Aggregation of power-plant components for
Generis Optimization Studio

Tommi Ekholm
57641 S
Contents

1 Introduction 3

2 Background for the optimization problem 4

3 The optimization model 9

4 The aggregation of a network of components 11
   4.1 Analyzing the structure of the network .............. 13
   4.2 Assigning the operators .............................. 16
   4.3 The cartesian product operator ...................... 16
   4.4 The dimension merge operator ...................... 18
   4.5 The intersection operator ......................... 18

5 An example case 23

6 Conclusions and further improvements 25

Bibliography 27
1 Introduction

Generis (General Energy Information System) is a versatile software platform from Process Vision Ltd, intended for different players of energy markets for a variety of operations from measurement data warehousing and reporting to forecasting or balance calculations. A new application for Generis, Optimization studio, has been recently implemented, which is intended to replace the former EHTO (Electricity and Heat Optimization) software, also by Process Vision Ltd. The main purpose of Generis Optimization Studio is to allow a utility to model its production system as a linear model, and optimize its electricity and heat production against price and consumption forecasts. Time resolution can be freely changed from minutes to months, as also the time period for optimization can be vary freely from days to years.

The optimization model is structured from extreme point formulated production units, which convert input commodities into different outputs. The units are connected to each other via commodity balances, forming a production network, or a topology, for the model. The model can be run in different cases with different time resolutions, intervals or forecasts, providing a flexible platform to make different optimization runs to give the decision maker great insight for future alternatives in plant operation.

To improve the usability and flexibility of modeling and to reduce optimization calculation times, an aggregation tool was developed for Optimization studio, which would allow the user to present a partition of the whole model as a single unit. This report describes how this aggregation is calculated.

The report is constructed in a following way: the model framework is first introduced on a concept level, after which a more formal representation of the optimization problem is presented. As the framework is familiar at this point, we proceed into the description of the aggregation procedure, including analyzation of the model to be aggregated, and the calculation procedures involved. Finally a small test case is presented to present some results from the aggregation and implications on the optimization calculations.
2 Background for the optimization problem

A medium sized utility can have numerous different power plants, using different fuels with different efficiencies, producing electricity and heat for district heating with different ratios. This creates a large optimization problem, in which the decision variables specify how each plant or an individual component in a plant is operated, and the problem constraints are defined by the production possibility frontier of the whole production system.

To define the production possibility frontier, we must naturally take a look at the production system and its possibilities and limitations. A production system can be illustrated as a flowchart - as in figure 2 - which defines the topology of the production network, i.e. how different components are connected to each other. The model topology is therefore a graph containing nodes and arcs connecting the nodes.

![Flowchart of a simple production system](image)

Figure 1: An example flowchart of a simple production system containing a boiler, high and low pressure turbines, a bypass valve and a condenser.

In our formulation, the model consists of three kind of components: units, balances and accumulators. Units are generally physical production units, such as boilers, turbines or valves, which take in variable amounts of certain commodities and produce simultaneously other commodities with specified proportions from the inputs. Usually the relation between inputs and outputs can be controlled to a certain extent, creating the possibility to operate the unit in different ways, and thus forming the need for our decision variable.

Balances are instead intermediary components between two or more units,
representing the arcs in traditional flow or process charts. Their purpose is to combine flows incoming to and outgoing from the balance to the units the balance is connected to, and to exhibit a zero-divergence constraint for these flows, i.e. that the sum of incoming flows is the sum of outgoing flows. Balances transmit only a single commodity, and a physical analog would be the transfer system for that commodity, for example a steam pipe or a coal conveyor. If the physical system exhibits a leak or loss in the transfer process, this can be modeled with a unit, so that the zero-divergence holds always for balances. A balance can be accompanied by an accumulator for the commodity of the balance, which generally stores the commodity up to a specified maximum level. The flows incoming to and outgoing from accumulators are always connected to a single balance as an output and an input, respectively. The flowchart shown in figure 2 is illustrated with this formulation in figure 2.

