PROCEEDINGS OF THE 31st European Symposium on Computer Aided Process Engineering (ESCAPE31), June 6-9, 2021, İstanbul, Turkey
M. Türkay, R. Gani (Editors).
http://dx.doi.org/10.1016/B978-0-323-88506-5.50065-6

OUTDOOR – An open-source superstructure construction and optimization tool

Philipp Kenkel[a*], Timo Wassermann[a], Celina Rose[a], Edwin Zondervan[b]

[a]*Advanced Energy Systems Institute, Enrique-Schmidt Straße 7, 28359 Bremen, Germany*

[b]*Department of Chemical Engineering, Twente University, 7522NB Enschede, Netherlands*

kenkel@uni-bremen.de

Abstract

To enhance the availability of superstructure optimization as a tool in process design for industries as well as science this work presents a fully ***O**pen s**U**perstruc**T**ure mo**D**eling and **O**ptimizati**O**n f**R**amework* (OUTDOOR). This framework is written in Python using object-oriented programming in combination with algebraic modeling utilizing the PYOMO modeling language. In addition, an Excel-based data preparation tool, called Excel-Wrapper is presented. It provides an intuitive way to prepare process data and generate superstructures, which are ready-made for solving using open-source as well as commercial optimization solvers.

Keywords: Superstructure Optimization, Python, Open Source

1. Introduction

Superstructure optimization is a powerful tool to perform preliminary process design. Superstructure models map all possible flowsheets to produce a product from a specified set of raw materials. Such models are defined as mixed integer (non)-linear mathematical programming (MI(N)LP) models and can be solved for different objective functions, e.g. minimal production costs using open-source and commercial optimization solvers. Superstructure models have been applied to different areas, from biomass-based economy up to power-to-x processes (Galanopoulos et al., 2019; Kenkel et al., 2020) . However, they can be formulated in different ways leading to unnecessary complexity and ambiguity. The common practice is that research groups develop their own models and tools, often using commercial software, e.g. GAMS. Only a few software solutions for generic superstructure optimization have been presented, none of them being fully open source (Mencarelli et al., 2020). In addition, the application of superstructure models in industry is limited by its complicated algebraic formulation and data provision. Generally, straight forward graphical solutions are preferred over complex mathematical models and extensive programming codes.

This work proposes an approach to solve both problems, presenting an ***O**pen s**U**perstruc**T**ure mo**D**eling and **O**ptimizati**O**n f**R**amework* (OUTDOOR). This framework is setup in Python using a combination of modular object-oriented programming and

algebraic model formulation in PYOMO. Its focus is to construct deterministic MILP models to solve superstructure design problems.

2. Programming and modeling fundamentals

The presented framework is a tool for intuitive construction of superstructure models. It utilizes different concepts such as superstructure optimization modeling, object-oriented programming as well as the open-source optimization modeling language (PYOMO). The combined effort leading to OUTDOOR is briefly discussed in the following sections.

2.1. Superstructure optimization and algebraic modeling

Superstructure optimization is a methodology which is used for process design. It is based on mathematical models derived from mass- and energy balances in combination with cost calculation of capital expenditures and operational costs to optimize process flow sheets for different objective functions (Quaglia et al., 2015)]. These models are formulated as depicted in Eq. (1). Here $\boldsymbol{c}$ and $\boldsymbol{b}$ are vectors of known constant parameters, while $\boldsymbol{x}$ is a vector or variables (continuous as well as integer), and A is matrix of known parameters.

$$\begin{aligned} &maximize\ \boldsymbol{c^T x} \\ &s.t\ A\boldsymbol{x} = \boldsymbol{b} \\ &\boldsymbol{x} \geq 0 \end{aligned} \tag{1}$$

Recurring equations are often formulated using indexed variables and parameters. For example, the electricity demand E_u^{EL} of a unit operation u is dependent of the total inlet flow F_u^{IN} and a specific factor p_u (kWh/t_{Input}). If this holds for each unit operation u in a set of U, the equations can be written as shown in Eq. (2).

$$E_u^{EL} = p_u \cdot F_u^{IN} \qquad with\ u \in U \tag{2}$$

The fixed parameters of p_u are then supplied to the model as an indexed list of a set U inside of a data file.

