District Heating Network Models for Production Planning

Henning Larsson

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
ISRN LUTFD2/TFRT--5975--SE
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Abstract

In this thesis, a methodology for aggregating district heating networks for use in dynamic optimization with the JModelica.org platform is investigated. Several methodologies for this network reduction are reviewed, and one, based on a detailed physical model of the system, is presented in greater detail. This method is implemented along with an abstract network representation and Modelica code generation to automate as much as possible of the workflow from network data to simulation and optimization models.

Experiments are presented both for aggregation for typical cases with 3–4 conusmers in a sub-network, and for a full district heating network. The tests on the smaller networks are mostly accurate, and while the results from the full network show the need for further improvement of the method, a likely source for the error is presented, along with solution proposals.

Acknowledgements

I would like to thank Modelon AB for providing both the idea for the thesis itself and the necessary resources to complete it, as well as a great working environment. A special thanks goes to my supervisors, Per-Ola Larsson and Stéphane Velut, for their help, suggestions and general expertise throughout the project.

Contents

1
Introduction

District heating plays a central role in heat delivery in urban areas, thanks to its ability to provide heating at a lower cost and with lower environmental impact than decentralized production [Frederikssen and Werner, 2014]. In order to fully take advantage of these benefits, it is desirable to operate the district heating system at optimal conditions. Given the size and complexity of a citywide system, directly calculating optimal control is computationally infeasible, and to avoid this problem an approximate model has to be introduced. In order for the model to be useful, it must accurately describe at least temperature losses, time delays and pressure drops in the network. If one or more of these fail, there is a risk of either delivering insufficient amounts of heat to the consumers in the network, or to overproduce, essentially wasting resources. Under-production is generally unacceptable, as consumers are paying for the heat delivery. Excessive production, while not as critical, is what the production plan strives to avoid in order to lower the production costs.

The focus of this thesis is on the distribution network and how the model of it can be reduced in size while maintaining reasonable accuracy, assuming that good models of production plants are available. Chapter 2 contains background on district heating systems, introduces the modeling software used, and presents an overview of currently available techniques for network approximation. In Chapter 3, the approach considered most suitable is described in detail together with details on implementation and code generation for optimization models. Chapter 4 presents optimization results obtained from the generated models, both for small cases where the optimal control for the origininal network can be calculated, and for a larger network where a reduced model is necessary. These results, along with suggested improvements and future development are discussed in Chapter 5, and the conclusions of the report are summarized in Chapter 6.

2

Background

2.1 District Heating

The purpose of a district heating system is to deliver heat to geographically distributed consumers. The heat is generated at one or a few central heating plants, the most common types of which are combined heat and power (CHP), generating electricity as well, and heat-only plants. Another possibility is to use excess heat from industries. A typical production plant has several boilers designed for different operating conditions. The base heat demand can be handled by efficient boilers with long start-up and cooldown times, as these will rarely be started or shut down. At the other end of the spectrum, peak load boilers can be started on rather short notice, but are more expensive to operate for prolonged periods of time. To further increase the robustness of heat delivery, and allow steadier levels of productions, hot water can be stored in accumulator tanks rather than sent to customers immediately. This allows "excess" production during periods with low demand, e.g. during the night, and handling of subsequent peak loads by using both the power plant and accumulator. Typically, the price of both heat and electricity will vary based on demand, adding another economic aspect to the accumulator as well, as it allows heat production at a low price and selling of the same heat at a later point when the price is higher.

Heated water from the power plant is transported through insulated underground pipes to the consumers, and a parallell network of return pipes transports the cooled down water back to the production site for reheating. An illustration of the concept is given in Fig. 2.1, and the layout of the distribution network in Uppsala, Sweden, is shown in Fig. 2.2 to give a real world example. Customers can be any kind of building with a need for heating, such as residential housing, offices or public buildings. Each one is connected to the distribution network by a heat exchanger, as shown in Fig. 2.3, and the heat transfer rate is controlled by a valve which determines the mass flow through the heat exchanger, to ensure that the temperature at the customer side is kept at a certain level. At the customer side of the loop, the transferred heat can be utilized for heating, but it is also possible that the customer-side loop is a smaller secondary distribution network, serving e.g. a small geographical area or

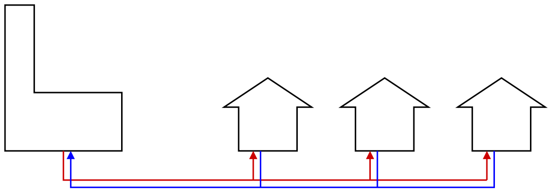


Figure 2.1 Conceptual district heating network.