![Flowchart](image)

Figure 2: An example topology of the simple production system from figure 2. The balances are shown with rectangles and units with circles.

In order to model the production system properly, we must naturally define the possibilities of individual units to produce and consume different commodities. A unit has a characteristic feasible area for its operation, that describes how much it can produce \( n_o \) different output commodities, and how much it consumes the \( n_i \) different input commodities. It is often reasonable to assume that the relationship with the inputs and outputs is linear, at least locally in the whole feasible area. It is also appropriate to approximate the operating area as delimited by linear constraints. Thus we end up with a linear production function in the form of a polyhedron in \( \mathbb{R}^{n_i+n_o} \), which we first assume to be convex. If the feasible area is known, as it should be to
model the unit, we can determine the extreme points \( p_i \) of the polyhedron, which we call the *characteristic points* of the unit. The set of characteristic points \( P = [p_1 \ldots p_n] \) is called simply the *characteristics* of the unit, and a simple characteristics is illustrated in figure 2. From the characteristics we can derive any point of operation \( p \) in the area as a convex combination

\[
p = P\lambda, \quad \lambda \geq 0, \quad \Sigma_i \lambda_i = 1.
\]

(1)

Here the weigth vector \( \lambda \) is the decision variable for the unit.

```
Figure 3: A basic characteristics with five points in two dimensions. The horizontal axis corresponds to the amount of input commodity, and the vertical to the amount of output commodity. The feasible area is the polyhedron between these points.
```

However, quite often the area in which we can adjust the production is inherently non-convex. Such situation could be for example a decreasing efficiency of a single-fuel boiler with increasing output levels, resulting with a concave upward sloping curve in the input-output plane. If we approximate the curve with linear segments, we have the endpoints of the segments as the characteristic points, as illustrated in 2. As the production is constrained to the curve, we can not move freely between all the characteristic points, but
only between two of them corresponding to the endpoints of a single line segment. To mark which point can be used in combination with other points, we divide the whole feasible area to convex subareas, and set a bitmask that we call an areamask for each point defining all the areas the point belongs in. Two points can be used to form a convex combination if there is a single mutual area that both the points belong in.

Figure 4: A concave two-dimensional production function approximated with line segments. The characteristic points are divided into three areas, $A_1$, $A_2$ and $A_3$.

The sinks for the generated energy in the model are energy contracts, notably for electricity and heat, which take in a limited amount of the commodity in question, and incur a negative cost to the cost function based on the tariff of the contract. Quite often contracts for heat are fixed in the manner that the producer must always satisfy the demand. Instead electricity can be freely sold to the deregulated market using the current spot price for electricity at the current area. This kind of setting defines an optimization problem to maximize the net profit from electricity sales while satisfying the
heat demand.

As the spot prices and heat demand change hourly, the problem must be solved for individual sub-models hourly. The hourly models are however interdependent, as plant operation cannot usually be raised from zero level to maximum straight away, the production units have ramp constraints which specify the rate how fast production can be increased or decreased. These dynamic constraints are however not important in this report, as the aggregation procedure cannot incorporate any dynamic constraints into the aggregated model, and thus receive no further attention in the report.
3 The optimization model

To set a more formal framework for the model, let us define the notation for the numerous elements in the model framework. This notation is also partly based on the EHTO framework, described in [1], which also gives some examples for the model framework. The set of units is noted as $U$, and a single unit is noted as $u_i \in U$. The units have I/O-terminals, a single $j^{th}$ terminal of $u_i$ noted by $t_j \in T_i$, where $T_i$ is the set of terminals for $u_i$. The set of characteristic points for $u_i$ is $P_i$, and the corresponding decision variable is $\lambda_i$, which can be interpreted as the weights of points $p_n \in P_i$. From these definitions we get the production or consumption of a commodity for unit $u_i$ into/from terminal $t_j \in T_i$ as $(P_i \lambda_i)_j$.

