Biobased Chemistry and Technology

Thesis Biobased Chemistry and Technology

A framework for formulating and optimising a superstructure

Tim Hoogstad

June 2015

WAGENINGEN UNIVERSITY
AGROTECHNOLOGY AND
FOOD SCIENCES

WAGENINGEN UR

A framework for formulating and optimising a superstructure

Name course	:	Thesis project Biobased Chemistry and Technology
Number	:	BCT-80424
Study load	:	24 ects
Date	:	June 2015

Student	:	Tim Hoogstad
Registration number	:	910910-361-010
Study programme	:	MBT (Biotechnology)
Report number	:	023BCT

Supervisor(s)	:	Ellen Slegers
Examiners	:	Ton van Boxtel
Group	:	Biobased Chemistry and Technology
Address	:	Bornse Weilanden 9 6708 WG Wageningen the Netherlands

WAGENINGEN UNIVERSITY
AGROTECHNOLOGY AND
FOOD SCIENCES

Contents

Introduction 6
Methods 10
Mathematical problem formulation 10
Non-linear Programming problems 10
Mixed Integer Non-linear Programming problems 11
NLP vs. MINLP 11
Optimisation tools 12
Genetic algorithm 12
Branch and bound 13
Interior point method 13
Dynamic programming 14
Route analysis 14
Conceptual superstructure 15
Input format 16
Results and Discussion 17
The Framework 17
Comparing the framework with other methods 22
Challenges 22
Conclusion 24
References 25
Appendix A: The manual 26

Introduction

Biorefinery can provide an alternative way to produce, among others, fuels and chemicals in a sustainable manner, thereby decreasing fossil fuel dependency. Climate change due to global warming is a troubling problem for modern society and forms a large environmental threat to human society and ecosystems globally (Mikael Höök, Xu Tang, 2013). The global warming phenomenon is caused by an increase of greenhouse gasses in earth's atmosphere. The consumption of fossil fuels if one of the major causes leading to an accumulation of carbon dioxide and other greenhouse gasses. Our society is highly dependent on the usage of fossil fuels for the production of materials, chemicals and fuels. Biorefinery is the concept of producing fuels, chemicals, materials and power from various biomass sources. A few examples of biomass sources suitable for biorefining are: Agricultural waste streams, micro-algae and duckweed, but in principle any organic material can be the source material in biorefineries. An example of a complex biorefinery is shown in figure 1. This figure shows how different organic raw materials e.g. straw, Algae and sugar crops, are converted to multiple different products like biodiesel, cattle feed and bioethanol. This is done through multiple processing steps which include pretreatments, conversions and purifications. Such a scheme is an example of a so-called superstructure. Superstructures are typical for biorefinery schemes. Biorefineries consist of a chain of unit operations that transform the biomass source into one or more products through multiple steps. For example, in a micro-algae biorefinery, the algae need to be concentrated, disrupted and the lipids need to be converted to usable fuel. For each of these steps, also referred to as levels in this thesis, multiple techniques are available. This leads to a lot of possible operation chains or "routes". An oversight of these possible techniques and how they can be connected to form working operation chains is called a superstructure. Slegers (2014) has made such a superstructure for an algae biorefinery, which can be seen in figure 2. Biorefineries are essential elements in a biobased-econemy.

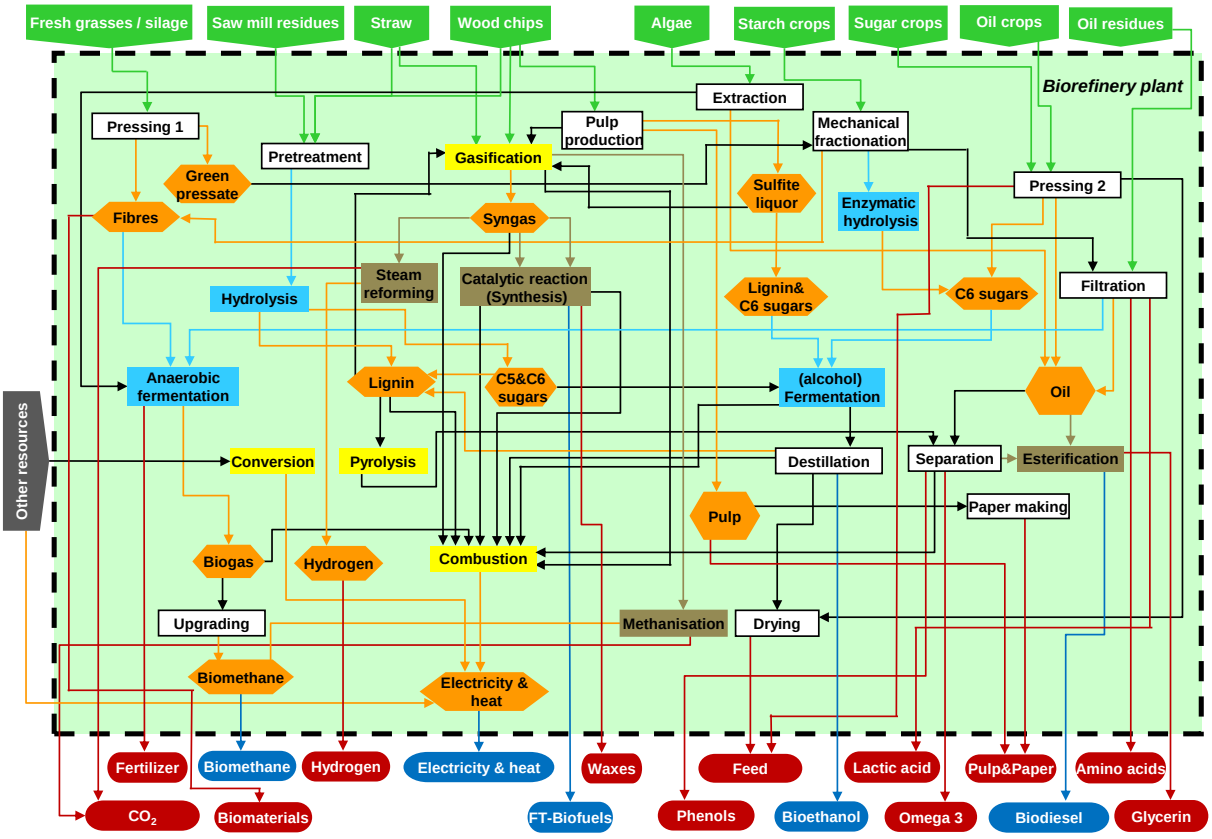

Figure 1 A biorefinery with multiple biomass sources, unit operations and end-products.

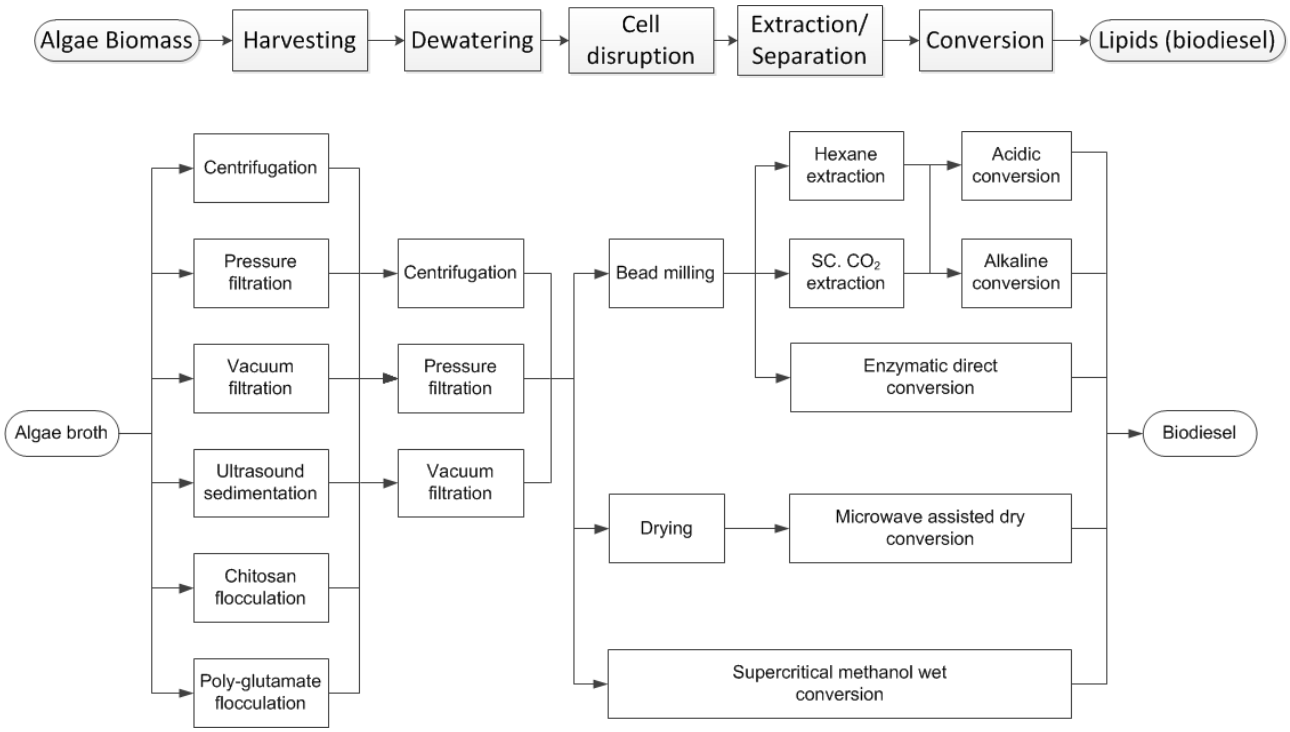

Figure 2 A superstructure for an algae biorefinery as used in Slegers (2014). At the top, the levels for this process are defined. This superstructure is aimed at the production of biodiesel as sole product. This superstructure describes 126 possible process chains.

