedure. In this case study, the auxiliary inlet and outlet blocks associated with respective parent ports are simply added to the single-choice blocks. The final superstructure generated by the library-assisted means-ends analysis has a similar structure to that given by the custom-generated representation presented earlier (see Figure 11).

We can also generate the methanol superstructure using hierarchical means-ends analysis in Pyosyn. Following the steps in Section 4, we first assess alternatives in the input-output structure. Here, we recognize that two grades of feed are available, provided in two different feed streams. The two feed options are mutually exclusive, so they are introduced as a single-choice source unit. The desired methanol product is also added as a stream. Given the gas-phase,

Port	Unit	Type	$\mathbf{C}_p^I$	$\mathbf{C}_p^{II}$	$\mathbf{C}_p^X$	Temperature	Pressure
1,1	Feed	out	A,B,D			300	1
2,1	Reactor	in	A,B,D		C	[423,873]	[2.5,15]
2,2	Reactor	out	A,B,C,D			[523,873]	[2.25,13.5]
3,1	Flash	in	A,B,C,D			[300,400]	[0.25,13.5]
3,2	Flash (L)	out	C		A,B,D	[300,400]	[0.25,13.5]
3,3	Flash (V)	out	A,B,D		C	[300,400]	[0.25,13.5]
4,1	Product	in	C		A,B,D	400	[0.25,13.5]
5,1	Purge	in		A,B,D	C	400	[0.25,13.5]

Table 3: Methanol synthesis: unit ports with annotations. Components: A=H_2, B=CO, C=CH_3OH, D=CH_4. Temperature is given in Kelvin, and pressure is in MPa.

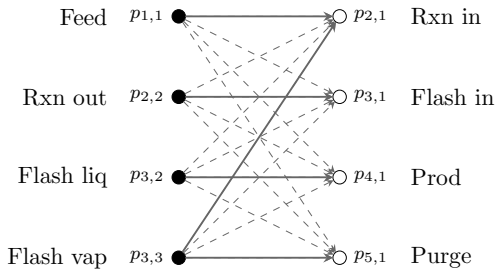


Figure 15: Ports and generated streams for the methanol synthesis superstructure. Solid lines indicate retained streams, while dashed streams are excluded by the described screening rules.

equilibrium-limited methanol reaction, a recycle stream and purge are postulated. Next, we assess the recycle structure and reactors. Only one reaction takes place, but two reactors alternatives are available, so they are added to the superstructure grouped as a single-choice unit. We identify the need for feed and recycle compression, each of which involves the choice between single-stage or two-stage compression. These are introduced to the superstructure as single-choice units. In the following stage, we assess the separation system options. Here, the separation is comparatively simple, and a flash vessel is used. We then evaluate the other conditioning needs in the superstructure by means-ends analysis, adding heaters/coolers and the flash valve. We do not consider heat integration in this problem, so the final step is not needed. Rather than imposing the decisions as described by Douglas (1985), we use the steps to generate the relevant superstructure alternatives, thus preserving more generality. The final generated superstructure using hierarchical means-ends analysis is similar to the original custom-generated representation given earlier.

After generating the superstructure representation, its logic must be encoded in mathematical form. The GDP model for the methanol problem is described by Türkay and Grossmann (1996), with recent extensions discussed by Chen and Grossmann (2019). We refer the reader to these papers for modeling details.

Outlet	Inlet	Exclusion Rationale
1,1	2,1	–
1,1	3,1	Rule 4: outlet does not provide primary species C to separator
1,1	4,1	Rule 5: outlet species A,B,D not useful for inlet
1,1	5,1	ENGR: do not feed directly to purge streams
2,2	2,1	Rule 5: outlet species C not useful for inlet
2,2	3,1	–
2,2	4,1	Rule 5: outlet species A,B,D not useful for inlet
2,2	5,1	Rule 5: outlet species C not useful for inlet
3,2	2,1	Rule 5: outlet species C not useful for inlet
3,2	3,1	Rule 4: outlet does not provide primary species A,B,D to separator
3,2	4,1	–
3,2	5,1	Rule 5: outlet species C not useful for inlet
3,3	2,1	–
3,3	3,1	Rule 4: outlet does not provide primary species C to separator
3,3	4,1	Rule 5: outlet species A,B,D not useful for inlet
3,3	5,1	–

Table 4: Methanol synthesis: screening candidate streams between unit ports. Components: A=H_2, B=CO, C=CH_3OH, D=CH_4. ENGR above indicates a use of engineering judgement.

				Conditioning	
Source	Port	Destination	Port	Temperature	Pressure
Feed	1,1	Reactor	2,1	Yes	Yes
Reactor	2,2	Flash	3,1	Yes	Yes
Flash L	3,2	Product	4,1	Yes	No
Flash V	3,3	Reactor	2,1	Yes	Yes
Flash V	3,3	Purge	5,1	Yes	No

Table 5: Methanol synthesis: conditioning needs for major candidate streams

Here, we implement the model in Pyosyn using Pyomo.GDP. The methanol synthesis model with 285 variables (8 Boolean, 277 continuous) and 429 constraints is computationally challenging. DICOPT (using either the BM or HR reformulations) is unable to provide even a feasible solution. SCIP and BARON are both able to identify the optimal solution, but they are not able to guarantee global optimality, as they have bound gaps of 81% and 145%, respectively, after an hour. GDPopt-LOA (using CPLEX and IPOPT as subsolvers) is able to identify the optimal solution after only 72 seconds, though, like DICOPT, it does not guarantee global optimality for nonconvex problems. However, it is possible to generate rigorous linearizations in GDPopt, while still using a local NLP solver, as was done in this case. Here, the logic-based outer approximation provides a robust approach to quickly obtain a high quality solution to the problem, demonstrating the value of solution flexibility in Pyosyn.