A single dimension of point $p_n$ thus correspond to a commodity flow, or equivalently to a I/O-terminal. In addition to these, there is a special net cost dimension for each component. The net cost represents the direct monetary cost for running the unit in each state $\lambda_i$. This does not account therefore for the fuel price of a boiler for example, but the operational costs of running it, wheather including labour, maintenance or whatever costs that seem fit for the optimization case at hand. If the unit produces incomes, for example an electricity sales contract, the net costs at its characteristic points are negative. To get a practical insight to the formulation, we can take for example a turbine which consumes high-pressure steam and converts it to low-pressure steam and electricity. The turbine unit has therefore three terminals, and four-dimensional characteristic points, with one dimension for each commodity flow through a terminal, and the cost dimension.

It is also necessary to constraint the convex combinations of the characteristic points so that there is always a single area, that all the points that have a non-zero coefficient $\lambda_n > 0$ belong to. The elements of an area mask matrix $R_i$ for unit $u_i$ are defined as $(R_i)_{r,n} = 0$ if $p_n$ is not in area $a_{i,r}$, and $(R_i)_{r,n} = 1$ if $p_n$ is in area $a_{i,r}$. As the sum of the decision variables is one,
we can write the area constraint as
\[ \exists a_{i,r} \quad \text{st.} \quad (R_i \lambda_i)_r = 1. \] (2)

The active area is not always ambiguous, as for example operating a unit in point \( p_n \) which belongs to multiple areas is definitely feasible.

The set balances are noted as \( B \) and a single balance as \( b_k \in B \). As there is no need no mark which incoming terminal is in question, we aggregate the terminals of a balance in this notation as a reference to the balance only. In the actual data model they still exist, though. Thus we have to only keep track on the direction of the arc and the terminal of the unit that the arc is connected to.

To define the model topology, \( A \) denotes the set of arcs. An arc from \( t_j \in T_i \) to \( b_k \) is \( A(t_j, k) \) and in reverse from \( b_k \) to \( t_j \) is \( A(k, t_j) \). The zero-divergence constraint for balance \( b_k \) is
\[
\sum_{\{j|A(t_j,k) \in A\}} (P_i \lambda_i)_j - \sum_{\{j|A(k,t_j) \in A\}} (P_i \lambda_i)_j = 0. \] (3)

Now we can define the whole optimization problem:
\[
\max_{\lambda} \quad P_c \lambda \\
\text{st.} \quad \sum_{\{j|A(t_j,k) \in A\}} (P_i \lambda_i)_j - \sum_{\{j|A(k,t_j) \in A\}} (P_i \lambda_i)_j = 0 \quad \forall k \\
\exists a_{i,r} \quad \text{st.} \quad (R_i \lambda_i)_r = 1 \quad \forall i \\
\Sigma_n(\lambda_i)_n = 1 \quad \forall i \\
\lambda \geq 0.
\] (4)

This kind of formulation exhibits always a certain structure, which can be utilized in creating specialized efficient LP-solvers for the problem [2].
4 The aggregation of a network of components

The purpose of the work was to define algorithms and implementation for aggregating a production system consisting of multiple units and balances to a single unit defined by a single set of characteristic points. The aggregation was seen as a beneficial tool for finding reasonable approximations for a detailed model, and to reduce calculation times of the optimization itself, as the number of constraints might be reduced significantly with lower number of balances in the model.

The first problem in defining the aggregation procedure is to analyze and identify the topology of the model. After this we can decide what procedures to take to form the aggregated characteristics. The calculation procedures were decided to be an operator system, so that during the analysis of the network we assign operation between different characteristics to several operator stacks, and finally run the operators in these stacks.

As all units are connected to each other always via a balance, we need to consider only relationships between two units both connected to a same balance. There are thus basically two different relationships between two units: either the other is connected through a forward and the other through a backward directed arc to the balance, or both are through a forward or a backward directed arc. Put more clearly: either both the units are inputs or outputs for the balance, or the other is an input and the other an output. This creates a need for two kinds of operations: to merge two characteristics across a balance, or to merge the flows from two characteristics on one side of the balance. The result from this reasoning was three different operators: an intersection which would handle the former case, and a cartesian product and a merging of dimensions, which would jointly handle the latter.