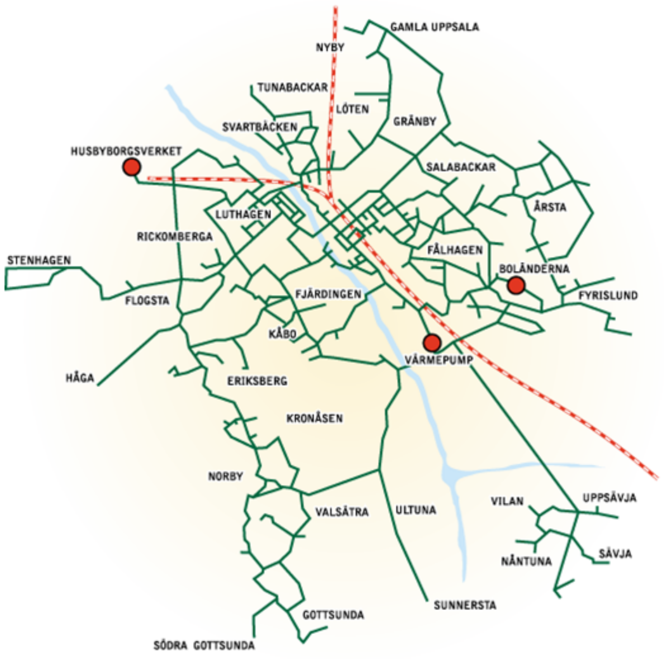


Figure 2.2 Uppsala district heating newtwork. Picture taken from [Larsson et al., 2014]

an apartment building. The actual consumers are then connected in the same way to this secondary system. For modeling purposes, these networks can generally be replaced by a single larger consumer.

2.2 Production Planning

Achieving optimal performance in a district heating system is dependent on tuning a large amount of variables, both discrete and continuous, and many of them non-

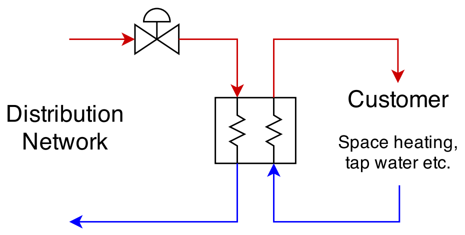


Figure 2.3 Connection between customer and distribution network.

linear. The time horizon of the production plan is also important — if the heating demand is high for a few hours, the solution is to increase output from the plant(s), but if it is expected to steadily increase for months or years, it may be necessary to invest in additional production units. Due to this, an initial partitioning into long-term, mid-term and short-term planning is done. The focus of this thesis is on the short-term planning problem, which considers the scheduling of production units, as well as optimal performance of the currently active units, on a time scale of hours to days.

Without further modification, the short-term planning is still a highly complex problem, involving both the scheduling of production units as well as deciding how these are to be run once active. The scheduling is a discrete problem (on/off variables), while optimizing the performance of an active unit is continuous. The combined problem, a mixed integer non-linear program, lacks a good solution algorithm and is instead split into a discrete and a continuous part, referred to as the Unit Commitment Problem (UCP) and Economic Dispatch Problem (EDP) respectively [Larsson et al., 2014]. These two problems can be solved individually, the UCP by mixed integer linear programming, and the EDP by dynamic optimization.

The goal of the production plan is to maximize the profit of the producers, and possibly also take other factors such as emissions into account. This is done by selecting control signal(s) u to the production plant(s). The mathematical formulation of the EDP on a time interval $[t_0, t_f]$ then becomes

$$\max_u \int_{t_0}^{t_f} (R(t) - C(t))dt$$

for functions $R(t)$ and $C(t)$ describing revenue and production cost, respectively. The production cost, in turn, is a combination of fuel and maintenance costs, along with any taxes or similar fees. For CHP plants, revenue is generated by both heat and electricity production, and these prices vary during the day, in proportion to the demand.

2.3 Dynamic Optimization

Optimal control is concerned with finding the best control trajectory to achieve a certain goal, while minimizing the cost of doing so. Mathematically, the optimal control problem for a system with dynamics described by a differential algebraic equation (DAE)

$$F(x,\dot{x},w,u) = 0$$

dependent on states x, algebraic variables w, and control variables u is written as

$$\begin{aligned}
\min_{u} \quad & J(x,\dot{x},w,u) \\
\text{s.t.} \quad & F(x,\dot{x},w,u) = 0 \\
& F_0(x(t_0),\dot{x}(t_0),w(t_0),u(t_0)) = 0 \\
& C_{end}(x(t_f),\dot{x}(t_f),w(t_f),u(t_f) \leq 0 \\
& C_i(x(t),u(t)) \leq 0 \\
& \forall t \in [t_0,t_f]
\end{aligned} \tag{2.1}$$

for some initial condition F_0, path constraints C_i and end point constraints C_{end}. The cost function J can be formulated in a general way as

$$J(x,\dot{x},w,u) = \phi(x(t_f),\dot{x}(t_f),w(t_f),u(t_f)) + \int_{t_0}^{t_f} L(x,\dot{x},w,u)dt \tag{2.2}$$

to capture both the cost of the final state and the trajectory.