Superstructures can be made for any multi-step process chain. Although this study focusses on biorefineries as an example for superstructures, superstructures can be found in any process chain where you can choice between several techniques. A few examples are given below:

- Nitrile production from amino acids
- Production of powdered milk from fresh milk
- A water hyacinth biorefinery to produce proteins for cattle feed
- The preparation of edible potato. (Different ways of cutting, spicing, baking, side dish choice, topping).

Modelling superstructures and solving respective optimising problems reveals the most efficient route through a superstructure. With the many possible operation chains that a superstructure can provide the key question is: "which route is most efficient?". The definition of efficient depends on the requirements determined by the designer of the process. This question can be solved using mathematical models for each unit operation. When sequentially executing the models in the order of a route in the superstructure, an estimate of the efficiency of that route can be made. Doing this for each route in a superstructure can predict the most efficient process chain (Zondervan et al, 2011).

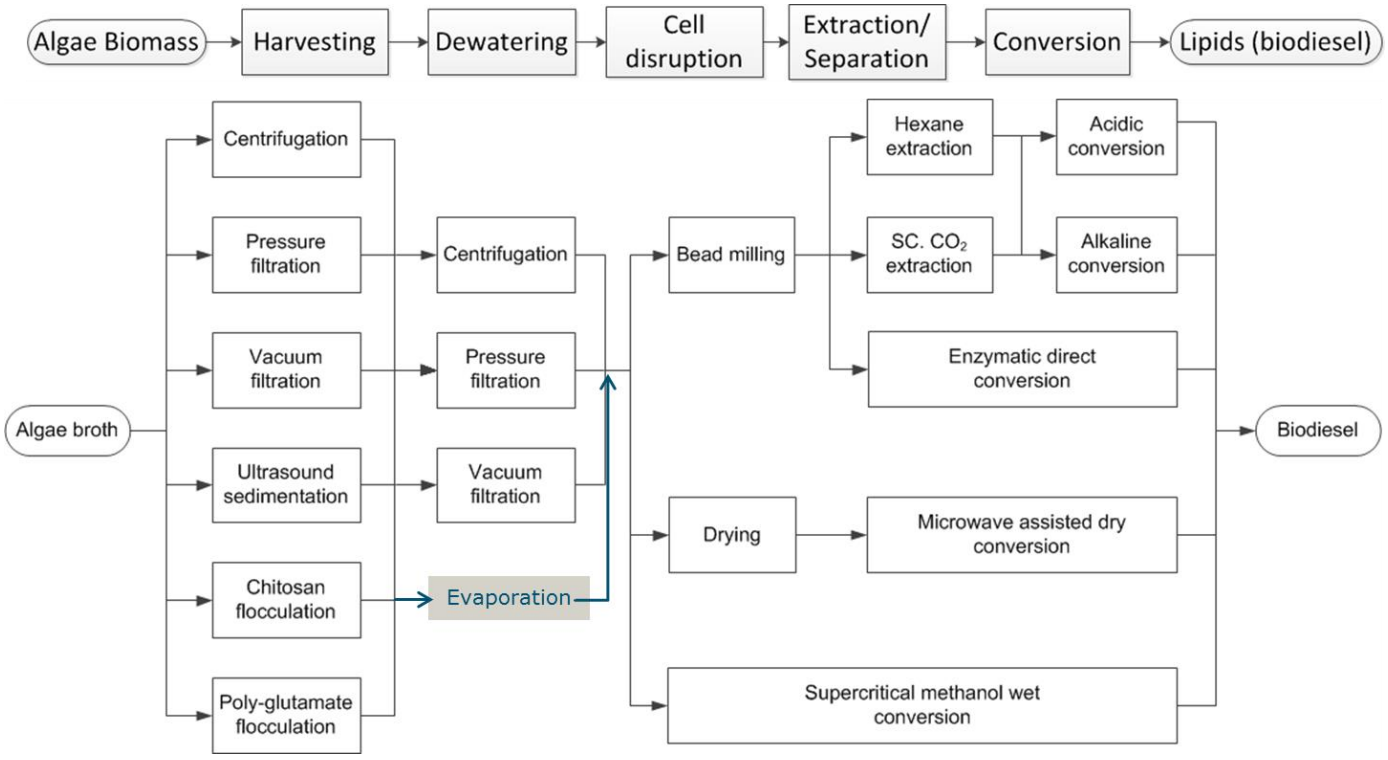

Figure 3 the superstructure for an algae biorefinery as shown in figure 2 with an extra dewatering step indicated by the blue arrows. The number of possible routes through the superstructure grows by 42(from 126 to 168) routes by the addition of this unit.

In a superstructure the different routes that can be taken is not the only variable, also process conditions can vary within a unit operation and affect the overall performance. These variables in operating conditions are called decision variables. The implementation of decision variables in a superstructure makes the optimisation problem more complex, but can provide information on optimal control of the unit operations. For example, the concentration factor of the initial algae broth in an algae biorefinery has a significant impact on the size of the flows for the rest of the process chain. The implementation of decision variables in a superstructure makes finding the best route and conditions much more difficult, optimisation algorithms are required to find the optimal conditions for each route. Optimisation of superstructures and process conditions has been done with various techniques in different papers (Slegers, 2014), (Wang et al, 2013), (Yeomans, H. Grossmann, I.E. 1999). Formulation and optimisation of a superstructure can be tedious if inefficient formulation formats are used and can become unmanageable if problem size increases. Superstructure modelling can be a time consuming task, the time taken strongly depends on the method by which the superstructure optimisation problem is formulated in a modelling environment. Here, superstructure formulation is defined as connecting the functions for the unit operations according to the superstructure and defining the optimisation problem such that a solution can be found by an optimisation method. Yeomans (1999) used a mixed integer nonlinear programming formulation while Slegers (2014) connected all the unit operations for each of the routes manually.

Manual assembly of all routes may work well for small superstructure systems, but will quickly become unmanageable when the superstructure size grows since the number of possible routes grows exponentially with superstructure size. Simplifications to a system need to be made in order to analyse large superstructures, these simplifications come in the form of deleting subunits from the superstructure that seem unlikely to be included in the optimal route. Every simplification made negatively influences the completeness and reliability of the outcome. This challenge leads to the following research question:

- How can the number of simplifications in superstructure based research be minimised in order to increase the quality and completeness of such research?

In this thesis an approach is proposed to improve current methods by avoiding manual assembly. The feasibility of this approach is tested by comparing it with two methods that use manual assembly. This leads us to the objective of this thesis:

- To produce a framework for superstructure formulation and optimisation to reduce time-consuming and inefficient manual input in order to make superstructure formulation and optimisation easier and more efficient.

This framework needs to meet the following requirements:

- The framework needs to be flexible so that superstructures can be readily modified
- The framework needs to be easy to use
- The framework needs to be applicable to all process based superstructures

Methods

In this chapter the approach to creating the framework is discussed. First different approaches for mathematical formulation of the problem are assessed, then a discussion on optimisation tools, followed by the introduction of an exemplary superstructure.

Mathematical problem formulation

The design of a framework strongly depends on the mathematical formulation of the problem and the corresponding solver. The solver needs to be reliable and applicable to all optimisation problems found in superstructures. Due to the non-linear characteristic of practically all process oriented superstructures there are two possible mathematical formulations of the problem, a Nonlinear Programming problem (NLP) or Mixed Integer Nonlinear programming problem (MINLP).

Non-linear Programming problems

A nonlinear programming problem is defined as:

$$
\begin{aligned}
&Minimise\ f(x) \\
&such\ that: C(x) \leq 0 && (1)\\
&C_{eq}(x) = 0 && (2)\\
&Ax\ \leq b && (3)\\
&A_{eq}\ x\ = b_{eq} && (4)\\
&lb\ \leq x\ \leq ub && (5)\\
&x\ \in\ \mathbb{R} && (6)
\end{aligned}
$$

Where *C(x)*, describes nonlinear inequality (≥ or ≤) constraints (eq. 1), $C_{eq}(x)$, describes equality (=) constraints (eq. 2). *C(x)* and $C_{eq}(x)$ are sets of equations. "*A*", describes linear inequality constraints (eq. 3), A_{eq} describes linear equality constraints (eq. 4) and lb, ub describe the lower and upper bounds respectively of decision variable set *x(eq. 6)* (Kuh, H.W. 2013)(Floudas, C.A. 1995) . *A,* $A_{eq,}$ lb and ub are matrices. In the superstructures that were evaluated for this thesis, no linear (in)equalities were present.

This type of optimization problem can be reliably solved with the genetic algorithm and the interior point method. The preferred solver for this in Matlab is the fmincon function. Fmincon uses the interior point method by default, and returns the value of decision variables so that the objective function is optimised. The required input is:

- A function to be optimised, the first output of this function needs to be the objective value.
- A set of decision variables available for optimisation.
- A set of initial values for the decision variables.

Optional input for fmincon is:

- A set of linear constraints, A and b.
- A set of lower and upper bounds for each decision variable, lb and ub.
- A set of nonlinear constraints in the form of a function that provides Ci, the nonlinear inequality constraints, and Ceq, the nonlinear equality constraints.
- In the additional input "*Options*" the maximum number of iterations, outputs etc. can be specified to tailor fmincon to your problem and required accuracy of the solution.

In a superstructure each route can be solved as a separate NLP problem. As a superstructure gets larger, the number of routes increases exponentially. As a result the computational time also grows exponentially with the superstructure size.

Mixed Integer Non-linear Programming problems

A mixed-integer nonlinear programming problem is defined as:

$$Minimise\ f(x)$$
$$Such\ that: C(x) \leq 0 \quad (7)$$
$$C_{eq}(x) = 0 \quad (8)$$
$$Ax \leq b \quad (9)$$
$$A_{eq}\, x = b_{eq} \quad (10)$$
$$lb \leq x \leq ub \quad (11)$$
$$x \in \mathbb{R} \quad (12)$$
$$y \subseteq x \quad (13)$$
$$y \in \mathbb{Z} \quad (14)$$

Where $C(x)$ describes nonlinear inequality constraints(eq. 7), $C_{eq}(x)$ describes equality constraints(eq. 8), A describes linear inequality constraints (eq. 9), A_{eq} describes linear equality constraints (eq. 10) and lb, ub describe the lower and upper bounds of decision variable set x (eq. 11). The difference between MINLP and NLP problems lies in Y. Y is a subset of variable set x (eq. 13) that contains the integer variables (eq. 14) (Floudas, C.A. 1995). In superstructures that were evaluated for this thesis, no linear (in)equalities were present.