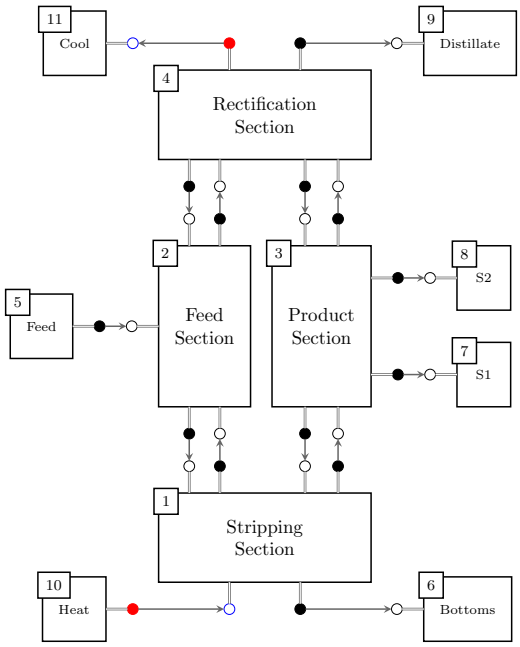


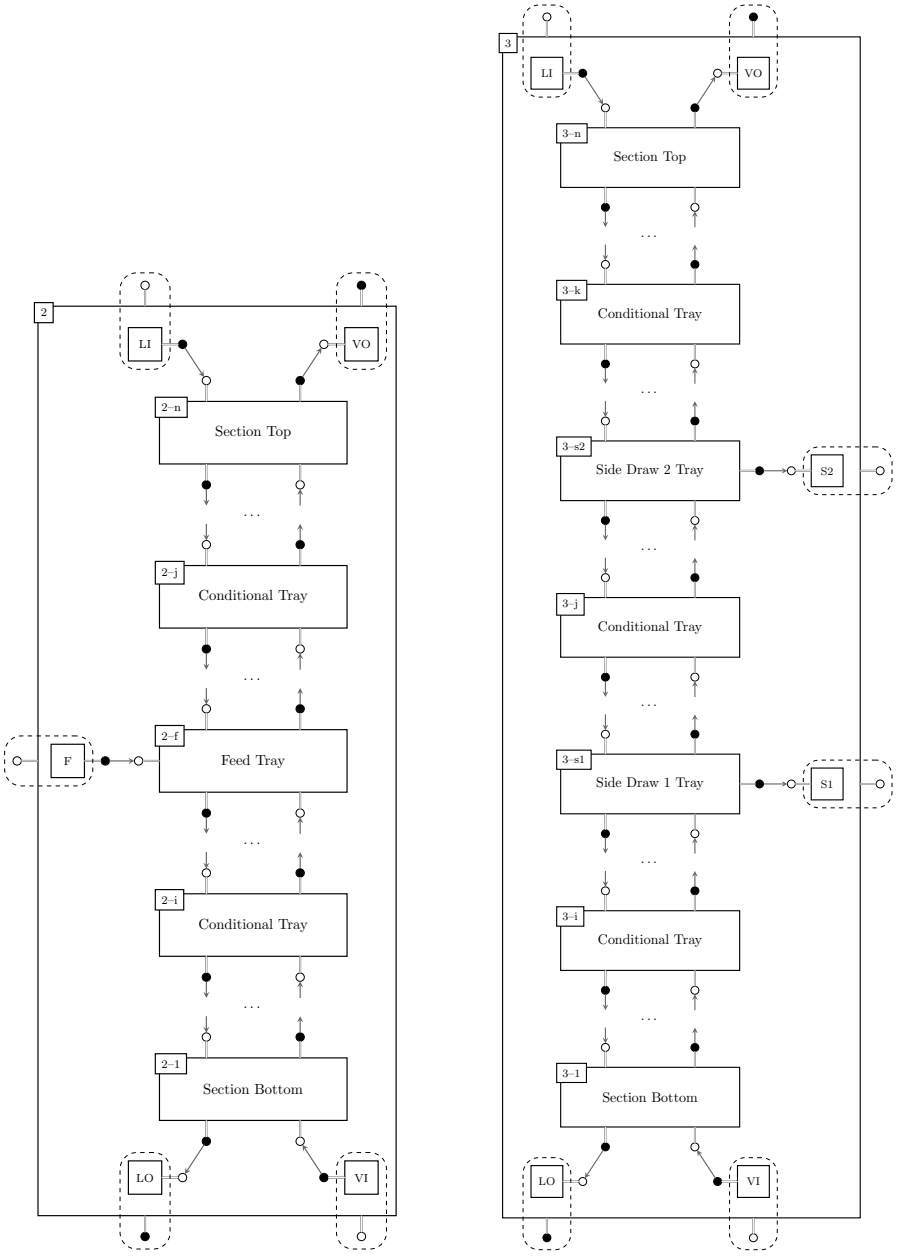
Figure 16: PSG representation for the overall Kaibel column structure.

7.3. Kaibel column

Pyosyn was also used in the conceptual design of a Kaibel column, an intensified dividing wall column involving the high purity separation of four chemical species in a single column (Soraya Rawlings et al., 2019). The PSG superstructure for the Kaibel column is given in Figure 16, highlighting the key material and energy flows to and from the overall system. The column itself is represented by four main units: the stripping section, the feed section, the product section, and the rectification section.