As the optimization is in continuous time the problem has infinite dimension, and except in simple cases where an analytic solution is possible, it has to be reduced to finite dimension and be solved numerically. In the JModelica.org framework, this is done by discretizing the system through collocation. The method replaces F with an approximate polynomial representation, yielding a non-linear program (NLP) of finite dimension. Without going into much detail, the procedure is [Magnusson and Åkesson, 2012]

- Divide the time $[t_0,t_f]$ into n_e elements of equal length.
- In each element, select n_c collocation points. When discretizing x, the points are chosen so that both ends of the interval are included. This is not done for w and u.
- Approximate all variables by Lagrange polynomials, based on the collocation points. The extra points at the beginning of each element for x are used for equality constraints at the junction points between intervals, to ensure continuity in the approximation, something that is not done for w and u.

For this approach to work, F is required to be at least twice continuously differentiable on the entire time interval. The size of the transcribed NLP for a system with $n_z = 2n_x + n_w + n_u$ variables is on the order of $n_e \cdot n_c \cdot n_z$, i.e. it increases rapidly in size for larger systems with long optimization horizons.

2.4 Network Aggregation

For district heating applications, the amount of customers and pipes and the relatively long time scales make optimization computationally infeasible even for small networks, and a way to aggregate the network into a smaller, simpler representation is needed. Several approaches to simplification of district heating network (DHN) models exist in literature, some more specialized than others. The main differences lie in what data is needed from the original network, how complex this network can be, and which properties are conserved in aggregation.

Physical Aggregation Methods

The methods proposed in [Wigbels, 2002] (Pressure-preserving physical method), and [Larsen et al., 2002] (pressure-less physical method) utilize detailed knowledge of the network along with steady state simulation or measurement data to produce a simpler, but mostly equivalent, model. The Danish method assumes a network with a tree structure and a single production unit. This tree is first merged into a single line, with all customers preserved but with much shorter pipes. This model can then be further simplified by combining customers, removing one customer node and one pipe, while modifying two other pipes. The method is designed to produce a model which accurately describes the heat loss in the system, but does not account for pressure losses. An extension is presented in [Bøhm and Larsen, 2005], where a pressure model of the net can be created in the same way. However, the aggregation procedure is slightly different for merging branches, and the resulting heat and pressure models are thus not guaranteed to be structured in the same way.

The German method is more versatile, and presents approaches for simplification of different loop structures in the network, as well as a combined model for temperature and pressure. It is also able to handle multiple producers, although the risk for errors in dynamic simulations is larger as aggregation is done based on steady state mass flows and thus cannot account for flow reversal resulting from changed production levels.

Both methods split customers during aggregation, meaning that a record has to be kept of orignal customers corresponding to each aggregated one. The load profile of a customer does not matter to the method, and in case it changes, the aggregated net can be modified directly to reflect this, as long as the change does not affect the dynamics of the entire network.

As the aggregation is partially based on the assumption of steady state operation, models found by either method will mostly be valid around these operating conditions, although both are shown to work in dynamic simulation as well. A comparison of the two methods, on terms where the Danish method can compete, is presented in [Larsen et al., 2004]. It is concluded that both methods perform well at high degrees of aggregation, though the German method may need to produce a slightly larger network to achieve the same accuracy.

Delay Based Aggregation

Another method for aggregation is used in [Saarinen and Boman, 2012], where consumers are grouped by network delay. The delays are determined either in offline simulations or from measurement data, and are dependent on the supply temperature and flow from the plant. If several plants are present, the temperature and flow from these will also be of importance. As the number of plants increases, the amount of delay configurations will quickly grow, making the method mostly suitable for networks with a few simultaneous producers. Aggregation can be done regardless of network structure, as this is not directly taken into account.

The load profile is considered for the entire network, and does not model individual customers. If more detailed customer models are used, the demands of all customers in each delay group can be summed up to get the group demand, as no modifications of individual customers are done.

In the current state, there is no detailed modeling of the network pressure. Instead, a critical limit for the whole system is used, and the corresponding mass flow is found and used as an optimization constraint.

Conditional Parametric Models

A statistical approach to modeling the temperature is considered in [Nielsen, 2002], where "critical nodes" are represented by conditional finite impulse response models (FIR) or autoregressive models with external input (ARX) instead of physical equations. The use of FIR models is further studied in [Pinson et al., 2009], and the result is used for predictive control in [Grosswindhager et al., 2012].

The models use a transfer function from supply temperature to node temperature, where the coefficients depend on mass flow at the plant:

$$T_t^{node} = \sum_{j=1}^{p} \alpha_j(\dot{m}_{t-1}) T_{t-j}^{node} + \beta_0(\dot{m}_{t-1}) + \sum_{j=1}^{q} \beta_j(\dot{m}_{t-1}) T_{t-j}^{supply} \tag{2.3}$$

A sufficiently high order has to be chosen for the model to be able to properly account for the delay, but some of the parameters will likely not be used (e.g. recent values for nodes far from the plant). The model order is also dependent on the time resolution used. An ARX model can keep a lower order than an FIR, as the autoregressive part can explain the node temperature caused by several previous supply temperature levels. In [Nielsen, 2002], similar representations with both model types used 14 coefficients in the ARX and 31 for the FIR.