Usually, to solve this type of problem a NLP algorithm is combined with a branch and bound algorithm with mutations, similar to the genetic algorithm. There is no reliable solver within Matlab to tackle this particular difficult type of problems. Different solvers are available as extensions of Matlab, like the KNITRO tool pack in the Matlab extension, Tomlab. KNITRO uses an algorithm of the class interior point and active set to solve NLP problems and supports MINLP solving by Branch and bound and hybrid Quesada-Grossman (which works by solving the MINLP problem first as an NLP problem, then as an MILP problem) algorithms.

In a superstructure, the whole system can be analysed at once as a MINLP problem, where integer decision variables indicate a chosen subunit from the next level. As a superstructure gets larger, the computational time when solving it as an MINLP problem, grows linearly.

NLP vs. MINLP

The decision of whether to formulate the superstructure as a set of NLP problems or as a MINLP problem is fundamental to the design of the framework. A practical oriented comparison of these methods is shown in table 1.

Table 1 A comparison of NLP or MINLP problem formulation of the superstructure.

	Set of NLPs	MINLP
Computational time in relation to problem size	+/- Grows exponentially with problem size	+ Grows linearly with problem size
Reliability of solving the problem in relation to problem size	++ Takes longer, but unchanging reliability	+/- Becomes more unreliable, the chance of getting stuck increases
Steadiness in time taken for solving a problem	++ Each NLP adds a more or less fixed amount of time	- Inherently, branch and bound has no guarantee to find a solution in a set amount of time
Software requirements	Matlab	Matlab + Tomlab
Solver	fmincon	KNITRO

The steadiness of time taken and reliability of finding a solution in a set amount of time led to the decision to formulate the superstructure optimisation problem as a set of NLP problems. The exponential growth of computational time for a NLP formulation is a drawback when compared to the linear growth of a MINLP formulation. There are however two options in the NLP formulation to decrease computational time that are unavailable for MINLP formulation:

- The number of decision variables per route
- Parallel computing

NLP problems can be formulated using only the decision variables that apply to the route, therefore limiting the number of decision variables in each NLP route. The number of decision variables per route grows as process chains grow longer, but not when increasing the number of routes. MINLP problems have to analyse all decision variables of the entire superstructure at the same time.

The NLP problems do not depend on intermediate values from each other for solving, and can thus be solved parallel to each other. An octacore processor can therefore solve 8 NLPs at the same time, while for an MINLP problem the cores cannot be used as efficiently. The parallel nature also allows outsourcing of (part of) the NLP problems to servers or computer clusters while this would be harder for MINLP problems.

Optimisation tools

Optimisation problems come in many forms and multiple options to solve these problems exist, a comprehensible explanation of the tools considered for this thesis is given below. The focus lies on providing a basic understanding how the techniques work rather than a complex mathematical analysis.

Genetic algorithm

The genetic algorithm is based on the principles behind natural selection. A random set of numbers is evaluated. Successful numbers become parents and create offspring children with similar characteristics. The new generation of numbers also consists of a few clones of the best scoring parents and some new random numbers. These new random numbers are called mutations. This is illustrated in figure 4.Over the generations the population will eventually concentrate on an optimum.

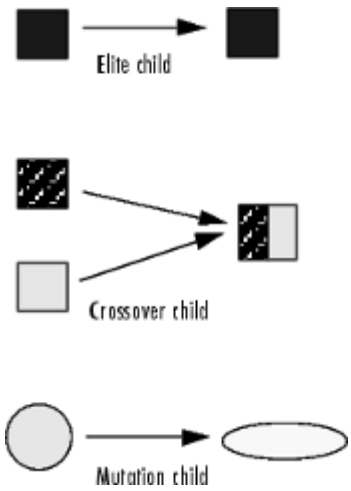

Figure 4 illustration of the formation of the new generation in the genetic algorithm.

Branch and bound

The branch and bound method is an important tool for integer or mixed-integer optimisation problems. It is used to find an optimal discrete value after an optimal continuous solution is found. Figure 5 illustrates this method. After a non-discrete solution is found, two new optimisation problems are generated with extra boundaries. For example, if the non-integer optimum is found at a decision variable value of 3.3 while an integer solution is required, two new problems are generated looking at optimal solutions ≤3 and ≥4 (the branch). The answers to these new problems are compared, and the path of the lower value is not further pursued (the bound). This process is repeated until an optimal integer solution is found (Claassen et al, 2007).

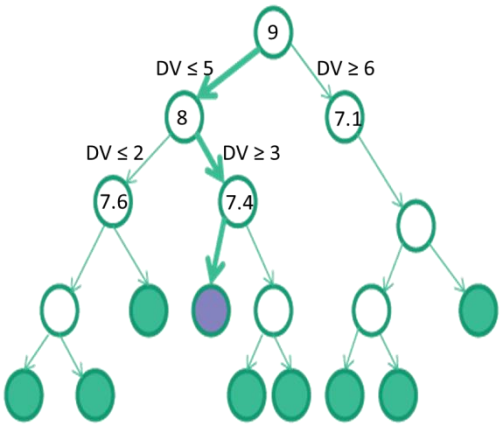

Figure 5 Branch and bound method for finding an optimal discretised solution. The white circles are optimal, non-discrete solutions, the green circles are discrete optimal solutions of branched optimisation problems and the purple circle is the best discrete solution to the overall problem. The values in the circles are the objective value function. Source: http://www.gurobi.com/resources/getting-started/mip-basics

Interior point method

The interior point method is an algorithm that solves optimisation problems in polynomial time by using Newton's method, which is illustrated in figure 6. A few modifications are made to an optimisation problem before Newtons method is applied. A constrained problem is turned into an unconstrained problem by

applying either Lagrange multiplier methods or the implementation of logarithmic barrier functions, depending on the type of constraint (Jensen et al, 2003 for further reading). Newton's method finds the intersection of the derivative with the horizontal axis where the original function has an optimum. This is done by finding the second degree derivative at an initial point in the curve, this second degree derivative is extended linearly to the horizontal axis. From the X-value at this point, a new second derivative is made and extended linearly to the horizontal axis e.g. x1 and x2. This process is repeated until the intersection between the horizontal axis and the first derivative is approached as precise as required.

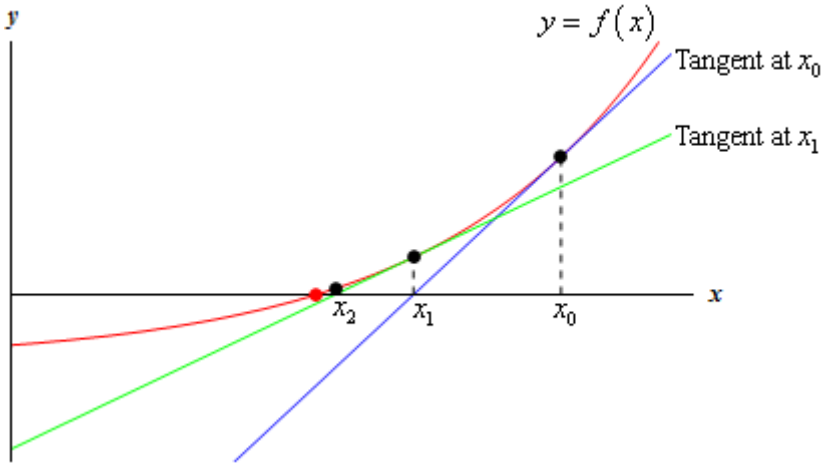

Figure 6 illustrates the iterative process of Newton's method to find an intersection with the x-axis of a curve. The derivative of initial point is X_0 is determined and linearly extended to the x-axis, this gives us the new derivative value of X_1. Source: http://tutorial.math.lamar.edu/Classes/CalcI/NewtonsMethod.aspx

Dynamic programming

Dynamic programming is a method for solving superstructure optimisation problems. The disadvantages of the method were too large for dynamic programming to be a viable method. Dynamic programming is a method that solves a optimisation problem by breaking it down into smaller and simpler sub-problems which are solved individually. The optimal solution for a small sub-problem is quickly found and saved in the memory. Then a larger sub-problem is formulated which contains the previous-sub problem. Instead of solving this new sub-problem as a whole, the optimal solution to the first problem is used as a basis (Claassen et al 2007). This can be illustrated by thinking of the problem and its sub-problems as an onion. The core of the onion forms the most basic sub-problem. Once a solution to this problem is found, the problem is expanded by adding an extra layer to the problem. Basically, for each point of a optimisation problem the optimal solution is listed in a large table. Since no infinite solutions can be stored for continuous variables, discretisation of variables is required. Discretisation of variables causes rounding errors in each discretisation, leading to an overall error accumulation throughout the process. This accumulation error is the reason for abandoning this method.

Route analysis

The number of possible routes through a superstructure grows exponentially with the problem size. You can analyze the total number of routes through a system by splitting it into smaller pieces and analyzing the number of routes to reach each subunit. This can be done with the simple equation:

$$\#\ subunit A = \sum \#\ subunit\ of\ \ each\ subunit\ in\ a\ previous\ stage\ that\ is\ connected\ to\ subunit A \qquad (15)$$

Where # is "number of possible routes through the superstructure to get to a subunit". This equation is applied on each subunit from start to end. Figure 7 illustrates how the number of routes through a

superstructure can be calculated. Figures 2 & 3 illustrate the rapid growth of the number of possible routes through a superstructure when the superstructure increases in size.