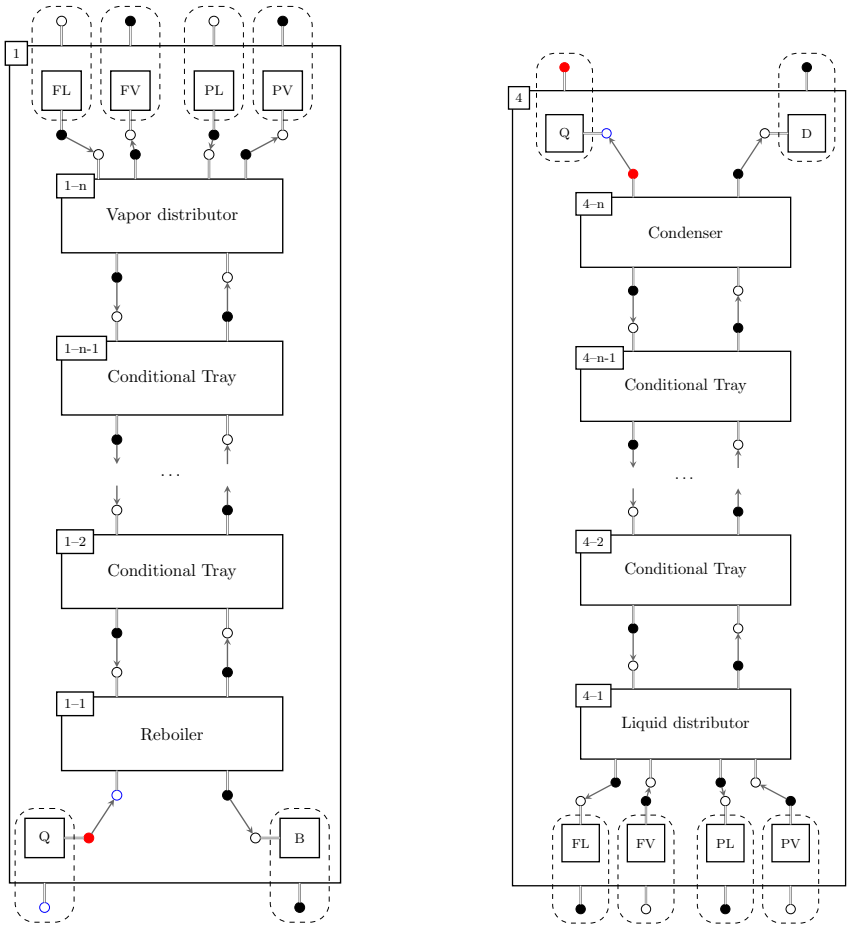
Detailed representations of each section are given in Figures 18a–18b, respectively. The column superstructure includes both fixed trays corresponding to specific processing tasks (e.g. feed) and conditional trays, inspired by the superstructure proposed in (Barttfeld et al., 2003). The conditional trays involve a disjunction between enforcement of mass/heat transfer if the tray exists, or a bypass if the tray is absent. By varying the number of conditional trays that exist between the fixed trays, the column size and feed/side draw locations can be adjusted simultaneously. Details on the identity and function of the fixed trays are covered by Soraya Rawlings et al. (2019).

Based on the PSG representation, a GDP model with 3605 variables (178 Boolean, 3427 continuous) and 5715 constraints was formulated and solved using GDPopt-LOA using a custom initialization routine, as described in (Soraya Rawlings et al., 2019). However, even without the custom initialization,



(a) Feed section. (b) Product section.

Figure 17: PSG representation for the feed and product sections of the Kaibel column.



(a) Stripping section.

(b) Rectification section.

Figure 18: PSG representation for the stripping and rectification sections of the Kaibel column.

GDPopt-LOA (using CPLEX and IPOPT as subsolvers) is able to identify a high-quality feasible solution (after 639 seconds), while MINLP full-space techniques are unable to produce a feasible solution even after an hour. Moreover, due to the openness and flexibility of GDPopt and the Pyosyn infrastructure, custom initialization and other modifications can be easily implemented.

8. Conclusions

In this work, we introduce the Pyosyn framework for systematic process synthesis. Premised upon the need to overcome common barriers to adoption, as well as provide users the support and flexibility to explore different trade-offs in

their problem formulations—including the central trade-off synthesis between generality, fidelity, and tractability—we develop Pyosyn upon principles of intuitive representations, systematic transformations to algebraic models, flexible solution strategies, and an open architecture.

We present the **P**yo**s**yn **G**raph (PSG) representation, a graph-based representation composed of units, unit ports, and streams, including the ability to nest flowsheet elements. We use this nested unit functionality both to support improved representation organization through hierarchical modeling, as well as the concept of "single-choice" units, which aggregate and highlight mutually-exclusive superstructure decision logic between sets of PSG units. We introduce two means of generating the superstructure representation, including both library-assisted and direct hierarchical approaches to the use of means-ends analysis. For the library-assisted approach, we introduce the concept of port annotations to indicate key port compatibility characteristics for screening connectivity. For material ports, we describe annotations for *primary*, *secondary*, and *residual* chemical species, and we use these as the basis for seven connectivity rules to screen candidate streams. We also discuss a potential error-checking extension making use of material port annotations describing feasible flow ranges for chemical species.

We then describe mathematical modeling of PSG representation elements, including automatic simplifications that can be detected and implemented for special cases (such as single-choice units). We construct the model in Pyomo, allowing programmatic model construction supported by the Python high-level programming language. Topology and extensive-variable simplifications are implemented in the open-source, graph-based Pyomo.Network tool. Pyomo.GDP allows for direct expression of superstructure logic and access to a broader suite of automated solution strategies, including advanced logic-based decomposition approaches, while retaining support for conventional MINLP and MILP modeling approaches via reformulation (e.g. BM and HR).