The components of the production network were arranged to levels, so that were a component in level \( l \), all components that are connected to it with forward directed arc would be on level \( l+1 \), and thus components that
are connected to it with backwards directed arc would be on level $l - 1$. As units are always connected to balances and vice versa, every other level contains only balances, and every other only units. This kind of arrangement exhibits nicely the relationships between unit described above, and we would have to only aggregate first all the units at level $l$ with the cartesian product and dimension merge, and after that continue by calculating the intersection of the characteristics between levels $l$ and $l + 2$.

All the operators are described in detail later, but to understand how the aggregation procedure runs as a whole, it is good to know already the basic ideas behind them. As said, we first want to merge all the characteristics at level $l$ into a single characteristics. We start this by forming all the possible combinations how the units at level $l$ can be run, which is done by taking a cartesian product of all characteristic points of these units. After this we merge dimensions in the aggregated level $l$ characteristics by summing all the flows the level characteristics produces for a single balance it is connected to. The rationale behind this is to present the flow to or from a single balance with a single number, and thus as a single dimension in the characteristics. As we have done these operations to the characteristics at levels $l$ and $l + 2$, we must take into account the divergence constraints of the balances at level $l + 1$. To do this we calculate the intersection of the polyhedra of characteristic points at levels $l$ and $l + 2$, but only with respect to the dimensions that correspond to the flows to or from the balances at level $l + 1$. Later we call these dimensions the common dimensions of the two characteristics. This situation is illustrated in figure 4, in a case of two characteristics with two common dimensions, meaning that there are two balances at level $l + 1$. The figure shows a projection from the characteristics to the common dimensions, and the feasible area to operate both the characteristics is indeed the intersection of these two planar polyhedra.

To show graphically the different phases of the aggregation procedure, let us take a look at a network in figure 4. The components are arranged to different levels as described before, with only units or balances at a single level.
Figure 5: Two dimensional intersection between two characteristics with four points. The dimensions in the characteristics are the input and output flows of commodity $Q$ to the balance in question.

First we calculate the cartesian products at all unit levels, and acquire a network that is shown in figure 7. Now each unit level is described with a single characteristics, which can produce multiple flows to a single balance. To sum all the flows, the dimensions corresponding to these flows are combined to a single dimension, and the resulting network after the dimension merge operation is shown in figure 8. Now we have to calculate intersections across all the balance levels, and finally acquire a single aggregated component with a single characteristics, that represents the production possibilities of the whole network. The resulting component is shown in figure 9.

4.1 Analyzing the structure of the network

The analyzation of the production network starts with a check that the network contains unconnected inputs and outputs, which will eventually be the input and output terminals of the aggregated component. If not, the whole aggregation is naturally pointless, as the model to be aggregated does not produce or consume anything.
The network to be aggregated should not contain any directed cycles, i.e., that there shouldn’t exist a path of forward directed arcs that would start and end in the same node. This requirement is the consequence of the level-
Figure 9: The resulting aggregated component after calculating the intersections across balance levels 2 and 4. The unit has two inputs and two outputs, just as the initial network had two unconnected input and output terminals.

based arrangement of the components, where always an forward arc from a componenent at level $l$ ends in a component at level $l + 1$.

The components were assigned to different levels so that all components with unconnected inputs are first inserted at level zero. The level assignment algorithm follows then every forward directed path from these components, trying to maximize the distance from the component to level zero.

However, the analyzation algorithm may find an arc in the network that connects two components on levels $l$ and $l + 1 + 2n$ with $n \geq 1$. In this case the algorithm inserts $n$ pairs of virtual balances and units as a chain to replace that arc. The virtual units have two characteristic points, $(0, 0)$ and $(M, M)$, with $M$ being very large, so that the virtual components will just carry on the flow without any loss throughout the chain. After the virtual components are inserted to the network, the requirement that all arcs connect components from levels $l$ and $l + 1$ holds.