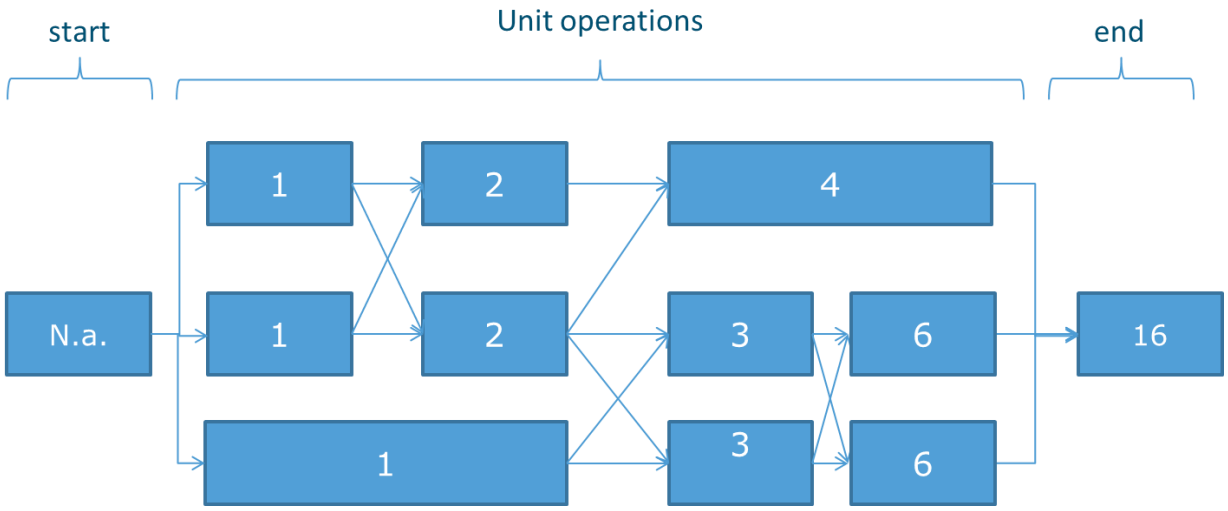

Figure 7 Illustration of calculating the number of possible routes through a superstructure. The numbers in the boxes indicate how many possible routes there are in the superstructure to get to that particular unit operation from the start. The system is analysed from left to right by applying eq. 15.

Conceptual superstructure

To create a framework for a superstructure, it is easiest to start with a simple system and gradually add complexity. Therefore a conceptual superstructure is used, which contains a limited number of routes and uses simple subunit functions. The main advantage of starting with very simple systems is that all important elements e.g. number of routes, route composition, proper optimisation, are known by heart by the designer. This gives the designer the ability to spot problems and flaws in an early stage and correct the framework accordingly.

The complexity of the subunits does not matter as long as the correct inputs and output formats are used and generated. As a result the problem in the conceptual superstructure can be a simpler linear programming problem. The conceptual superstructure used for framework design is shown in figure 8.

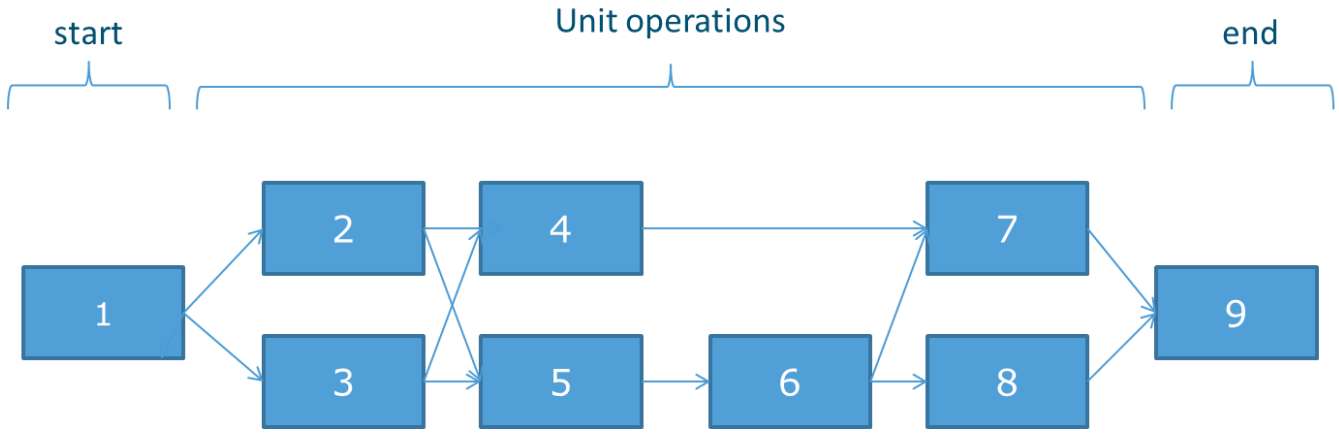

Figure 8 The layout of the conceptual superstructure used in framework design. It has 9 subunits (the numbers in the boxes) and 6 possible routes. Subunits will operate in the following fashion: variable = variable + a random number, for 5 variables.

Input format

The goal of the superstructure framework is to solve superstructure optimisation problems, while minimising the amount of manual input and coding required to do this. Therefore it is important to find the most condensed form of input that can be used to define a superstructure. For example: a list of decision variables used per route has thus far been the norm for manual input when it comes to decision variables, while only a list of which subunits require which decision variable(s) and a list of routes is required to automatically construct this list. A function providing this list already saves the effort of considering each route again and summing up the decision variables for this route. This kind of consideration is made for each of the current manual inputs so a template for minimal input can be derived. The framework uses these inputs to formulate the entire superstructure as a set of NLP problems in a format that is usable by fmincon for optimisation.

Results and Discussion

The Framework

The framework consist of 7 functions. The approach that the framework takes for formulating and optimising a superstructure is shown in figure 9. Each of the manual inputs as shown in figure 9 are elaborated on in the following subsections. The framework was made using the exemplary superstructure from the method section.

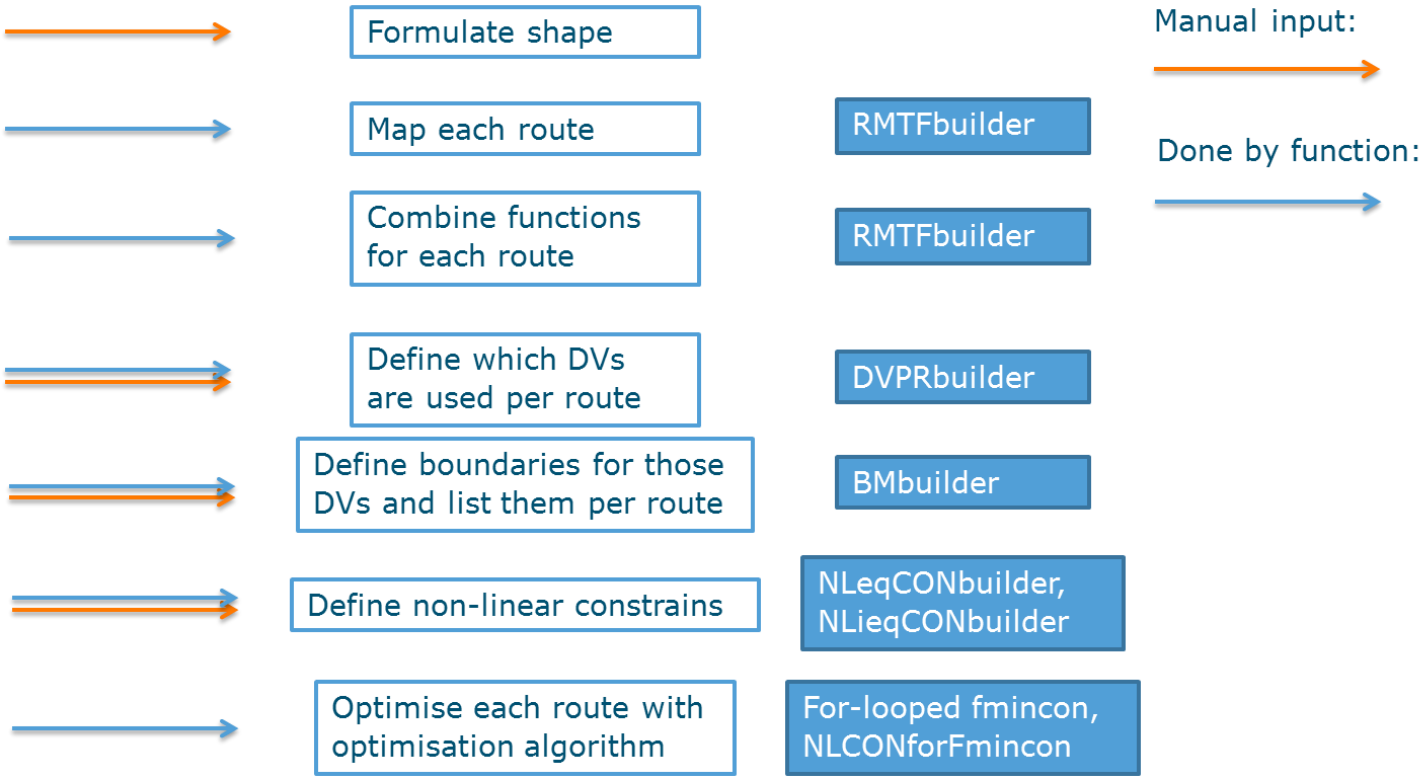

Figure 9 The different steps for formulating a superstructure through the framework. The steps are placed in the white boxes and the function(s) that are used in this step are placed in blue boxes. A distinction is made between manual input (orange) and actions that are done automatically by the framework (blue).