We illustrate the capabilities and flexibility of Pyosyn through a set of case studies that make use of various Pyosyn tools.

We provide Pyosyn to the community as an open-source set of interlinked tools and capabilities, with optional integration to commercial tools. In the future, we plan to continue development of a public synthesis case study library to aid researchers and provide decision-makers with interest in synthesis capabilities an up-to-date view of the state-of-the-art. Pyosyn is not the final word on process synthesis, and opportunities for extensions and refinements remain. In this work, we do not explicitly handle solvents and reactive agents; nor have we explored opportunities to define enhanced screening criteria for energy ports and streams. We plan to continue developing new support and interfaces to incorporate advanced solution strategies into the Pyosyn framework. As an open platform, we view Pyosyn as an excellent complement to more commercially-oriented efforts, bringing exposure to new audiences for systematic synthesis capabilities, and serving as a basis for developing new capabilities.

9. Acknowledgements

We graciously acknowledge funding from the US Department of Energy, Office of Fossil Energy's Crosscutting Research, Simulation-Based Engineering Program through the Institute for the Design of Advanced Energy Systems (IDAES).

References

Agrawal, R., 1996. Synthesis of Distillation Column Configurations for a Multicomponent Separation. Industrial & Engineering Chemistry Research 35, 1059–1071. URL: http://pubs.acs.org/doi/abs/10.1021/ie950323h, doi:10.1021/ie950323h.

Avraamidou, S., Baratsas, S.G., Tian, Y., Pistikopoulos, E.N., 2020. Circular Economy - A challenge and an opportunity for Process Systems Engineering. Computers & Chemical Engineering 133, 106629. URL: https://doi.org/10.1016/j.compchemeng.2019.106629https://linkinghub.elsevier.com/retrieve/pii/S0098135419309913, doi:10.1016/j.compchemeng.2019.106629.

Bagajewicz, M.J., Manousiouthakis, V., 1992. Mass/heat-exchange network representation of distillation networks. AIChE Journal 38, 1769–1800. URL: http://doi.wiley.com/10.1002/aic.690381110, doi:10.1002/aic.690381110.

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

Baldea, M., Edgar, T.F., Stanley, B.L., Kiss, A.A., 2017. Modular manufacturing processes: Status, challenges, and opportunities. AIChE Journal 63, 4262–4272. URL: http://doi.wiley.com/10.1002/aic.15872, doi:10.1002/aic.15872.

Barnicki, S.D., Siirola, J.J., 2004. Process synthesis prospective. Computers & Chemical Engineering 28, 441–446. URL: http://linkinghub.elsevier.com/retrieve/pii/S0098135403002552, doi:10.1016/j.compchemeng.2003.09.030.

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

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

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

Bertran, M.O., Frauzem, R., Sanchez-Arcilla, A.S., Zhang, L., Woodley, J.M., Gani, R., 2017. A generic methodology for processing route synthesis and design based on superstructure optimization. Computers & Chemical Engineering 106, 892–910. URL: http://dx.doi.org/10.1016/j.compchemeng.2017.01.030http://linkinghub.elsevier.com/retrieve/pii/S0098135417300303, doi:10.1016/j.compchemeng.2017.01.030.

Biegler, L.T., Grossmann, I.E., Westerberg, A.W., 1997. Systematic methods of chemical process design. Prentice-Hall international series in the physical and chemical engineering sciences, Prentice Hall PTR.

Boonstra, S., Van Der Blom, K., Hofmeyer, H., Amor, R., Emmerich, M.T., 2016. Super-Structure and Super-Structure Free Design Search Space Representations for a Building Spatial Design in Multi-Disciplinary Building Optimisation. 23rd International Workshop of the European Group for Intelligent Computing in Engineering, EG-ICE 2016 .

Brook, A., Kendrick, D., Meeraus, A., 1988. GAMS, a user's guide. ACM SIGNUM Newsletter 23, 10–11. doi:10.1145/58859.58863.

Bruno, J.C., Fernandez, F., Castells, F., Grossmann, I.E., 1998. A rigorous MINLP model for the optimal synthesis and operation of utility plants. Chemical Engineering Research and Design 76, 246–258. doi:10.1205/026387698524901.

Bussieck, M.R., Drud, A., 2001. SBB: A new solver for mixed integer nonlinear programming. URL: https://old.gams.com/presentations/present_sbb.pdf.

Cafaro, D.C., Grossmann, I.E., 2014. Alternate approximation of concave cost functions for process design and supply chain optimization problems. Computers & Chemical Engineering 60, 376–380. URL: http://dx.doi.org/10.1016/j.compchemeng.2013.10.001http://linkinghub.elsevier.com/retrieve/pii/S0098135413003025, doi:10.1016/j.compchemeng.2013.10.001.

Chen, Q., Bernal, D.E., Johnson, E.S., Kale, S., Bates, J., Siirola, J.D., Grossmann, I.E., 2020. Pyomo.GDP: an ecosystem for logic based modeling and optimization development. In preparation.

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

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

Chen, Q., Johnson, E.S., Siirola, J.D., Grossmann, I.E., 2018. Pyomo.GDP: Disjunctive Models in Python, in: Eden, M.R., Ierapetritou, M.G., Towler, G.P. (Eds.), Proceedings of the 13th International Symposium on Process Systems Engineering. Elsevier B.V., San Diego, pp. 889–894. URL: https://linkinghub.elsevier.com/retrieve/pii/B9780444642417501439, doi:10.1016/B978-0-444-64241-7.50143-9.

Demirel, S.E., Li, J., Hasan, M.M.F., 2017. Systematic Process Intensification using Building Blocks. Computers & Chemical Engineering 105, 2–38. URL: http://dx.doi.org/10.1016/j.compchemeng.2017.01.044, doi:10.1016/j.compchemeng.2017.01.044.