In the literature, the models are only used on networks with a single producer, and these are previously aggregated in an unspecified manner to limit the amount of modelled nodes. The models can likely be extended to allow more production units, although this can increase the complexity. Also, the number of producers will be limited to avoid excessive parametrization.

Interpolation from Simulation Results

In [Sandou et al., 2005], the optimization model is a simplification of the simulation model, without any aggregation performed. The network is made up of pipes, valves, pumps and producer/consumer nodes, modeling both heat transfer and pressure in the system. The optimization model is simplified by assuming constant pump speed and low heat loss from pipes. This results in a simpler heat loss equation, and makes the pressure difference introduced by pumps dependent only on mass flow. Standard values for mass flow and pressure are then found by simulation. During optimization, linear interpolation based on these values is used to find approximate values.

2.5 Software

Modelica

Modelica is an object oriented language for acausal modeling with differential algebraic equations. On a highly abstracted level, the code needed for a simple model (enough for the purpose of this thesis) is

```
model M
    [keyword] type name[(parameters)] [= assignment];
equation
    //component relations
end M;
```

where statements enclosed in brackets are optional. The first part of the model contains all component declarations, and everything after `equation` describes the component relations. The acausality of the language means that the order of declarations and equations are unimportant, contrary to the traditional procedural paradigm used by most common programming languages.

To give a more concrete example than the one above, consider a model of a first order low pass filter, mathematically described by the transfer function

$$y = \frac{1}{sT+1}u \tag{2.4}$$

or, equivaletly, by the differential equation

$$y' = \frac{1}{T}(u-y) \tag{2.5}$$

In Modelica code, this can be expressed much in the same way, as

```
model LowPassFilter
    parameter Real y0 = 0;
    parameter Real T = 1;
    RealInput u;
    Real y(start = y0);
equation
    der(y) = 1/T * (u - y);
end LowPassFilter;
```

This model declares the starting point y_0 and time constant T as parameters, i.e. modifiable attributes, allowing them to be changed without rewriting the model. u is declared as an input, as it does not come from the model itself. y, finally, is contained within the model, but not modifiable from the outside. The system equation is identical to (2.5), highlighting the usefulness of Modelica to model systems based on differential (algebraic) equations.

To transfer values between different models, connectors are used. These contain one or more variables to to be shared, and are used in `connect` statements, which are part of the model equations:

```
model M
    Component c1;
    Component c2;
equation
    connect(c1.connector_a, c2.connector_b);
end M;
```

This can be done to connect an input signal to the low pass filter. First, a unit step model, to use as input, is created:

```
model UnitStep
    parameter Real stepTime;
    RealOutput y;
equation
    y = if time >= stepTime then 1 else 0;
end Step;
```

The two components can now be combined in a model where the step model's output y is used as input in to the filter:

```
model LPFSimulation
    LowPassFilter filter;
    UnitStep step(stepTime=1);
equation
    connect(step.y, filter.u);
end LPFSimulation;
```

Simulating this model will evaluate the step response of the low pass filter, available as `filter.y`.

Optimica

Optimica is an extension of the Modelica language, allowing it to be used to specify dynamic optimization problems as presented by [Åkesson, 2008]. For this purpose, the `optimization` class is introduced, where optimization specific properties such as cost functions, variable bounds and constraint functions can be specified.

Continuing the low pass filter example, suppose now that a control signal is to be found to minimize the error between y and a unit step reference signal y_{ref}. The control signal is constrained to $-1 \leq u \leq 1$, giving the optimal control problem

$$\begin{aligned} \min_{u} \quad & (y(t) - y_{ref}(t))^2 \\ \text{s.t.} \quad & |u(t)| \leq 1 \\ & \forall t \in [t_0, t_f] \end{aligned} \tag{2.6}$$

In code, this can be written using the previously described components:

```
optimization LPFOpt(startTime=0,
                    finalTime=10,
                    objectiveIntegrand=cost)

    UnitStep reference(stepTime=5);
    LowPassFilter filter(u(min=-1, max=1));
    Real cost = 0;
    RealInput u;

equation
    cost = (reference.y - filter.y)^2;
    connect(u, filter.u);
end LowPassFilter;
```

JModelica.org

JModelica.org [Åkesson et al., 2010] is an open source framework for Modelica based simulation and optimization, developed by Modelon AB. It provides a Modelica and Optimica compiler, and a Python interface to simulate compiled models with the Assimulo package [Andersson et al., 2014] or, for the `optimization` class, use collocation to transfer them to an NLP and optimize them using IPOPT [Wächter and Biegler, 2006].

3

Network Modeling

3.1 Network Components

All network components used, except the production plant, are from the Modelica library DHNLibrary, implemented for [Larsson et al., 2014] with the specific intent of providing easily optimizable DHN models.