Superstructure formulation

The first manual input that needs to be defined is the shape of the superstructure (see Figure 9). The shape of a superstructure is a description of connections between subunits. There are multiple ways to formulate the shape. In the methods section the format of describing a superstructure for an MINLP problem was introduced. The MINLP format of formulating a superstructure shape requires subunits to be placed in certain levels (e.g. harvesting, disruption, conversion etc.) whereafter "forbidden" connections to the next level are described. This method has its disadvantages, especially in larger and chaotic systems. An early indication of one of these disadvantages can be seen in figure 2. One route uses microwave assisted dry conversion, however, the microwave assisted dry conversion requires a drying step beforehand. This drying step is currently placed in the disruption level, although it does not perform actual disruption. This does not lead to any difficulties in this model, but when analysing large and complex superstructures, it becomes harder to subdivide every subunit in a specific level of the superstructure. To avoid this problem, another method of describing the shape of a superstructure is proposed that does not have this disadvantage, the so-called "Arrow Matrix"

The Arrow Matrix (AM) describes the superstructure by defining connections that are allowed. The Arrow Matrix literally describes each "arrow" going from one subunit to the next. One arrow is defined as a 1x2 vector containing the number of the subunit from which the arrow originates to the subunit it points to. An example: [5,9] would mean that subunit number 5 is compatible with subunit number 9. The AM method results in an N by 2 matrix. Figure 10 illustrates how the arrowmatrix is made from an exemplary superstructure.

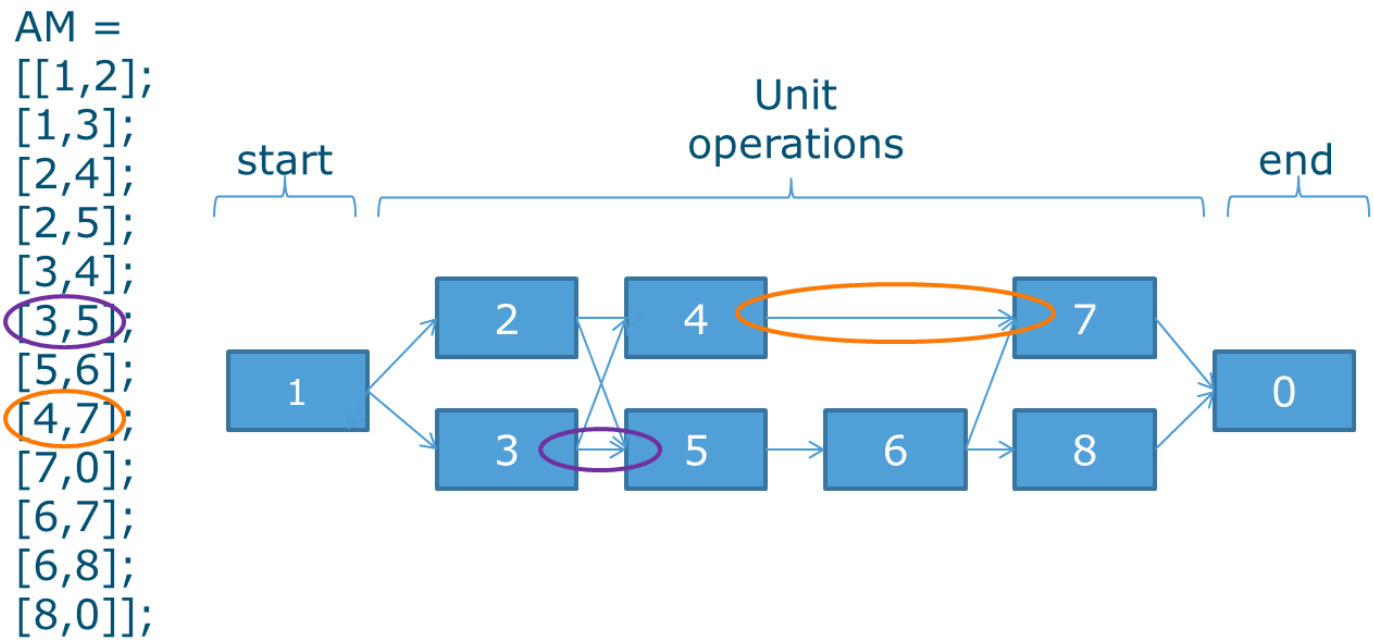

Figure 10 The arrowmatrix (AM) is made by describing each connection in a superstructure as a 1x2 vector. E.g., if unit 4 can be followed by unit 7, then [4,7] is added to the AM (illustrated in orange in the figure).

For Slegers' (2014) model, which has 126 routes, The AM would result in a 46x2 matrix. This may seem large, but the input format is so easy that describing 46 connections takes a couple of minutes. The main advantages of the AM method over the MINLP format are its simplicity, easy modification of the superstructure and abandon the need to assign a level to each subunit. The latter is especially useful in complex and chaotic superstructures as seen in Figure 1. In the example given in figures 2 and 3, where one additional subunit was added to the superstructure in figures 2 & 3, the number of possible routes increased with 42, while the arrow matrix only increases in size by 9. An increase of 9 rows is even close to a worst case scenario, as adding a subunit similar to *enzymatic direct conversion* in this superstructure increases the size of the arrow matrix by 2 while adding 24 routes to the superstructure.

From the AM a list of all possible routes through the superstructure can be derived. This is done by the RMTFbuilder. This function can effectively map all possible routes through the superstructure and places them in a Route matrix (RMF) containing all the function handles in the order of operation. The RMF is used by following builder functions as well. The RMTFbuilder needs to be able to recognise where to start and where to end, this is made possible by a fixed subunit, which is always numbered 1 and is placed in front of the superstructure, and one subunit which is always numbered 0 which is placed at the end of the superstructure. The last subunit does not require a functions while and the first can be either an empty function or one where initial values are defined.

Function combining

Next to the shape of the superstructure, another task that requires excessive manual work is combining the functions for all subunits into one function for each route. Each route will be separately optimised. Combining functions is done by filling in the output of one function as the input of the following function. This action is performed for each unit in a route from first to last.

For this task to be automated, it is imperative that subunits have compatible inputs and outputs, otherwise it is simply impossible to achieve. A suggested formulation of inputs and outputs is as follows: There is one argument that needs iteration over the different subunits, which contains all values for the likes of concentrations, flows, and energy consumptions (this argument will be referred to as the Primary Component Matrix (PCM)). This argument is the first input and first output of each subunit function. The second input is a list of decision variables required for that specific route. All subsequent arguments can contain inputs that are identical for each subunit function e.g. parameters like, density, viscosity, gas constants etc. As second output, the value of the objective function is required.

When the requirements of standardised subunit formulation are met, all functions in a route can be combined into one function for each route. This results in a Route Matrix with Total Functions (RMTF). These functions will be used in the optimisation process.

Decision variables

The minimal required input to define which decision variables per route are used, is a list of which unit operation uses which decision variable. This list with Decision Variables per Unit Operation (DVUO) together with the list of which subunits are used per route, the RMF, can be combined to derive the decision variables used per route (DVPR). The DVUO is formulated as shown in table 2. The function first recognises which subunits are used in a route through the RMF and then lists all associated decision variables for that route. Thereafter, all unique elements from this list are identified, sorted and placed in the DVPR matrix. This process is done for each route in the RMF.

Table 2 The format for describing decision variables per unit operation, the number of the row corresponds with the number of the subunit it represents, so the order of this vector is fixed.

Unit operation (row number):	Decision variables used:
1	DV 1
2	
3	DV 2, DV 3
4	DV 1, DV 4

Lower and Upper bounds

The minimal required input to define which lower and upper bounds are used per route, is a list of which decision variable is subjected to which bounds. This is described in the Lower Bounds, Upper Bounds matrix (LBUB) and is formulated as shown in table 3. The function that is used to build matrices describing the lower and upper bounds uses the LBUB and DVPR matrices to create the Lower Bound Matrix (LBM) and Upper Bound Matrix (UBM).

Table 3 The format for describing which decision variables are subjected to which lower and upper bounds, empty bound values are substituted by - infinite and + infinite respectively.

Decision variable (row number):	Decision variables used:
DV 1	lower bound value, upper bound value
DV 2	lower bound value
DV 3	
DV 4	lower bound value, lower bound value

Non-linear constraints

The key of non-linear constraints is that at one specific subunit in a route, for one specific component, either the input or output of that subunit is subjected to restrictions. This can be either equality constraints or inequality constraints. For the inequality constraints, the subjected value can be either greater or equal than (≥), or smaller or equal than (≤) a given value. The last characteristic causes the non-linear equality and inequality constraints to be handled slightly different. The minimal input required for the non-linear constraints consists of:

- The subunit for which these restrictions are applicable.
- The position of the relevant component in the primary component matrix.
- The value of the boundary.
- Whether the boundary value should be compared to the input or the output of the function.
- Whether the value should be greater or equal than (≥), or smaller or equal than (≤) the boundary value (for inequality constraints only).

A template for defining these values is shown in tables 4 and 5.

Table 4 Format for describing equality constraints, filled with example values.

Subunit (functionhandle)	Index of component in PCM	Boundary value	Input or output (1 or 0 respectively)
@subunit1	2	21.4	1
@subunit3	6	-80	1
@subunit4	6	40	0

Table 5 Format for describing inequality constraints, filled with example values.

Subunit (functionhandle)	Index of component in PCM	Boundary value	Input or output (1 or 0 respectively)	Above or below boundary value (1 or -1 respectively)
@subunit1	4	21.4	1	1
@subunit2	3	4	0	-1
@subunit5	4	50	1	1

There are two reasons why nonlinear constrains are the most complex items to implement. First, they need to be formulated as a function instead of regular values. Second, they apply to intermediate values within the route which cannot be found by functions from the RMTF.

The approach for an efficient non-linear constraint function builder is to first create separate matrices for:

- Combined functions of all subunit functions op to (and including if input or output = 1) the subunit given in column 1 of the constraint matrix for each subunit in the constraint matrix present in a route. Named Non-Linear (in)equality Constraints functions (NL(i)eqCONf).
- The boundary values corresponding to the functions in NL(i)eqCONf for each route, named Non-Linear (in)equality Constraint numbers (NL(i)eqCONn.
- A list of indices of the constrained component in the primary component matrix corresponding to the functions and numbers in NL(i)eqCONf and NL(i)eqCONn, named "Function output indices (in) equality" (fOutindices(i)eq).