2.2. Object-oriented programming

Object-oriented programming, in contrast to procedural programming, uses classes and objects as containers to allocate attributes and methods. These objects can be used to store data related to unit operations (such as the presented specific electricity demand p_u) directly in the process unit object. This way process-data can be built up around the process classes and objects, presenting a more intuitive way to handle and store data compared to indexed lists.

2.3. PYOMO Modeling language

PYOMO is a Python-based, open-source optimization language with a diverse set of capabilities to solve algebraic optimization models, while providing access to optimization solvers such as CPLEX or IPOPT. (Hart et al., 2017) The user benefits from PYOMO implemented direct access to solver configurations and solver output regarding calculation procedure, infeasibilities or other occurrences.

3. OUTDOOR

OUTDOOR combines the concepts that were discussed in the earlier sections. It provides an intuitive workflow for superstructure model creation and solving. Figure 1 depicts the main concepts and workflow of the application of OUTDOOR, and will be discussed in the following sections.

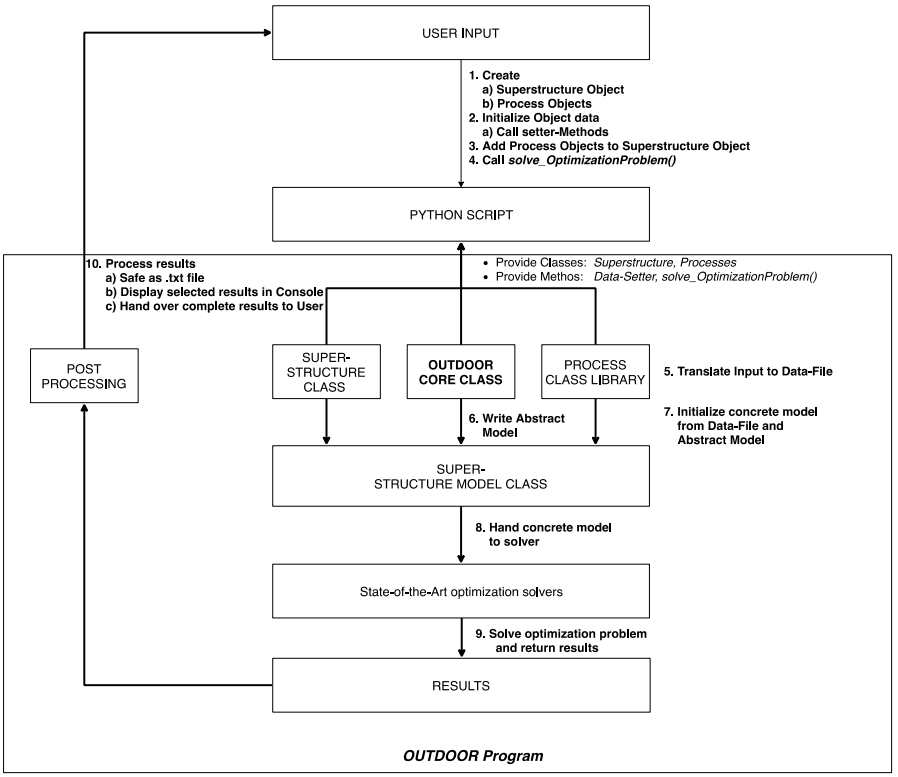

Figure 1: Visualization of the OUTDOOR workflow.