Customers

A customer is described by the following equation and constraints:

$$Q = c_p \dot{m}(T^{in} - T^{out}) \tag{3.1}$$
$$T^{in}_{min} \leq T^{in} \leq T^{in}_{max}$$

The heat demand Q is considered to be a known function $Q(t)$, and the mass flow $\dot{m}$ and supply temperature T^{in} are given by the incoming flow. As a consequence, a customer calculates its return temperature from the other variables.

The supply temperature constraints are present in the real network — the lower bound represents the lowest temperature that prevents bacterial growth in the water, and the upper bound serves to avoid excessively hot tap water at the customer and to limit maintenance costs.

Since the customer side of the heat excahnger is not modeled, there is no limit on the heat transfer. Due to this, low supply temperature and mass flow when demand is high will simply cause the return temperature to drop, possibly to only a few degrees K. In networks with customers with large differences in mass flow, this can be exploited in the optimization by giving the smaller customer(s) a lower supply temperature and mass flow, as the unrealistic return temperature will be hidden when the flows are mixed prior to returning to the plant and thus not influence the supply temperature noticeably. To prevent this, without having to create a more complex customer model, a constraint on the return temperature is added:

$$T^{out}_{min} \leq T^{out}$$

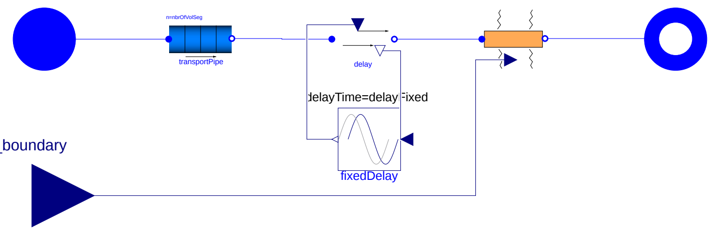


Figure 3.1 Dymola representation of simulation pipe model.

A reasonable value for this bound is the lowest expected return temperature from a customer or slightly below, typically around 30–50°C.

Unlike the customers described in Section 2.1, each customer in the network model is required to have its own pump. This is due to the fact that the modeling library used bases its calculations on mass flows rather than pressure differences. In the model, the individual customer pumps determine the flow through each customer, whereas in a real network, these flows are caused by a higher pressure in the supply side of the network. As long as the maximal flow rate through the pump and the valve are the same, the result is the same in terms of flow.

Pipes

The pipe model presented in [Larsson et al., 2014], which combines a finite volume pipe model with a fixed delay and a heat loss, is used. The setup is shown in Fig. 3.1. A finite volume pipe alone has the problem that it requires a high number of elements in order to be accurate, while the optimization requires few elements for computational efficiency. A static delay, on the other hand, is only correct for a single mass flow, as the delay decreases with mass flow. If, however, the mass flow is known to be within a certain range, a static delay combined with a low order finite volume pipe can provide a good approximation of the delay dynamics.

The time delay is handled slightly differently in optimization and simulation. In simulation, the delay can have any value, but in optimization it is limited to multiples of the interval length used in discretization (see Section 2.3). While this introduces additional error in the optimization model, the effect can be kept small by using a high enough resolution in the discretization.

Producers

A power plant is a complex process, the modeling of which is outside the scope of this thesis. For the purpose of testing aggregated models, it is sufficient to introduce

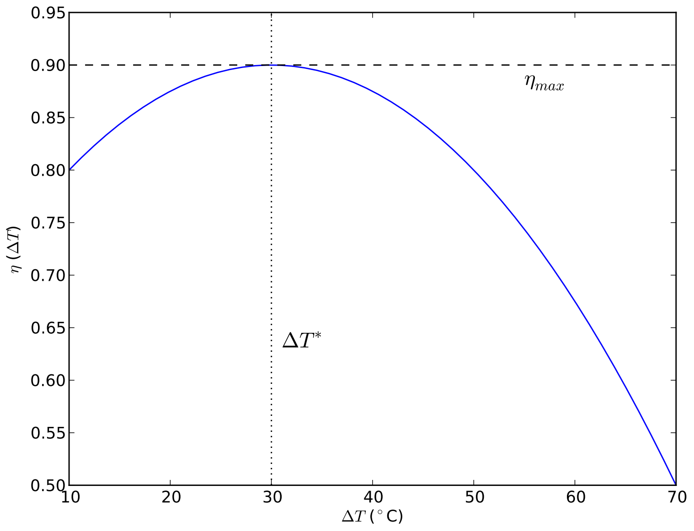


Figure 3.2 Producer efficiency for different ΔT, with $\alpha = 2.5 \cdot 10^{-4}\ K^{-2}$ and $\eta_{max} = 0.9$.