The analyzation algorithm iterartively runs the level assignment and virtual component addition procedures. If at some iteration no virtual components are added to the network, we can be certain that all arcs are between components of adjacent levels, and thus continue to the operator assignment phase.
4.2 Assigning the operators

For each unit level $l$ with a set of units $U_l$, the first unit in that levels component list is assigned as *level main characteristics*, here noted by $\hat{P}_l$, and for every characteristics $P_i \neq \hat{P}_l$, $u_i \in U_l$ an cartesian product operator $CART(\cdot, \cdot)$ is assigned:

$$\hat{P}_l = CART(\hat{P}_l, P_1).$$

(5)

For each balance in the network, two dimension merge operators $M ERGE(\cdot)$ are assigned, one for merging all the incoming flows to, and one for merging all the outgoing flows from that balance. The merge operator acquires the indices of the dimensions to merge as the indices of the terminals in the characteristics of level $l$ that are connected to the balance in question. The operation thus changes only the main characteristics:

$$\hat{P}_l = M ERGE(\hat{P}_l).$$

(6)

An intersection $I N T E R S E C T(\cdot, \cdot)$ operator is assigned to aggregate the unit levels beside each balance-level:

$$\hat{P}_{l+1} = I N T E R S E C T(\hat{P}_{l-1}, \hat{P}_{l+1}).$$

(7)

The result of the operation is here inserted to the level main characteristic with a larger level, as the intersections are run from smaller levels towards larger levels. As all intersections are finally run in the intersection operator stack, the characteristics of the aggregated component is the main chara of the last level $\hat{P}_{max}$.

4.3 The cartesian product operator

The cartesian product is a general operation between two sets $P_1$ and $P_2$ with $n_1$ and $n_2$ elements respectively, in which all ordered pairs of $P_1$ and $P_2$ are combined to the resulting set, therefore with $n_1 n_2$ elements some of which might be degenerate. Put more explicitly

$$P_1 \times P_2 = \{(p_1, p_2) | p_1 \in P_1, p_2 \in P_2\}.$$ 

(8)
As we are trying to calculate all the possible combinations of commodity flows that the units on a single level \( l \) might produce to the balance levels on each side of the unit, we can find this by taking cartesian products among all the characteristics on that level:

\[
\hat{P}_l = \Pi_x \{ P_i | u_i \in U_l \} P_i,
\]  

(9)

where \( \Pi_x \) stands for a cartesian product. For a practical example, we could imagine two boilers \( A \) and \( B \), the production of which we wish to combine. Both the boilers consume coal and produce steam, so the result of the cartesian product from two characteristic points \( p_A \) and \( p_B \) would be \( (p_A, p_B) \) with two dimensions for coal intake and two dimensions for steam output.

At this point of the aggregation procedure we don’t yet assume the flows produced by different units would be interdependent on each other, but merely calculate the different possibilities that were able if no balance constraints would exist. The balance constraints are considered later while calculating the intersections of resulting characteristics across a balance-level, when all the unit levels have been aggregated to a single characteristics.

The complexity and the time used by the calculation of the cartesian product was not an issue, but the number of points in the resulting characteristic will be a point of interest later on. If we assume that all the units have roughly the same number of points, say \( n \), and we have \( N \) units on a level, the result characteristics will contain \( n^N \) points. Thus the number of points increases quite rapidly both with increasing number of units and the complexity of the characteristics.