Douglas, J.M., 1985. A hierarchical decision procedure for process synthesis. AIChE Journal 31, 353–362. URL: http://doi.wiley.com/10.1002/aic.690310302, doi:10.1002/aic.690310302.

Douglas, J.M., 1988. Conceptual design of chemical processes. McGraw-Hill, New York.

Duran, M.A., Grossmann, I.E., 1986a. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. Mathematical Programming 36, 307. doi:10.1007/BF02592064.

Duran, M.A., Grossmann, I.E., 1986b. Simultaneous optimization and heat integration of chemical processes. AIChE Journal 32, 123–138. doi:10.1002/aic.690320114.

Ehrgott, M., Wiecek, M.M., 2005. Mutiobjective Programming. Springer New York, New York, NY. pp. 667–708. URL: https://doi.org/10.1007/0-387-23081-5_17, doi:10.1007/0-387-23081-5_17.

Farkas, T., Rev, E., Lelkes, Z., 2005. Process flowsheet superstructures: Structural multiplicity and redundancy. Part I. Basic GDP and MINLP representations. Computers & Chemical Engineering 29, 2180–2197. URL: https://linkinghub.elsevier.com/retrieve/pii/S009813540500181X, doi:10.1016/j.compchemeng.2005.07.007.

Friedler, F., Aviso, K.B., Bertok, B., Foo, D.C., Tan, R.R., 2019. Prospects and challenges for chemical process synthesis with P-graph. Current Opinion

in Chemical Engineering 26, 58–64. URL: https://doi.org/10.1016/j.coche.2019.08.007, doi:10.1016/j.coche.2019.08.007.

Friedler, F., Tarján, K., Huang, Y., Fan, L., 1992. Graph-theoretic approach to process synthesis: axioms and theorems. Chemical Engineering Science 47, 1973–1988. URL: http://www.sciencedirect.com/science/article/pii/0009250992803154, doi:10.1016/0009-2509(92)80315-4.

Friedler, F., Tarjan, K., Huang, Y., Fan, L., 1993. Graph-theoretic approach to process synthesis: Polynomial algorithm for maximal structure generation. Computers & Chemical Engineering 17, 929–942. doi:10.1016/0098-1354(93)80074-W.

Friedman, Z., Ingalls, J., Siirola, J.D., Watson, J.P., 2013. Block-oriented modeling of superstructure optimization problems. Computers and Chemical Engineering 57, 10–23. URL: http://dx.doi.org/10.1016/j.compchemeng.2013.04.008, doi:10.1016/j.compchemeng.2013.04.008.

Gani, R., Hytoft, G., Jaksland, C., Jensen, A.K., 1997. An integrated computer aided system for integrated design of chemical processes. Computers & Chemical Engineering 21, 1135–1146. URL: http://www.sciencedirect.com/science/article/pii/S0098135496003249, doi:10.1016/S0098-1354(96)00324-9.

Glasser, D., Crowe, C.M., Hildebrandt, D., 1987. A geometric approach to steady flow reactors: the attainable region and optimization in concentration space. Industrial & Engineering Chemistry Research 26, 1803–1810. URL: http://pubs.acs.org/doi/abs/10.1021/ie00069a014, doi:10.1021/ie00069a014.

Grossmann, I.E., 2014. Challenges in the application of mathematical programming in the enterprise-wide optimization of process industries. Theoretical Foundations of Chemical Engineering 48, 555–573. URL: http://link.springer.com/10.1134/S0040579514050182, doi:10.1134/S0040579514050182.

Grossmann, I.E., Harjunkoski, I., 2019. Process systems Engineering: Academic and industrial perspectives. Computers and Chemical Engineering 126, 474–484. doi:10.1016/j.compchemeng.2019.04.028.

Grossmann, I.E., Lee, S., 2003. Generalized Convex Disjunctive Programming: Nonlinear Convex Hull Relaxation. Computational Optimization and Applications 26, 83–100. URL: http://link.springer.com/10.1023/A:1025154322278, doi:10.1023/A:1025154322278.

Grossmann, I.E., Trespalacios, F., 2013. Systematic modeling of discrete-continuous optimization models through generalized disjunctive programming. AIChE Journal 59, 3276–3295. URL: http://doi.wiley.com/10.1002/aic.14088, doi:10.1002/aic.14088.

Hagberg, A.A., Schult, D.A., Swart, P.J., 2008. Exploring network structure, dynamics, and function using networkx, in: Varoquaux, G., Vaught, T., Millman, J. (Eds.), Proceedings of the 7th Python in Science Conference, Pasadena, CA USA. pp. 11 – 15.

Hart, W.E., Laird, C.D., Watson, J.P., Woodruff, D.L., Hackebeil, G.A., Nicholson, B.L., Siirola, J.D., 2017. Pyomo — Optimization Modeling in Python. volume 67 of *Springer Optimization and Its Applications*. 2nd ed., Springer International Publishing, Cham. URL: http://link.springer.com/10.1007/978-3-319-58821-6, doi:10.1007/978-3-319-58821-6.

Hohman, E.C., 1971. Optimum networks for heat exchange. Phd dissertation. University of Southern California.

Horn, F., 1964. Attainable and non-attainable regions in chemical reaction technique, in: Proceedings of the European Symposium on Chemical Reaction Engineering, Pergamon Press Ltd, London.

Kelly, J.D., 2004. Production Modeling for Multimodal Operations. Chemical Engineering Progress 100, 44–46.

Kocis, G., Grossmann, I., 1989. A modelling and decomposition strategy for the minlp optimization of process flowsheets. Computers & Chemical Engineering 13, 797–819. doi:10.1016/0098-1354(89)85053-7.