a simple model which delivers heat according to

$$\begin{gathered} Q(t) = Q_{max}u(t) \\ 0 \leq u(t) \leq 1 \end{gathered} \tag{3.2}$$

given some control signal $u(t)$. This is combined with the heat transfer equation Eq. (3.1) and an efficiency model as

$$\begin{gathered} \eta(\Delta T)Q_{max}u(t) = c_p\dot{m}(t)\Delta T(t) \\ \eta_{min} \leq \eta(\Delta T) \leq \eta_{max} \end{gathered} \tag{3.3}$$

where the efficiency is calculated from an ideal increase in temperature ΔT^* as (see Fig. 3.2)

$$\eta(\Delta T) = \eta_{max} - \alpha(\Delta T(t) - \Delta T^*)^2, \quad \alpha > 0 \tag{3.4}$$

The added efficiency gives a fully determined system by forcing ΔT to a certain value. If this is not done, any combination of ΔT and $\dot{m}$ is equally valid, causing infinitely many solutions to the optimization.

Pumps

The models used are flow driven, meaning that the mass flows are determined directly from flow sources (such as pumps), rather than from pressure differences. To fully determine the mass flows throughout the network, each customer gets its own pump, and the mass flows closer to the producer are sums of the pump flows beyond that point.

All pumps are modeled as ideal, i.e. there is no heat loss or storage, and no loss of flow. The mass flow through the pump is determined by a control signal $u(t)$, such that

$$\dot{m}(t) = u(t)\dot{m}_{max} \qquad (3.5)$$
$$0 \leq u(t) \leq 1$$

Volumes

Volumes contain mass and energy balances, calculated from mass flow and enthalpy. They are not actual network components, but are used in modeling to keep track of these balances.

Temperature Model

The outdoor temperature is used to calculate heat losses in the network. Temperature is modeled by the equation

$$T(t) = T_0 + (T_{peak} - T_0)e^{-\frac{1}{c}(t-t_1)^2} \qquad (3.6)$$

i.e. as a base temperaure T_0, rising to T_{peak} during the day and then falling back down. The peak is reached at time t_1, and the sharpness of the increase and decrease is determined by c. An example trajectory for the outdoor temperature, with explanations for the parameters, is displayed in Fig. 3.3.

Heat Demand Model

A deterministic load model is used, where the heat demand for a day is considered to be a base load with two peaks in the demand. The heat demand Q_d is described by the equation

$$Q_d(t) = Q_0 + Q_1 e^{-\frac{1}{c}(t-t_1)^2} + Q_2 e^{-\frac{1}{c}(t-t_2)^2} \qquad (3.7)$$

Q_0 is the base load, and peak demands of $Q_0 + Q_1$ and $Q_0 + Q_2$ W occur at times t_1 and t_2. c is a smoothing parameter, determining the sharpness of the peaks. An example of a heat demand is shown in Fig. 3.4, along with explanations of the parameters. The demand curve is typical for residential heating, with the peaks occuring in the morning and evening.

3.2 Aggregation Method Selection

The approaches to aggregation covered in Section 2.4 all have different strengths and weaknesses, depending on application. For the purpose of production planning, it is desirable that the method used should have the following traits:

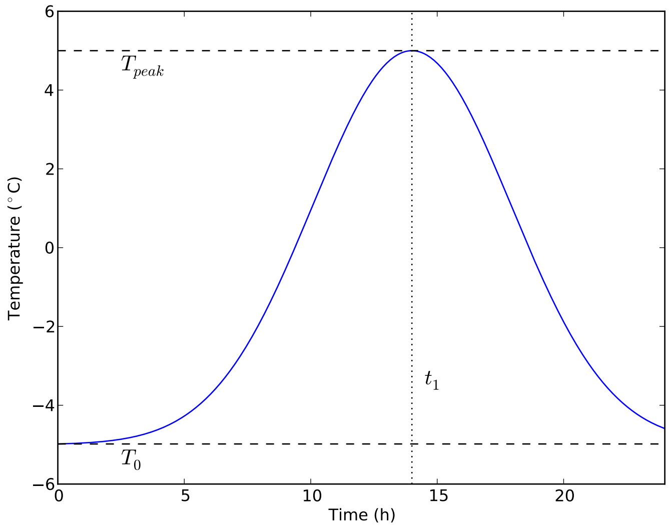


Figure 3.3 Example of outdoor temperature during 24 hours.

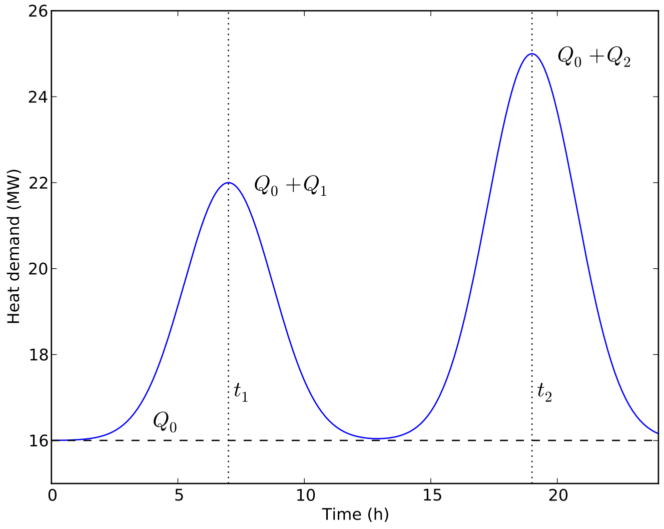


Figure 3.4 Example heat demand for 24 hours.