The areamask for the resulting point can be simply defined: consider two resulting points \( p = [p_1, p_2] \) and \( q = [q_1, q_2] \), \( p, q \in P \). The condition for these to be in the same area is that both \( p_1 \) and \( q_1 \), and \( p_2 \) and \( q_2 \) are in the same area. We thus take a cartesian product also from the areas of charas \( P_1 \) and \( P_2 \) to form the new areas for \( P \). The areamask for the new point \( p = [p_{n1}, p_{n2}] \) can be then formed:

\[
(R)_{r_1, r_2},(p_1, p_2) = ((R)_{r_1, n1} \land (R)_{r_2, n2}.
\]  

(10)
4.4 The dimension merge operator

The cartesian product of all units on a single level $l$ results possibly with multiple terminals which are connected to a single balance $b_k$. Because we want to make a divergence constraint in the balance so that the incoming and outgoing flows match, we must form a single measure of the flow from or to a balance with each charapoint. This is simply done by summing all the dimensions that are connected to the balance with each charapoint, and presenting the result of the sum on a single dimension. Put formally

$$ (q)_j = (p)_j, \quad \forall j \in \{j|A_{t_j,k}, A_{k,t_j} \notin \mathcal{A}\} $$

$$ (q)_d = \sum_{\{t|A_{t_i,k}, A_{k,t_i}\} \in \mathcal{A}} (p)_i, $$

where the $p$ is the original and $q$ the resulting point, and the subscript $d$ denoting the dimension of $q$ to which the sum from the values of the dimensions $i$ to be merged, is inserted. As the operator does not combine the points in any way, any areamask calculations are not needed.

The merge operation is assigned twice for each balance, one for its input and one for its output side. When run for every balance the characteristics $\hat{P}_l$ is connected to, ie. all balances at the levels $l - 1$ and $l + 1$, the dimension merge operation results with one arc to each balance. Therefore we are now ready to calculate the balance constraint as the intersection of the characteristics $\hat{P}_l$ and $\hat{P}_{l\pm2}$.

4.5 The intersection operator

The intersection operator was the most challenging to implement efficiently. The problem of calculating multidimensional intersections of polyhedra has been acknowledged generally in computational geometry. Especially in the field of computer graphics the problem has really been an issue, and efficient intersection algorithms have been designed to calculate intersections of polyhedra in up to three dimensions. However, three dimensions is far below the number of dimensions the aggregation procedure would involve intersections.
in - even in a moderately simple case. General \( n \)-dimensional algorithms, for example [3], are based on half-space formulation of polyhedra, and to be able to utilize these algorithms we would have to first find the convex hull of the characteristic points to form the planar constraints, and later find again the extreme points of the intersection. These are both quite formidable operations computationally, and this resulted as a decision to construct a more simple algorithm utilizing the extreme point formulation.

To set a sufficient condition for a point \( x \) to be an extreme point under a constraint \( Ax = b \), the columns of matrix \( A \) must be arrangeable to form \([BN] \), where \( B \) is an invertible matrix, so that [4]

\[
x = \begin{bmatrix} x_B \\ x_N \end{bmatrix} = \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix}.
\]

(12)

This is directly applicable to our formulation, for which the balance constraint takes the form

\[
\tilde{P}_l \lambda = \tilde{P}_{l+2} \mu \\
\Sigma_i \lambda_i = 1 \\
\Sigma_i \mu_i = 1 \\
\lambda, \mu \geq 0,
\]

(13)

where \( \tilde{P}_l \) and \( \tilde{P}_{l+2} \) are respectively matrices containing only the rows corresponding the common dimensions of the two main characteristics from levels \( l \) and \( l + 2 \), and \( \lambda \) and \( \mu \) their decision variables. This can be arranged to form

\[
Ax = \begin{bmatrix} P_l & -P_{l+2} \\ 1.1 & 0.0 \\ 0.0 & 1.1 \end{bmatrix} \begin{bmatrix} \lambda \\ \mu \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}
\]

(14)

\[
\lambda, \mu \geq 0,
\]

to which the condition of (12) can be applied. We can therefore take \( m + 2 \) point combinations from both characteristics as the columns of the base
Figure 10: An example intersection with the boundary points given by the algorithm marked with crosses.

matrix $B$, where $m$ is the number of common dimensions, and solve the equation. If the result is feasible, ie. $x \geq 0$, it corresponds to a vertex in the intersection polyhedra in the $\lambda\mu$-space. From these decision variable we can define the amounts of flows in non-common dimensions of the characteristics at this point, that is the inflow needed at level $l + 2$ and the outflow produced at level $l$. However, not all of these points in the commodity flow space are extreme points, but are located inside or at the edge of the polyhedron, which is illustrated in figure 4.5.