And for inequality constraints only:

- A matrix with multiplication factors, either 1 or -1, to indicate whether the value should be ≥ or ≤ the boundary value given in NLieqCONn.

When these matrices are constructed, it is possible to write a function that gives the equation sets *Ci(x)* and *Ceq(x)* as output required for fmincon. If the above matrices are not pre-constructed, they would have to be evaluated with every iteration of fmincon, significantly slowing the process to the point of unviability as every iteration takes approximately a second. By pre-constructing these matrices, only a "read" operation has to be performed on the matrices.

The final non-linear constraint function evaluates the function from NL(i)eqCONf with the decision variables and subjects the outcome to the boundary values given in NL(i)eqCONn through the mapping provided by fOutindices(i)eq. It does so for both equality and inequality constraints to give final values of $Ci(x) \leq 0$ and $Ceq = 0$ as used by fmincon for each route. This function is called Non-Linear Constraints for fmincon (NLCONforFmincon).

Optimisation

Every input required for fmincon is described by the framework. These inputs can be combined and evaluated by fmincon for each route. This is done by a *for-loop* where every route, i.e. every row, of RMTF, DVPR, LBM, UBM and the NLCONforFmincon function where NLCONforFmincon is dependend on the PCM, DVPR, DVIPR, route/row number, NLeqCONf ,NLeqCONn, fOutindiceseq, NLieqCONf, NLieqCONn, fOutindicesieq and multiplicationfactorieq, is evaluated.

Comparing the framework with other methods

The framework was applied to the algae biorefinery superstructure, as shown in table 6. This allowed comparison to two other methods for formulating a superstructure. The effectiveness of the framework is evaluated based on time taken, amount of code written and the number of files required to formulate a superstructure. The other methods are the original method from Slegers 2014 and Houthoff's MSc thesis project. The results of this comparison are displayed in table 6.

Table 6 A comparison of different methods of superstructure formulation.

	Enumeration (slegers,2014)	NLP optimisation with manual routing (Houthoff,2015)	NLP optimisation through framework
Number of routes analysed	126	19	126
Time taken for formulation	± 9 days minimum	± 2 days	± 2 hours
Amount of code	± 15.000 lines	± 400 lines	± 250 lines
Number of files for formulation and optimisation	2	39 (Would be 253 for 126 routes)	1

The framework significantly reduces the time taken, code used and files used for formulating and optimising a superstructure compared to the two previously used methods. Furthermore, the framework provides flexibility to a superstructure and will reduce the risk of forgetting routes. The framework cannot be compared to an MINLP formulation since there is no data available yet. Further research can provide this data and complete the comparison.

Challenges

Integer decision variables

The framework work is based on NLP problems. The NLP approach causes a challenge when integer decision variables are used in the superstructure as those variables cannot be handled by the solver. There are two possibilities to implement integer decision variables in this framework:

- Implement the integer variable as a continuous variable and round it to an integer within the unit operation.
- Turn the integer decision variable into extra unit operations for every possible value of the decision variable and solve it through extra routes instead of as a decision variable.

The first of the options is only suitable for a "large" range of possible values for the decision variable while the second option is only suitable for a "small" range of values. In this case, large can be defined as >5 and small as <5.

Local versus global optima

The fmincon solver finds a local minimum while global minima need to be found for superstructure optimisation. A multistart option can be implemented to counter this problem. This can be done by placing the final *for* loop that contains the fmincon operation into another *for* loop that will perform all optimisations several times. The more times an optimisation is performed, the greater the chance that the global optimum is found.

Recycling of streams within a route

The modelling of recycling side streams in process systems can be challenging. In this framework it is currently possible to recycle using fixed side streams through the use of nonlinear equality constrains and it is possible to recycle within the same unit operation. A small change needs to be implemented to the function combining process to allow iterative recycling over a multi-unit operation loop. Further research is required.

Conclusion

In the introduction three requirements were posed for this framework. The framework needed to be flexible, easy to use and applicable to all process related superstructures. The flexibility is realised by the usage of the arrowmatrix, the addition or deletion of subunits from a superstructure is easy and quick. The framework is easy in use due to the input formats and minimal manual coding, no advanced understanding of Matlab is required to apply the framework. The framework is applicable to all process related superstructures due to the use of NLP problems, which have very reliable solvers.

References

Claassen, Hendriks, Hendrix (2007) Decision science. Wageningen, Wageningen academic publishers.

Floudas, C.A. Nonlinear and Mixed-integer optimisation fundamentals and applications. Oxford University Press, Inc. 1995

Höök, M., Xu Tangb. Depletion of fossil fuels and anthropogenic climate change—A review. Energy Policy. Volume 52, January 2013, Pages 797–809.

Houthoff, I. Development of a sustainable biorefinery process from microalgae. Biobased Chemistry and Technology. MSc thesis, 2015

http://tutorial.math.lamar.edu/Classes/CalcI/NewtonsMethod.aspx, retrieved may 2015.

http://nl.mathworks.com/help/gads/how-the-genetic-algorithm-works.html, retrieved may 2015.

http://www.gurobi.com/resources/getting-started/mip-basics, retrieved may 2015

Kuhn, H.w. Nonlinear Programming: A Historical View. Traces and Emergence of Nonlinear Programming 2014, pp 393-414

Ranjan Parajuli, Tommy Dalgaard, Uffe Jørgensen, Anders Peter S. Adamsen, Marie Trydeman Knudsen, Morten Birkved, Morten Gylling, Jan Kofod Schjørring, Biorefining in the prevailing energy and materials crisis: a review of sustainable pathways for biorefinery value chains and sustainability assessment methodologies, Renewable and Sustainable Energy Reviews, Volume 43, March 2015, Pages 244-263.

Slaper, T, F. Hall, T, J. "The Triple Bottom Line: What Is It and How Does It Work?" Indiana Business Review. Spring 2011, Volume 86, No. 1. 2011.

Slegers, P.M. Koetzier, B.J. Fasaei, Wijffels, F. R.H. van Straten, G. van Boxtel, A.J.B.A model-based combinatorial optimisation approach for energy-efficient processing of microalgae, Algal Research, Volume 5, July 2014, Pages 140-157.

Wang, B. Gebreslassie, B. H. You, F. Sustainable design and synthesis of hydrocarbon biorefinery via gasification pathway: Integrated life cycle assessment and technoeconomic analysis with multiobjective superstructure optimization, Computers & Chemical Engineering, Volume 52, 10 May 2013, Pages 55-76,.

Yeomans, H. Grossmann, I.E. A systematic modeling framework of superstructure optimization in process synthesis. Computers & Chemical Engineering. Volume 23, Issue 6, 1 June 1999, Pages 709–731.

Zondervan, E. Nawaz, M. De Haan, A.B. Woodley, J.M. Gani, R. Optimal design of a multi-product biorefinery system. Computers & Chemical Engineering. Volume 35, Issue 9, 14 September 2011, Pages 1752–1766

Appendix A: The manual

Manual for the superstructure formulation and optimisation framework

By: Tim Hoogstad, version June 2015
(Matlab version 2014b)

Contents

List of abbreviations 27
Scope 27
Contents of the toolbox 27
Description of functions, inputs and outputs 28
BMbuilder 28
DVPRbuilder 28
NLCONforFmincon 28
NLeqCONbuilder 28
NLieqCONbuilder 29
OG 29
RMTFbuilder 29
Superstructure formulation (inputs and function execution order) 31
Step 0. Requirements and framework tuning 31
Step 1. Shape formulation and function combination 32
Step 2. Defining Decision variables 34
Step 3. Defining DV boundaries 35
Step 4. Define the non-linear (in)equalty constrains 36
Step 5. Define additional parameters 36
Superstructure optimising 37

List of abbreviations

Other abbreviations may be from these abbreviations

AM – Arrow Matrix

CON – Constraints

DV – Decision Variable

DVI – Decision Variable Index

Eq – Equality

FHM – Function Handle Matrix

Ieq – Inequality

LB – Lower Bound

NL – Non-Linear

PCM – Primary Component Matrix

PR – Per Route

RMF - Route Matrix with Function handles

RMTF – Route Matrix with Total Functions

UB - Upper Bound

UO – Unit operation

Scope

This framework provides an easy way for superstructure formulation and optimisation, thereby reducing time-consuming manual input. In the framework the superstructure formulation is done by a series of functions and requires a limited amount of manual input. The functions formulate the superstructure as a series of non-linear programming problems, one Non-linear programming problem for each possible route. Superstructure optimisation is done by solving these NLP problems in Matlab with the function Fmincon.

Contents of the toolbox

The toolbox consists of a script template and the following functions:

- BMbuilder.m
- DVPRbuilder.m
- NLCONforFmincon.m
- NLeqCONbuilder.m
- NLieqCONbuilder.m
- OG.m
- RMTFbuilder.m

First a description of the function inputs and outputs is given, whereafter the functions are applied to an exemplary superstructure.

Description of functions, inputs and outputs

BMbuilder

```
function [ LBM , UBM ] = BMbuilder( DVPR, LBUB )
```

Description

The BMbuilder creates arrays that describe the upper and lower boundaries of decision variables, compatible with Fmincon.

Inputs

- DVPR, as created by DVPRbuilder
- LBUB, manual input describing the LB and UB per DV

Outputs

- LBM, a matrix listing all lower bounds(columns) applicable in each route(rows) as used by Fmincon
- UBM, a matrix listing all upper bounds(columns) applicable in each route(rows) as used by Fmincon

DVPRbuilder

```
function [ DVPR , DVIPR ] = DVPRbuilder( RMF , DVUO, FHM , Order)
```

Description

The DVPRbuilder creates lists for all decision variables and their index in the order matrix per route

Inputs

- RMF, as created by RMTFbuilder
- DVUO, manual input describing which DVs are used by which UO
- FHM, manual input defining the function handles for each UO
- Order, the order of the decision variables that is handled.