- As the previous work presented in [Larsson et al., 2014] has been is focused on physics based modeling, the aggregated network should either be physics based in itself, or easily transferrable to such a setting.
- Both heat and pressure losses in the network are accounted for.
- Different types of customers, with different demand models, can be handled within the same model.

To motivate the choice of method, a brief overview of the different methods and their respective pros and cons is presented below.

Pressure-less physical method A physics based approach, the pressure-less physical method is capable of reducing network size greatly. Individual customers can be traced through the aggregation, allowing the use of different demand models at all network sizes. The method has two main drawbacks: it cannot handle heat and pressure losses within the same model, and it is only able to aggregate networks without loops.

Pressure-preserving physical method Mostly similar to the pressurelless method, it has the additional strengths that it includes both heat and pressure losses in the same model, and that it, at least theoretically, can handle any network structure. The high level of detail means that a great amount of data needs to be available for any network considered for aggregation.

Conditional parametric models These models are based on network statistics rather than physical properties, and the resulting polynomial model is computationally simpler than a physics-based one. The main flaw with this approach is that only a few points in the network are modeled. This means that aggregation is done by hand, in the selection of these locations.

Delay-based aggregation Aggregating by the delay from producer to each customer is a simple procedure, and the method is also independent of network structure. However, it does not describe pressure losses fully, and considers heat loss only as a function of the delay, and not e.g. different types of pipes.

Simplification by interpolation This approach does not perform any aggregation at all, but instead simplifies the model. While this preserves the original structure of the network, it does so at the expense of fixing all mass flows. This removes a degree of freedom in the production planning problem, unnecessarily restricting it.

Taking all this into account, the pressure-preserving physical method fits the requirements best, and will thus be the method used in this thesis.

3.3 Aggregation Procedure

The network structure is simplified based on steady state operating conditions, using detailed information on network layout and physical characteristics (temperatures, pressure differences, pipe dimensions etc.). In the considered production planning setting, this steady state will be based on the results of the UCP, but more generally, values could be found e.g. by averaging measurement data. Three kinds of modifications of different structures are applied repeatedly until the desired network size has been reached. These are

- Removing the middle customer of three serially connected ones
- Merging a branch in the network into a serial connection
- Splitting a customer with multiple incoming flows into separate customers, each with only one input

These scenarios are derived and presented, along with some extensions, in detail in [Loewen, 2001; Wigbels, 2002]. For each considered subnet, the following requirements are imposed on the aggregated result, to maintain equivalence between the two networks:

1. The mass flows into and out of the aggregated net remain constant
2. The total pipe volume remains constant
3. All time delays in the subnet remain constant
4. The temperatures at the subnet boundaries remain constant
5. The pressures at the subnet boundaries remain constant
6. The sum of consumer mass flows remains constant
7. The sum of consumer heat loads remains constant
8. The total heat loss in the subnet remains constant

When these requirements are fulfilled, the original and aggregated subnets interact identically with the rest of the network under the assumed steady state conditions.

In all following discussion of the method, the notation convention will be to index original network components by integers as C_1, C_2 etc, and aggregated components by letters C_A, C_B... Supply and return pipes are denoted as P_i and $P_{i,R}$ respectively.

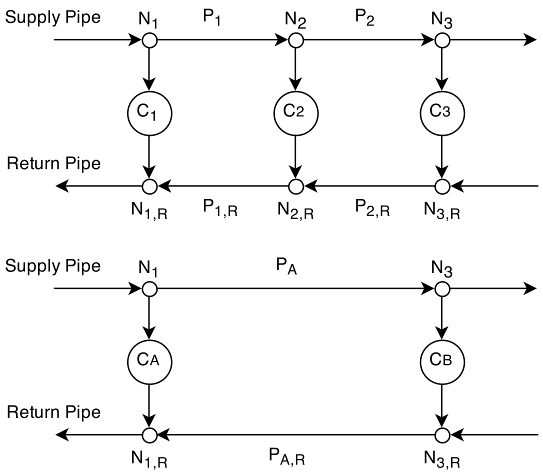


Figure 3.5 Aggregation of serial subnet.