Kondili, E., Pantelides, C., Sargent, R., 1993. A general algorithm for short-term scheduling of batch operations—I. MILP formulation. Computers & Chemical Engineering 17, 211–227. URL: http://linkinghub.elsevier.com/retrieve/pii/009813549380015F, doi:10.1016/0098-1354(93)80015-F.

Kravanja, Z., Grossmann, I.E., 1990. Prosyn-an MINLP process synthesizer. Computers and Chemical Engineering 14, 1363–1378. URL: http://www.scopus.com/inward/record.url?eid=2-s2.0-0025623478{&}partnerID=40{&}md5=abad0df253d0a61f611af80fa89c6193.

Kronqvist, J., Bernal, D.E., Lundell, A., Grossmann, I.E., 2019. A review and comparison of solvers for convex MINLP. Optimization and Engineering 20, 397–455. URL: https://doi.org/10.1007/s11081-018-9411-8, doi:10.1007/s11081-018-9411-8.

Kuhlmann, H., Skiborowski, M., 2016. Synthesis of Intensified Processes from a Superstructure of Phenomena Building Blocks, in: Proceedings of the 26th European Symposium on Computer Aided Process Engineering, pp. 697–702. URL: http://linkinghub.elsevier.com/retrieve/pii/B9780444634283501211, doi:10.1016/B978-0-444-63428-3.50121-1.

Lang, Y.D., Biegler, L., Grossmann, I., 1988. Simultaneous optimization and heat integration with process simulators. Computers & Chemical Engineering 12, 311–327. URL: http://www.sciencedirect.com/science/article/pii/0098135488850440, doi:10.1016/0098-1354(88)85044-0.

Lee, A., Ghouse, J.H., Chen, Q., Eslick, J.C., Siirola, J.D., Grossman, I.E., Miller, D.C., 2018. A Flexible Framework and Model Library for Process Simulation, Optimization and Control, in: Proceedings of the 13th International Symposium on Process Systems Engineering, pp. 937–942. URL: https://linkinghub.elsevier.com/retrieve/pii/B9780444642417501518, doi:10.1016/B978-0-444-64241-7.50151-8.

Lee, S., Grossmann, I.E., 2000. New algorithms for nonlinear generalized disjunctive programming. Computers & Chemical Engineering 24, 2125–2141. URL: http://www.sciencedirect.com/science/article/pii/S0098135400005810, doi:10.1016/S0098-1354(00)00581-0.

Lee, S., Grossmann, I.E., 2001. A global optimization algorithm for nonconvex generalized disjunctive programming and applications to process systems. Computers & Chemical Engineering 25, 1675–1697. URL: http://linkinghub.elsevier.com/retrieve/pii/S0098135401007323, doi:10.1016/S0098-1354(01)00732-3.

Linnhoff, B., Hindmarsh, E., 1983. The pinch design method for heat exchanger networks. Chemical Engineering Science 38, 745–763. URL: http://www.sciencedirect.com/science/article/pii/0009250983801857, doi:https://doi.org/10.1016/0009-2509(83)80185-7.

Lutze, P., Babi, D.K., Woodley, J.M., Gani, R., 2013. Phenomena Based Methodology for Process Synthesis Incorporating Process Intensification. Industrial & Engineering Chemistry Research 52, 7127–7144. URL: http://pubs.acs.org/doi/abs/10.1021/ie302513y, doi:10.1021/ie302513y.

Marler, R.T., Arora, J.S., 2004. Survey of multi-objective optimization methods for engineering. Structural and Multidisciplinary Optimization 26, 369–395. doi:10.1007/s00158-003-0368-6.

Martín, M., Adams II, T.A., 2019. Challenges and future directions for process and product synthesis and design. Computers & Chemical Engineering 128, 421–436. URL: https://linkinghub.elsevier.com/retrieve/pii/S0098135418312997, doi:10.1016/j.compchemeng.2019.06.022.

Mencarelli, L., Chen, Q., Pagot, A., Grossmann, I.E., 2020. A review on superstructure optimization approaches in process system engineering. Computers & Chemical Engineering , 106808URL: https://linkinghub.elsevier.com/retrieve/pii/S0098135419313924, doi:10.1016/j.compchemeng.2020.106808.

Miller, D.C., Agarwal, D., Siirola, J., 2016. Institute for the Design of Advanced Energy Systems (IDAES) Proposal.

Miller, D.C., Siirola, J.D., Agarwal, D., Burgard, A.P., Lee, A., Eslick, J.C., Nicholson, B., Laird, C., Biegler, L.T., Bhattacharyya, D., Sahinidis, N.V.,

Grossmann, I.E., Gounaris, C.E., Gunter, D., 2018. Next Generation Multi-Scale Process Systems Engineering Framework. Computer Aided Chemical Engineering 44, 2209–2214. doi:10.1016/B978-0-444-64241-7.50363-3.

Misener, R., Floudas, C.A., 2014. ANTIGONE: Algorithms for coNTinuous / Integer Global Optimization of Nonlinear Equations. Journal of Global Optimization 59, 503–526. doi:10.1007/s10898-014-0166-2.

Mitsos, A., Asprion, N., Floudas, C.A., Bortz, M., Baldea, M., Bonvin, D., Caspari, A., Schäfer, P., 2018. Challenges in process optimization for new feedstocks and energy sources. Computers and Chemical Engineering 113, 209–221. URL: https://doi.org/10.1016/j.compchemeng.2018.03.013, doi:10.1016/j.compchemeng.2018.03.013.