3.1. User input

As a first step user input is required. This input of data can be divided into two categories. The first category is the superordinate system data. An object of the provided *Superstructure-Class* is created and initialized using Python scripting. This object demands important attributes such as the costs for electricity, the occurring chemical compounds or reactions or the desired product and objective. The second category is the input of considered unit operations. Different objects of the *Process-Classes* such as stoichiometric reactors or stream splitters are created and initialized with their respective data such as the required electricity demand or stoichiometric coefficients. The initialization of data of the different objects (step 2) is performed by calling predefined setter-methods, which are provided by the Classes.
Next, (step 3) the process objects are added to the superstructure creating a complete superstructure system with all unit operations and external parameters such as electricity

prices. An input of initial guesses is not required at this state, however it could be necessary in the future to investigate more complex problems. When all inputs are provided, the user calls the *solve_OptimizationProblem()* method from the OUTDOOR module (step 4) to start the OUTDOOR program and solve the optimization problem.

3.2. OUTDOOR program

OUTDOOR processes the inputs provided by the user in a series of steps. First (step 5) the data in the process and superstructure objects is translated to a Python dictionary. Generally, two different types can be generated. The first one comprises easy convertible parameters and the second one comprises complex convertible parameters.

The specific electricity demand p_u (kWh/t_{Input}) is an illustration of a parameter of the first type. It stores only a numeric value which has to be converted into the right format while being assigned to the right unit operation.

Other relationships however, demand more complex translation methods. One example is the allocation of the referred reference flow of the electricity demand calculation. This flow can be the total inlet flow as depicted in Eq. (2). It could also be the outlet flow, or even just specific components of one of these flows. Hence, a method which translates correlations like *"outlet flow"* or *"inlet flow"* into algebraic understandable parameters is required. The procedure of these methods will be explained exemplary for this electricity demand reference flow in the following. The program needs two different information. The delivery of these information is still conducted during the input-phase of the user. One of the already mentioned setter-methods of the process class asks if "FIN" or "FOUT" has to be considered. Another setter-method demands the input of a Python list of components or the string "all components".

During the calculation, the program calls a translation method which transforms the input of the user into process specific $\kappa^1_{EL,u,i}$ and $\kappa^2_{EL,u}$ parameters. The first one is a binary parameter being 1 if the component *i* is to be considered for electricity in unit operation *u*. The second parameter determines if the reference flow is the inlet (Value = 1) or the outlet flow (Value = 0). These parameters are integrated into the algebraic model using $\kappa^1_{EL,u,i}$ as equation parameter and $\kappa^2_{EL,u}$ as a conditional factor in construction of the constraint (cf. Figure 2).

```
def ElectricityBalance_1_rule(self,u):
    if self.kappa_2_ut[u,'Electricity'] == 1:
      return  self.REF_FLOW_EL[u] == sum(self.FLOW_IN[u,i] \
                        * self.kappa_1_ut[u,'Electricity',i] for i in self.I)
    elif self.kappa_2_ut[u,'Electricity'] == 0:
      return self.REF_FLOW_EL[u] == sum(self.FLOW_OUT[u,i] \
                        * self.kappa_1_ut[u,'Electricity',i] for i in self.I)
    else:
      return self.REF_FLOW_EL[u] == 0
```

Figure 2: Python code example for calculation of references flow in electricity consumption

Subsequent to the translation of input into the right format, the program creates the generic equations of the model (step 6). This model is written as an object of the PYOMO *AbstractModel Class*, which means it contains empty parameters which can be filled at a later stage using a data file.

Step 7 of the program uses the created Python dictionary (data file) and the abstract model to create a concrete model, depicting the algebraic formulation of the defined superstructure. This model is then handed over to the selected optimization solver using the capabilities of the PYOMO modeling language (Step 8) and optimized for the defined objective function (Step 9)

3.3. User output

After the solver found a solution to the optimization problem, the complete raw results are returned to the user. It is possible to check the solutions for the variables and constraints utilizing the Python console. However, to simplify the results access, the OUTDOOR program also processes them (step 10). This step includes the presentation of selected results in the Python console as well as the saving of the main results as a .txt file into a chosen folder. The main results include the net production costs and CO_2 emissions, the cost breakdown and the energy consumption details including heat integration details.