To get a general idea of the complexity, we can take a look on two characteristics containing $n_1$ and $n_2$ points each, and which are connected to each other via $m$ balances, that is $m$ common dimensions. Therefore we must calculate the intersection of these characteristics in $m$ dimensions. From $n_1 + n_2$ points we can take $\binom{n_1 + n_2}{m + 2}$ combinations of $m + 2$ base points, which increases very fast as the number of points increases. A naive approach could have been to iterate all possible combinations of bases, and to solve (12) for
each of these. This would probably be sufficient in small-scale problems, but as the number of points the intersection gets from the calculation of the cartesian products is around $n^N$, we would soon run into troubles with this kind of implementation as the complexity of characteristics $n$ increases.

The general idea used in the algorithm was the same that in the simplex-method [5]. We would start with a rough guess for a basis, then move along the edges of the intersection polyhedron by replacing a single basic point at a time. Were the initial basis infeasible, we would always move to a direction that improves most the greatest violation in the constraints $\lambda_i \geq 0$, and eventually acquire a feasible basis. The difference in a simplex algorithm and the one used here, is that the simplex-algorithm will seek its way to the optimal point, but in the intersection algorithm we would have to pass through every point in the polyhedra. Thus the algorithm used two lists, one for maintaining a stack for feasible bases to be processed, and a list for bases that have been already found to be feasible. As the stack of bases to process runs empty, the algorithm terminates.

Therefore, always after updating the matrix and acquiring the coefficients $\lambda$ for the basic points, we iterate all possible replacements for a single basic point and check their feasibility against the $\lambda_i \geq 0$ requirement, the requirement for the areas of the points to match and that the new basis is not on the list of already found bases. The feasibility of removing a basic point with index $j$ in the base and addition point $i$ to $j^{th}$ base variable is tested quickly by checking the condition

$$\tilde{\lambda}_j = \frac{b_j}{a_{i,j}} \geq 0, \quad \land \quad b_k - a_{k,i}\tilde{\lambda}_j \geq 0, \quad \forall k \neq j, \quad (15)$$

but without changing the values of the matrix, as also the other possible replacements had to be checked also first.

The areamask for a combined result point is acquired by first checking the set of areas to which the points from characteristics $\tilde{P}_1$ and $\tilde{P}_{1+2}$ individually belong. After this all the combinations of areas from $\tilde{P}_1$ and $\tilde{P}_{1+2}$ are created. If $\tilde{P}_1$ has $r_1$ areas and $\tilde{P}_{1+2}$ has $r_2$ areas, the result characteristics from the
intersection has \( r_1 r_2 \) areas, and the area matrix \( R \) is defined as

\[
(R)_{i, r_1+j, n} = \begin{cases} 
1, & \text{if} \quad (R_l)_{i, n} = 1 \quad \text{and}quad 2 \quad (R_{l+2})_{j, n} = 1
\end{cases} \quad (16)
\]

\[
(R)_{i, r_1+j, n} = 1, \quad \text{otherwise},
\]

where \( R_l \) and \( R_{l+2} \) are the area matrices for characteristics \( \hat{P}_1 \) and \( \hat{P}_{1+2} \), respectively.
5 An example case

In practical use the aggregation of whole production system, which might consist of several power stations, is probably not the best way to model the system, as the aggregation loses information on how each individual component in the aggregated model is run. Instead, small portions - for example the steam supply system - of the power stations could be modeled with greater detail, and then aggregated to improve the performance of the optimization procedure. This kind of arrangement would allow the user to define more fine tuned models without making the optimization model extensively complex. However, to extract this improvement in calculation times, the test case was chosen to be a wholly aggregated system.