Morbach, J., Yang, A., Marquardt, W., 2007. OntoCAPE-A large-scale ontology for chemical process engineering. Engineering Applications of Artificial Intelligence 20, 147–161. doi:10.1016/j.engappai.2006.06.010.

Muts, P., Nowak, I., Hendrix, E.M.T., 2020. The decomposition-based outer approximation algorithm for convex mixed-integer nonlinear programming. Journal of Global Optimization , 1–22URL: http://link.springer.com/10.1007/s10898-020-00888-x, doi:10.1007/s10898-020-00888-x.

Nemhauser, G.L., Wolsey, L.A., 1988. Integer and combinatorial optimization. Wiley, New York.

Neveux, T., 2018. Ab-initio process synthesis using evolutionary programming. Chemical Engineering Science 185, 209–221. URL: http://linkinghub.elsevier.com/retrieve/pii/S0009250918302136, doi:10.1016/j.ces.2018.04.015.

Newell, A., Simon, H.A., 1961. Computer Simulation of Human Thinking. Science 134, 2011–2017. URL: http://www.jstor.org/stable/1708146.

Nishida, N., Stephanopoulos, G., Westerberg, A.W., 1981. A review of process synthesis. AIChE Journal 27, 321–351. doi:10.1002/aic.690270302.

Papalexandri, K.P., Pistikopoulos, E.N., 1994. Synthesis and Retrofit Design of Operable Heat Exchanger Networks. 1. Flexibility and Structural Controllability Aspects. Industrial & Engineering Chemistry Research 33, 1718–1737. URL: http://pubs.acs.org/doi/abs/10.1021/ie00031a012, doi:10.1021/ie00031a012.

Puranik, Y., Sahinidis, N.V., 2017. Domain reduction techniques for global NLP and MINLP optimization. Constraints 22, 338–376. doi:10.1007/s10601-016-9267-5, arXiv:1706.08601.

Quesada, I., Grossmann, I.E., 1995. Global optimization of bilinear process networks with multicomponent flows. Computers and Chemical Engineering 19, 1219–1242. doi:10.1016/0098-1354(94)00123-5.

Raman, R., Grossmann, I., 1994. Modelling and computational techniques for logic based integer programming. Computers & Chemical Engineering 18, 563–578. doi:10.1016/0098-1354(93)E0010-7.

Ruiz, J.P., Grossmann, I.E., 2012. A hierarchy of relaxations for nonlinear convex generalized disjunctive programming. European Journal of Operational Research 218, 38–47. doi:10.1016/j.ejor.2011.10.002.

Sargent, R., Gaminibandara, K., 1976. Optimum Design of Plate Distillation Columns, in: Dixon, L. (Ed.), Optimization in Action. Academic Press, London, pp. 267–314.

Schembecker, G., Simmrock, K.H., Wolff, A., 1994. Synthesis of chemical process flowsheets by means of cooperating knowledge integrating systems, in: Institution of Chemical Engineers Symposium Series.

Siirola, J.J., Powers, G.J., Rudd, D.F., 1971. Synthesis of system designs: III. Toward a process concept generator. AIChE Journal 17, 677–682. URL: http://doi.wiley.com/10.1002/aic.690170334, doi:10.1002/aic.690170334.

Siirola, J.J., Rudd, D.F., 1971. Computer-Aided Synthesis of Chemical Process Designs. From Reaction Path Data to the Process Task Network. Industrial & Engineering Chemistry Fundamentals 10, 353–362. URL: http://dx.doi.org/10.1021/i160039a003http://pubs.acs.org/doi/abs/10.1021/i160039a003, doi:10.1021/i160039a003.

Sitter, S., Chen, Q., Grossmann, I.E., 2019. An overview of process intensification methods. Current Opinion in Chemical Engineering URL: https://linkinghub.elsevier.com/retrieve/pii/S2211339818300601, doi:10.1016/j.coche.2018.12.006.

Smith, E., Pantelides, C., 1995. Design of reaction/separation networks using detailed models. Computers & Chemical Engineering 19, 83–88. URL: http://linkinghub.elsevier.com/retrieve/pii/0098135495870199, doi:10.1016/0098-1354(95)87019-9.

Soraya Rawlings, E., Chen, Q., Grossmann, I.E., Caballero, J.A., 2019. Kaibel column: Modeling, optimization, and conceptual design of multiproduct dividing wall columns. Computers & Chemical Engineering 125, 31–39. URL: https://linkinghub.elsevier.com/retrieve/pii/S0098135419300250, doi:10.1016/j.compchemeng.2019.03.006.

Stankiewicz, A., 2018. Introduction to Process Intensification Principles and Approaches: Structure, Energy, Synergy and Time. URL: https://www.aiche.org/academy/webinars/intro-pi-principles-and-approaches-structure-energy-synergy-and-time.

Stephanopoulos, G., Westerberg, A.W., 1976. Studies in process synthesis—II. Chemical Engineering Science 31, 195–204. doi:10.1016/0009-2509(76)85057-9.

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

Tian, Y., Demirel, S.E., Hasan, M.M., Pistikopoulos, E.N., 2018a. An overview of process systems engineering approaches for process intensification: State of the art. Chemical Engineering and Processing - Process Intensification 133, 160–210. URL: https://doi.org/10.1016/j.cep.2018.07.014, doi:10.1016/j.cep.2018.07.014.

Tian, Y., Sam Mannan, M., Kravanja, Z., Pistikopoulos, E., 2018b. Towards the synthesis of modular process intensification systems with safety and operability considerations - application to heat exchanger network. Computer Aided Chemical Engineering 43, 0–5. doi:10.1016/B978-0-444-64235-6.50125-X.