Outputs

- DVPR, DVs(columns) used per route (rows)
- DVIPR, DVIs(columns) used per route (rows)

NLCONforFmincon

```
function [ Ciout,Ceqout ] = NLCONforFmincon( Components, DV, routenumber,
NLeqCONf, NLeqCONn, fOutindiceseq, NLieqCONf, NLieqCONn, fOutindicesieq,
multiplicationfactorieq )
```

Description

Uses premade outputs from other functions and formats them to be compatible and quickly accessible by Fmincon.

Inputs

- Components, the PCM
- DV, the DVs for the route described in routenumber
- Routenumber, the number of the route
- NLeqCONf, as created by NLeqCONbuilder
- NLeqCONn, as created by NLeqCONbuilder
- Foutindiceseq, as created by NLeqCONbuilder
- NLieqCONf, as created by NLieqCONbuilder
- NLieqCONn, as created by NLieqCONbuilder
- foutindicesieq, as created by NLieqCONbuilder
- multiplicationfactorieq, as created by NLieqCONbuilder

Outputs

- Ciout, describing the inequality constraints in a format compatible with Fmincon
- Ceqout, describing the equality constraints in a format compatible with Fmincon

NLeqCONbuilder

```
function [ NLeqCONf, NLeqCONn, fOutindiceseq ] = NLeqCONbuilder( RMF,NLeqCON )
```

Description

NLeqCONbuilder creates one function handle matrix and two arrays describing the non-linear equality constraints per route.

Inputs

- RMF, as created by RMTFbuilder
- NLeqCON, manual input describing non-linear equality constrains

Outputs

- NLeqCONf, combined functions of the route up to the point where the equality constraint is in place
- NLeqCONn, the value that the outcome of the function described in NLeqCONf needs to match
- fOutindiceseq, the index of the component which is subjected to the constraint in the PCM

NLieqCONbuilder

```
function [ NLieqCONf, NLieqCONn, fOutindicesieq, multiplicationfactorieq ] =
NLieqCONbuilder( RMF,NLieqCON )
```

Description

NLieqCONbuilder creates one function handle matrix and three arrays describing the non-linear equality constraints per route.

Inputs

- RMF, as created by RMTFbuilder
- NLieqCON, manual input describing non-linear inequality constrains

Outputs

- NLeqCONf, combined functions of the route up to the point where the equality constraint is in place
- NLeqCONn, the value that the outcome of the function described in NLeqCONf needs to match
- fOutindiceseq, the index of the component which is subjected to the constraint in the PCM
- multiplicationfactorieq, an list of indicators whether the function outcome should be above or below the reference value from NLeqCONn

OG

```
function varargout = OG(func,outputNo,varargin)
```

Description

The Output Grabber (OG) grabs the output(s) of your choice for direct input in another function without needing extra lines of code.

Inputs

- Func, the function it needs to grab the output from
- OutputNo, the number(s) of the output(s) it needs to grab
- Varargin, the original inputs for func

Outputs

- Varargout, the requested output(s) of the function func

RMTFbuilder

```
function [ Routematrixfunctions, RoutematrixTotalfunction ] = RMTFbuilder(
Compatibilitymatrix, Functionhandlematrix  )
```

Description

The RMTFbuilder formulates the shape of the superstructure in a forms the basis for the rest of the framework and describes the combined functions for each route.

Inputs

- Compatibilitymatrix, manual input describing the connections between UOs, also referred to as AM
- FunctionHandleMatrix (FHM) manual input linking to the functions that describe the UOs

Outputs

- Routematrixfunctions (RMF), a matrix listing all UOs (columns) per route (rows)
- Routematrixtotalfunctions, a matrix containing one combined function per route describing the entire route

Superstructure formulation (inputs and function execution order)

The general approach and different steps within the framework are summarised in figure 11. Each of the manual inputs is described in this chapter.

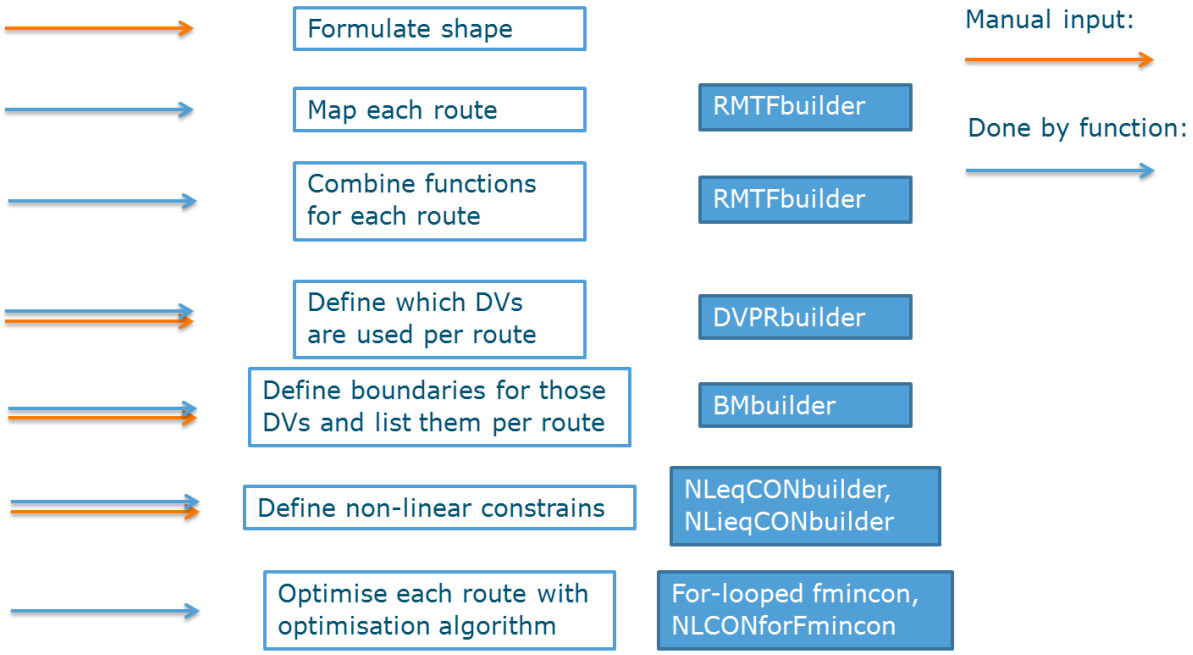

Figure 11 the general approach and steps taken by the framework, a distinction is made between manual input and automated actions.

Step 0. Requirements and framework tuning

Function formats

Create functions for all UOs with consistent inputs and outputs. Adjust functions RMTFbuilder, NLeqCONbuilder and NLieqCONbuilder to your function inputs.

The functions that were written for each unit operation have to be uniform in input and output formats. Each unit has three obliged inputs.

- The Primary Components Matrix (PCM) where every component that requires iteration over the unit operations is listed e.g. concentrations of compounds, energy consumption and co-streams.
- The Decision Variables Per Route for each route (DVPR) so that fmincon only varies the right DVs.
- The Decision Variables Index Per Route (DVIPR) so that UOs know which DV was originally meant.

On top of these three obligatory arguments additional inputs can be used that describe parameters (par) of which the value does not change, e.g. densities of compounds, gas constants.

Each function has two obligatory outputs, a new version of the PCM with modified values and the value of the objective function for the new PCM.

An example of a function and the arguments in Matlab could be:

```
function [ PCM, Objective ] = Centrifuge( PCM , DV, DVI, par)
```

Changing function formats in the framework

The framework is adjusted to the previously shown format of inputs and outputs. If you want to add another input argument there are a few functions in the framework that need to be adjusted. For example, we want to use the following function format.

```
function [ PCM, Objective ] = Centrifuge( PCM , DV, DVI, par, Add)
```

In RMTFbuilder.m change lines 64 and 67 from:

```
R =  @(PCM,DV,DVI,par) RMF{1,1}(PCM,DV,DVI,par);
RT = @(PCM,DV,DVI,par) RMF{m,n}(R(PCM,DV,DVI,par),DV,DVI,par);
```

To:

```
R =  @(PCM,DV,DVI,par,Add) RMF{1,1}(PCM,DV,DVI,par,Add);
RT = @(PCM,DV,DVI,par,Add) RMF{m,n}(R(PCM,DV,DVI,par,Add),DV,DVI,par,Add);
```

In NLeqCONbuilder.m and NLieqCONbuilder.m the same changes have to be made, located on lines 22 & 24 and 24 & 26 respectively.

In NLCONforFmincon.m the following changes have to be made.

Change line 27 to:

```
[Tempout] = NLeqCONf{routenumber,n}(Components,DV,DVI,par, Add);
```

Change line 47 to:

```
[Tempout] = NLieqCONf{routenumber,n}(Components,DV,DVI,par, Add);
```

Function contents

Within the functions a small operation is required at the start to correctly formulate the decision variables. This operation ensures that decision variables always have the same index in the input, even when some are not used in a route. Copy/paste the following to the beginning of all unit operations:

```
for n = 1:size(DVI,2)
DVnew(DVI(n)) = DV(n);
end
DV = DVnew;
```

Step 1. Shape formulation and function combination

This step takes you through the process that converts a superstructure from paper to the format used in this framework by using a small exemplary superstructure, which is shown in figure 12.