An simple test case was examined to get a view on the results of the calculation procedure. The production system consisted of two boilers B1, and B2, turbine T1, steam valve S1 and a condenser C1, and their connection are illustrated in figure 5. B2, S1 and C1 had linear two-point characteristics, whereas B1 and T1 had non-convex three point characteristics with two areas.

![Diagram of the example case](image)

Figure 11: The topology of the example case.

The calculation resulted with 102 points in four areas, with 57 of the points not being extreme points. Thus the aggregate characteristic contains 45 extreme points. For comparison, an aggregation model roughly twice the size and complexity resulted with over 5000 points in contrast to the 102 points with this model. A model with 23 components at 11 levels and more complicated characteristics resulted with a calculation time of several days for the aggregation routine, whereas the more simple models were formed in seconds or minutes.
The test case gave quite good results with the time taken by the optimization code. Two models were constructed, one with the aggregate component connected to four contracts, two as fuel sources, two as power sinks, and another one similar to the first, but with the five original units in place of the aggregate component. These models were run on hourly optimization case of 816 hours (34 days), with such varying input time series’ so that the optimum would change from hour to hour, and thus the result from previous model would not give the optimum straight away as an initial guess.

Ten optimization runs were run on both cases on similar conditions to reduce the effects of CPU usage by other programs. The case with the aggregated component found the optimum in 1338 iterations, using 0.685 seconds of CPU time on average with a standard deviation of 0.017 seconds. The number of iterations did not vary. In contrast the case without the component aggregation resulted in 6269 iterations and 0.802 seconds of CPU time on average with a standard deviation of 0.016 seconds. Based on this the aggregation reduced the CPU time to 85% on average with the current case. Further improvement, mainly the removal of interior points after and during the intersection calculation, could give still better results.
6 Conclusions and further improvements

This report has presented the optimization formulation used in GENERIS Optimization studio - an application developed by Process Vision Ltd, the need for an aggregation system which enables the possibility to design more detailed optimization models without complicating the optimization runs extensively, and how this aggregation procedure works. The aggregation algorithm itself is based on two phases: first the production network to be aggregated is analyzed and three kinds of operators are declared to aggregate the units pairwise. This is followed by running the operators - cartesian sums first, followed by the merging of dimensions and finally N-dimensional intersections - in the order that they were declared.

The results given by the aggregation algorithm were tested by calculating the production possibility frontiers by hand on moderately complex models, and the resulting characteristics from both of the methods were congruent with each other. To compare the improvement in calculation times for the optimization algorithm two similar cases were optimized, one model with five units and one with these five aggregated into a single unit. The aggregated model performed with 15% reduction in calculation time and 79% reduction in iteration count. Further reduction could also be acquired, were the non-corner points removed from the result characteristics.

Thus the most important improvement that should be made to the aggregation system is the removal of interior points from the result characteristics. This should be done both during the operations and after them, so that we would minimize the number of interior points that would end up in the result, but also remove those that do so. Were the intersection algorithm replaced by more efficient one, this problem might be solved simultaneously. The intersection algorithm has also some drawbacks with strongly non-convex characteristics, as the basic points are removed and replaced one at a time.

As the aggregation loses information on the flows inside the aggregated model, a deaggregation scheme could also be introduced to gain information on what happens inside the model. The problem is that the flows inside
the aggregated model are not ambiguous. We know the flows to and from the model, but this usage-production combination could probably be implemented by operating the plant in several ways. This would not be a problem if we consider the plant operation only during a single hour, as the decision maker should be indifferent about which of the operation alternatives should be chosen. However, the result from the deaggregation might also be very different for adjacent hours, which would imply huge transients in plant operation. Were the flows from the previous hour used as a starting point, this problem might be solved.
References