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

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

Tsay, C., Pattison, R.C., Piana, M.R., Baldea, M., 2018. A survey of optimal process design capabilities and practices in the chemical and petrochemical industries. Computers and Chemical Engineering 112, 180–189. URL: https://doi.org/10.1016/j.compchemeng.2018.01.012, doi:10.1016/j.compchemeng.2018.01.012.

Tula, A.K., Eden, M.R., Gani, R., 2019a. Computer-aided process intensification: Challenges, trends and opportunities. AIChE Journal URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/aic.16819, doi:10.1002/aic.16819.

Tula, A.K., Eden, M.R., Gani, R., 2019b. Hybrid method and associated tools for synthesis of sustainable process flowsheets. Computers and Chemical Engineering 131. doi:10.1016/j.compchemeng.2019.106572.

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

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

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

Westerberg, A.W., 2004. A retrospective on design and process synthesis. Computers & Chemical Engineering 28, 447–458. URL: http://www.sciencedirect.com/science/article/pii/S0098135403002564, doi:http://dx.doi.org/10.1016/j.compchemeng.2003.09.029.

Wu, W., Henao, C.A., Maravelias, C.T., 2016. A superstructure representation, generation, and modeling framework for chemical process synthesis. AIChE Journal 62, 3199–3214. URL: http://doi.wiley.com/10.1002/aic.15300, doi:10.1002/aic.15300.

Yee, T., Grossmann, I., 1990. Simultaneous optimization models for heat integration—II. Heat exchanger network synthesis. Computers & Chemical Engineering 14, 1165–1184. URL: http://linkinghub.elsevier.com/retrieve/pii/0098135490850108, doi:10.1016/0098-1354(90)85010-8.

Yeomans, H., Grossmann, I.E., 1999. A systematic modeling framework of superstructure optimization in process synthesis. Computers & Chemical Engineering 23, 709–731. URL: http://www.sciencedirect.com/science/article/pii/S0098135499000034, doi:10.1016/S0098-1354(99)00003-4.

Appendix A. Zero-flow singularities

We demonstrate the "zero flow" numerical difficulty with two small examples relevant to process synthesis.

First, we consider a mole fraction calculation that may be required to support physical property computations for a process unit. Consider a three-component system with species molar flows f_1, f_2, f_3 into the unit, total molar flow F, and species molar fraction x_1. The mole fraction computation for x_1 may be expressed as in Equation (A.1).

$$x_1 = \frac{f_1}{F} = \frac{f_1}{f_1 + f_2 + f_3} \tag{A.1}$$

When the unit is absent from the flowsheet, its corresponding flows are set to zero: $f_1 = f_2 = f_3 = F = 0$. This leads Equation (A.1), and thus x_1, to become the undefined quantity $0/0$. A common workaround is to impose a small minimum value for F, denoted $\underline{F}$. Consider the case where $x_1 = 0.75, x_2 = 0, x_3 = 0.25$, and the minimum value $F = \underline{F} = 0.001$. Equation (A.1) now yields the correct quantity (subject to computational floating-point tolerances), giving $x_1 = 0.75$. However, note that the first derivative $\frac{\partial x_1}{\partial f_1}$, given by Equation (A.2), is equal to 250.

$$\frac{\partial x_1}{\partial f_1} = \frac{f_2 + f_3}{(f_1 + f_2 + f_3)^2} = \frac{0.00025}{(0.001)^2} = 250 \tag{A.2}$$

And its second derivative $\frac{\partial^2 x_1}{\partial f_1^2}$, given by Equation (A.3), is equal to $-500,000$.

$$\frac{\partial^2 x_1}{\partial f_1^2} = -\frac{2(f_2+f_3)}{(f_1+f_2+f_3)^3} = -\frac{0.0005}{(0.001)^3} = -500,000 \tag{A.3}$$

These derivatives are orders of magnitude higher than the expected range for a molar fraction. This can result in poor scaling within modern nonlinear programming solvers, which rely on first- (and sometimes second-) order derivative information to guide the direction and size of their iterates, degrading their performance and robustness. Note that decreasing the value of $\underline{F}$ to preserve greater model accuracy only exacerbates this numerical stability problem. Furthermore, though skilled modelers often express Equation (A.1) as $Fx_1 = f_1$ to avoid division-by-zero, this does not alleviate poor derivative scaling.

A second common source of zero-flow issues arises from the use of power-law capital cost scaling for processes and process equipment. This empirical scaling rule, given in Equation (A.4), is a staple of process design (Biegler et al., 1997).

$$C = C_0 \left(\frac{S}{S_0}\right)^{\gamma} \tag{A.4}$$

Here, the cost C of a process of size S is estimated in relation to a base process of size S_0, with cost C_0. The scaling factor γ often takes a value around 0.67. At zero flow, the size is zero, $S = 0$, and therefore, by Equation (A.4), the cost is also zero, $C = 0$. However, note that the derivative of the power law cost function, given in Equation (A.5), becomes undefined at $S = 0$, due to the fractional exponent $\gamma < 1$.

$$\frac{\partial C}{\partial S} = \frac{C_0 \gamma}{S}\left(\frac{S}{S_0}\right)^{\gamma} \tag{A.5}$$

Again, this "zero flow" issue would result in degraded solver performance or even solver failure. A common workaround is to add a small value ε to the base of the exponent, giving the approximation in Equation (A.6).

$$C = C_0 \left(\frac{S}{S_0} + \varepsilon\right)^{\gamma} \tag{A.6}$$

Cafaro and Grossmann (2014) give an alternative approximation approach. However, these approximations may affect solution accuracy. Note that physical property calculations involving fractional exponents would be similarly susceptible. Effective solution strategies must therefore be robust to "zero flow" singularities, when they are relevant.