3.4. Extensions and development

OUTDOOR depends on the user input data in form of a Python script using the definition of objects as well as calling setter-methods and the *solve_OptimizationProblem()* -method. This is however, not really user friendly for non-programmers. Therefore, two extensions are under development. The first module is an *Excel-Wrapper*. This tool enables user to insert data via a predefined Microsoft Excel template, Afterwards the user has to start the *Excel-Wrapper* tool, which itself creates the objects, calls the solve-function and saves the results. The second module presents a graphical user interface in which processes can be created by drag and drop, and data can be provided by popup tables. Other features of interest which are under development include more detailed and customized error handling and output, as well as capabilities for multi-criteria decision making and robust decision making taking stochastic programming into account.

4. An example: Power-to-methanol process

The goal of this case study is to investigate renewable methanol production with three different CO_2 sources as well as five hydrogen supply technologies and two offgas utilization methods. Two objective functions were considered in the superstructure optimization: i) Minimal net production costs and ii) Minimal net production CO2 emissions. Detailed outcomes of the study are presented in Kenkel et al. (2020).
To construct and solve the superstructure model, OUTDOOR is applied in combination with the Excel-Wrapper. All relevant data was entered into the predefined Excel sheet provided by OUTDOOR. Data on considered unit operations was implemented in single Excel sheets. Figure 3 depicts an excerpt of this user input in form of the specific energy demand, reference flow and component for one unit operation. Afterwards, OUTDOOR automatically translated the input into a readable data file, formulated the general model including mass- and energy balances as well as cost and emission functions. The generated model is automatically handed to the Gurobi solver and optimization results in terms of chosen technologies, net present costs and emissions, as well as costs and energy break-down are presented in the Python console as well as in a .txt. file. The cpu time of the data collection using the excel-wrapper is 0.016 seconds. The construction of the superstructure objects takes 2.3 seconds, resulting in a MILP with 29411 variables (5746 binary) and 43581 constraints. The cpu time for solving this problem and returning the results using Gurobi is 11.1 seconds.

Energy Related Factors			
Index	**Electricity**	**Cooling**	**Heating**
Specific Energy demand (MWh/t)	54,3	-20,97	
Reference Flow	FOUT	FOUT	
Considered Component	H2	H2	
Considered Component			
Considered Component			
Considered Component			

Figure 3: Excerpt of Excel-based parameter input for OUTDOOR

5. Conclusions

The main programming concepts and workflow of the ***O**pen s**U**perstruc**T**ure mo**D**eling and **O**ptimizati**O**n f**R**amework* (OUTDOOR) were presented in this work. This framework enables the initialization of data as unit operation related attributes using object-oriented programming, while still writing the actual model as algebraic model utilizing the PYOMO optimization language. To connect both of these concepts, different methods in terms of so-called translation functions were developed. To further strengthen the applicability of the tool an Excel-Wrapper was developed which uses data stored in predefined Excel-Templates as well as the Class-specific methods to create the superstructure, solve it and display and save the results. First results were derived using a beta-version of the framework. These results were presented in Kenkel et al. (2020).

Funding

Funding of this research by the German Federal Ministry of Economic Affairs and Energy within the KEROSyN100 project (funding code 03EIV051A) is gratefully acknowledged.

References

Galanopoulos, C., Kenkel, P., Zondervan, E., 2019. Superstructure optimization of an integrated algae biorefinery 130. https://doi.org/10.1016/j.compchemeng.2019.106530

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, Springer Optimization and Its Applications. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-319-58821-6

Kenkel, P., Wassermann, T., Zondervan, E., 2020. Design of a Sustainable Power-to-methanol Process: a Superstructure Approach Integrated with Heat Exchanger Network Optimization. Comput. Aided Chem. Eng. 48, 1411–1416.

Mencarelli, L., Chen, Q., Pagot, A., Chemical, I.G.-C.&, 2020, U., 2020. A review on superstructure optimization approaches in process system engineering. Elsevier.

Quaglia, A., Gargalo, C.L., Chairakwongsa, S., Sin, G., Gani, R., 2015. Systematic network synthesis and design : Problem formulation , superstructure generation , data management and solution. Comput. Chem. Eng. 72, 68–86. https://doi.org/10.1016/j.compchemeng.2014.03.007