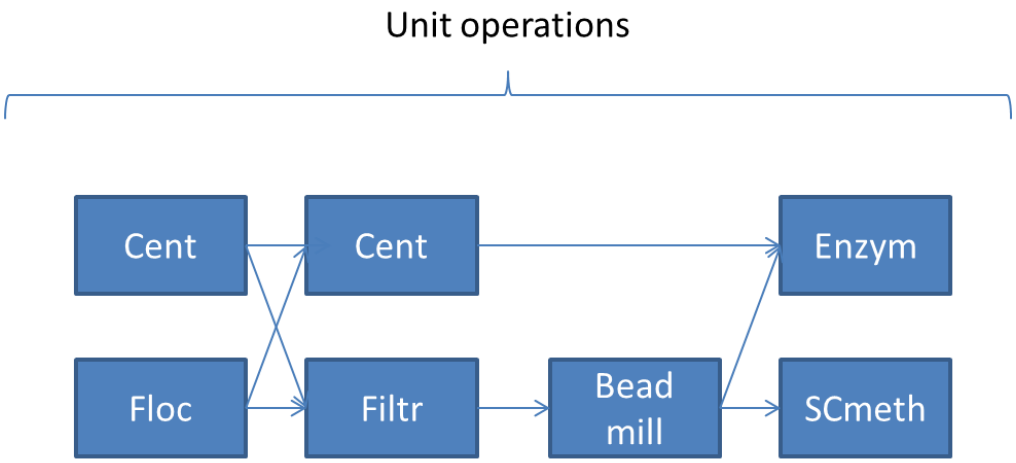

Figure 12 Exemplary superstructure

In the superstructure add a UO to the beginning and end of the superstructure then number all UOs, the result is shown in figure 13. The first one has to be number 1, the final one has to be number 0.

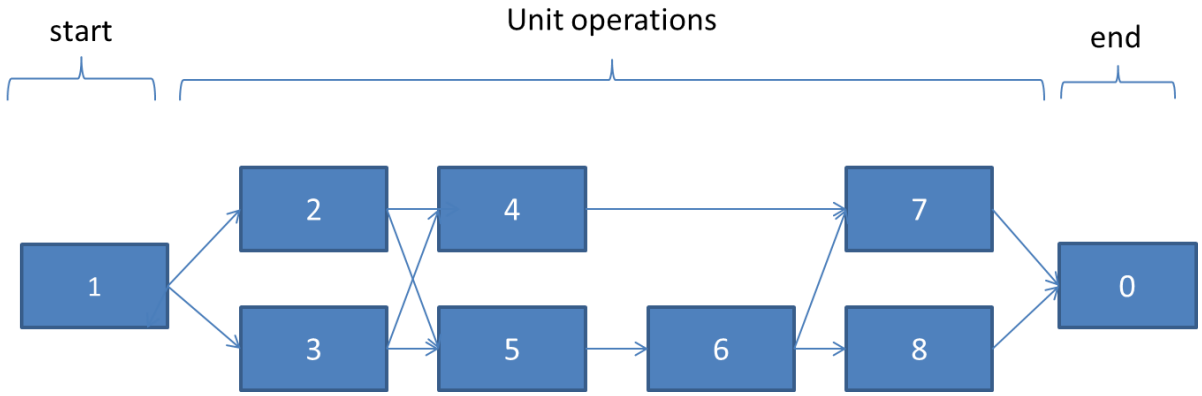

Figure 13 the superstructure after numbering and adding a function at the start and end

Now the Arrow Matrix (AM) can be produced. The arrow matrix is a n by 2 array. Where n stands for the total amount of connections between Unit Operations. Each row n describes one connection in the superstructure. The format of AM and how it is produces is illustrated in figure 14.

```
AM = [[1,2];
[1,3];
[2,4];
[2,5];
[3,4];
[3,5];
[5,6];
[4,7];
[7,0];
[6,7];
[6,8];
[8,0]];
```

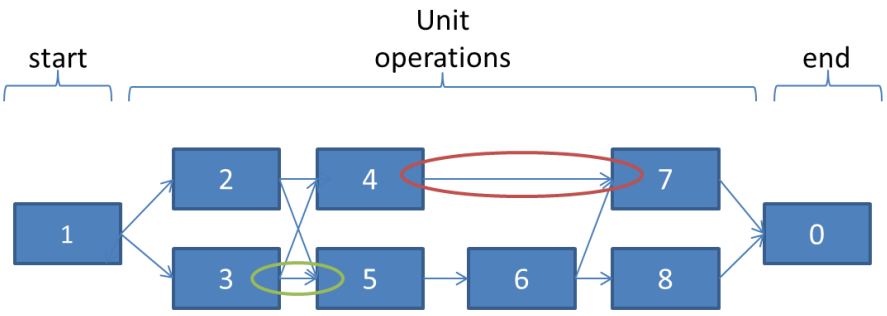

Figure 14 The format and production of the Arrow Matrix

Besides the AM, also the Function Handle Matrix (FHM) needs to be made. For the conceptual superstructure, the FHM would be the following:

```
FHM(1) = {@UOstart};
FHM(2) = {@Cent};
FHM(3) = {@Floc};
FHM(4) = {@Cent2};
FHM(5) = {@Filtr};
FHM(6) = {@Beadmill};
FHM(7) = {@Enzym};
FHM(8) = {@SCmeth};
```

Note how the row number corresponds to the given UO number. UOstart is an empty function defining the starting point of the superstructure. UO labelled with "0" is just a concept indicating the end, it does not need a function.

Run the RMTFbuilder as shown in the template script.

Step 2. Defining Decision variables

For structure and formulation of the remaining problem, it is best that the decision variables are in a fixed order and defined in a structure array. The order does not matter as long as it is consistent. This can be done in Matlab in the following manner:

```
DV =
struct('Confac',1,'Floccon',2,'Beadfill',3,'Cooling',4,'ExtracTemp',5,'MethFlo
w',6);

Order = [DV.Confac DV.ConChi DV.Beadfill DV.ExtracTemp1 DV.ExtracTemp2
DV.MethFlow];
```

Note: The actual initial values for the decision variables are defined just before optimisation.

The minimal required input to define which decision variables per route are used, is a list of which unit operation uses which decision variable. This list with Decision Variables per Unit Operation (DVUO) together with the list of which subunits are used per route, the RMF, can be combined to derive the decision variables used per route (DVPR). The DVUO is formulated as shown in table 7.

Table 7 The format for describing decision variables per unit operation, the number of the row corresponds with the number of the subunit it represents, so the order of this vector is fixed.

Unit operation (row number):	Decision variables used:
1	
2	DV 1
3	DV 1, DV 2
4	
5	
6	DV 3, DV 4
7	DV 5
8	DV 6

In Matlab this would look like:

```
DVUO{1} = [];
DVUO{2} = [DV.Confac];
DVUO{3} = [DV.Confac DV.Floccon];
DVUO{4} = [];
DVUO{5} = [];
DVUO{6} = [DV.Beadfill DV.cooling];
DVUO{7} = [DV.ExtracTemp];
DVUO{8} = [DV.MethFlow];
```

Run the DVPRbuilder as shown in the template script.

Step 3. Defining DV boundaries

Lower and Upper bounds

The required input to define which lower and upper bounds are used per route, is a list of which decision variable is subjected to which bounds. This is described in the Lower Bounds, Upper Bounds matrix (LBUB) and is formulated as shown in table 8.

Table 8 The format for describing which decision variables are subjected to which lower and upper bounds, empty bound values are substituted by - infinite and + infinite respectively.

Decision variable (row number):	Decision variables used:
DV 1	lower bound value, upper bound value
DV 2	lower bound value, upper bound value
DV 3	lower bound value, upper bound value
DV 4	lower bound value, upper bound value
DV 5	lower bound value, upper bound value
DV 6	lower bound value, upper bound value

Or in Matlab syntax for our sample superstructure:

```
LBUB(1,:) = [DV.Confac, 1,40];
LBUB(2,:) = [DV.Floccon, 0.001,0.151];
LBUB(3,:) = [DV.Beadfill, 75,85];
LBUB(4,:) = [DV.Cooling, 513,533];
LBUB(5,:) = [DV.ExtracTemp1, 308,348];
LBUB(6,:) = [DV.MethFlow, 0.01,10];
```

Run the BMbuilder as shown in the template script.

Step 4. Define the non-linear (in)equalty constrains

Create NLeqCON and NLieqCON and execute NLeqCONbuilder and NLieqCONbuilder

The input required for the non-linear constraints consists of: the subunit for which these restrictions are applicable, the position of the relevant component in the primary component matrix, the value of the boundary, whether the boundary value should be compared to the input or the output of the function, and, for inequality constraints only, whether the value should be greater or equal than (≥), or smaller or equal than (≤) the boundary value. A template for defining these values is shown in tables 9 and 10.

Table 9 Format for describing equality constraints, filled with example values.

Subunit (functionhandle)	Index of component in PCM	Boundary value	Input or output (1 or 0 respectively)
@UO1	2	80	1
@UO3	6	-80	1
@UO4	6	90	0

Table 10 Format for describing inequality constraints, filled with example values.

Subunit (functionhandle)	Index of component in PCM	Boundary value	Input or output (1 or 0 respectively)	Above or below boundary value (1 or -1 respectively)
@UO2	3	4	0	-1
@UO5	4	50	1	1

In Matlab this would look like:

```
NLeqCON(1,:) = {@Cent2, 2, 80, 1};
NLeqCON(2,:) = {@Filtr, 6, -80, 1};
NLeqCON(3,:) = {@Beadmill, 6 ,90 ,0};

NLieqCON(1,:) = {@Beadmill, 3, 4, 0, -1};
NLieqCON(2,:) = {@SCmeth, 4, 50, 1,  1};
```

Run the NLeqCONbuilder and NLieqCONbuilder as shown in the template script.

Step 5. Define additional parameters

Define all additional parameters your superstructure requires.

Superstructure optimising

Each route is analysed separately as an NLP. This is done by using a *for loop* with an increasing routenumber. In accordance with the routenumber, specific information is extracted from the RMTF, DVPR, DVIPR etc. and put in a temporary vector. In this for loop, the initial values for the components in the PCM and the initial values for the decision variables need to be defined. The outcome of the optimisation by Fmincon is rerun through the function from RMTF to gain the outcome of the objective function. These values are stored in two result matrices. All these operations are uniform and premade in the template script.

Run the premade optimisation section from the template script.

Matlab will still assign values from non-successful optimisations to the result matrices. Therefore rerun the optimisation specifically for the best few outcomes to see if indeed an optimal solution was found.