Serial Pipes

Three serially connected customers can be reduced to two by splitting the middle customer, as shown in Fig. 3.5. A pipe connecting the resulting customers C_A and C_B is created to fulfill the specified requirements. Beginning with the requirement that the total pipe volume remains constant, the volume of the aggregated pipe becomes

$$V_{P_A} = V_{P_1} + V_{P_2} \tag{3.8}$$

The pipe geometry can be chosen in any way that gives this volume, but it is reasonable to assume that the length sums up in the same way

$$L_{P_A} = L_{P_1} + L_{P_2} \tag{3.9}$$

which makes calculation of the radius or diameter straightforward. With the volume decided, the mass flow in the aggregated pipe follows from requirement 3.

$$\dot{m}_{P_A} = \frac{V_{P_A}}{\frac{V_{P_1}}{\dot{m}_{P_1}} + \frac{V_{P_2}}{\dot{m}_{P_2}}} \tag{3.10}$$

Knowing the mass flow through the aggregated pipe, the flows through the aggregated customers are found as

$$\dot{m}_{C_A} = \dot{m}_{C_1} + \dot{m}_{P_1} - \dot{m}_{P_A} \tag{3.11}$$

$$\dot{m}_{C_B} = \dot{m}_{C_3} + \dot{m}_{P_A} - \dot{m}_{P_2} \tag{3.12}$$

Viewed in another way, this represents the mass flow of C_2 being split in two as

$$\dot{m}_{C_A} = \dot{m}_{C_1} + k\dot{m}_{C_2} \tag{3.13}$$
$$\dot{m}_{C_B} = \dot{m}_{C_3} + (1-k)\dot{m}_{C_2} \tag{3.14}$$

with a split factor

$$k = \frac{\dot{m}_{P_1} - \dot{m}_{P_A}}{\dot{m}_{C_2}} \tag{3.15}$$

Since the boundary temperatures of the subnet remain the same, the inlet temperature of C_B is known, while the return temperature can be found by calculating the energy balance at the return node:

$$T_{C_B}^{in} = T_{C_3}^{in} \tag{3.16}$$
$$T_{C_B}^{out} = T_{N_{3,R}} + \frac{T_{N_{3,R}} \sum_{i=1}^{N} \dot{m}_{i,in} - \sum_{i=1}^{N} \dot{m}_{i,in} \cdot T_{i,in}}{\dot{m}_{C_B}} \tag{3.17}$$

The load Q_{C_B} can be calculated from (3.1), and Q_{C_A} is then the rest of the subnet load

$$Q_{C_B} = c_p \dot{m}_{C_B} \Delta T_{C_B} \tag{3.18}$$
$$Q_{C_A} = Q_{C_1} + Q_{C_2} + Q_{C_3} - Q_{C_B} \tag{3.19}$$

As with C_B, the inlet temperature of C_A is required to be the same as in the original subnet. Given the load and mass flow, the updated return temperature can be found as

$$T_{C_A}^{out} = T_{C_A}^{in} - \frac{Q_{C_A}}{c_p \dot{m}_{C_A}} \tag{3.20}$$

Once all temperatures are known, and given the surrounding temperature T^{env}, the heat loss coefficients for the new supply and return pipes can be calculated. These are derived from the heat loss model

$$T^{out} = T^{in} + (T^{env} - T^{in}) \exp\left(-\frac{\pi \lambda_{P_A} L_{P_A} d_{P_A}}{c_p \dot{m}_{P_A}}\right) \tag{3.21}$$

which is rearranged to find λ_{P_A} as

$$\lambda_{P_A} = \frac{c_p \dot{m}_{P_A}}{\pi L_{P_A} d_{P_A}} \ln\left(\frac{T_{P_A}^{in} - T^{env}}{T_{P_A}^{out} - T^{env}}\right) \tag{3.22}$$

An issue not covered in the original paper is the handling of variable constraints in aggregation. To avoid constraint violations, the tightest bound from the aggregated customers must be kept. As a motivating example, consider the upper bound on mass flows for two customers C_1 and C_2. Let

$$\dot{m}_{C_1} = 1, \quad \dot{m}_{C_1,max} = 10$$
$$\dot{m}_{C_2} = 9, \quad \dot{m}_{C_2,max} = 10$$

If the two are fully combined and the mass flow constraint is updated in the same way as the mass flow, the resulting customer C_A will have a nominal flow of 10 kg/s and a max flow of 20 kg/s. However, as C_2 is already at 90% capacity, C_2's mass flow constraint will be implicitly broken if

$$\dot{m}_{C_A} > \frac{\dot{m}_{C_1,max}}{0.9} = 11.11\dots$$

which is less than $\dot{m}_{C_A,max}$, and thus a viable result in the optimization.

The problem can be adressed by calculating the tightest variable bound in each aggregation step, to ensure that no constraint is violated. The upper bounds can be calculated as in the example above, and the lowest of these is chosen as the flow constraint of the aggregated customer. The modifications are made in the serial case, where the upper bounds become

$$\dot{m}_{C_A,max} = \dot{m}_{C_A} \cdot \min\left(\frac{\dot{m}_{C_1,max}}{\dot{m}_{C_1}}, \frac{\dot{m}_{C_2,max}}{\dot{m}_{C_2}}\right) \tag{3.23}$$
$$\dot{m}_{C_B,max} = \dot{m}_{C_B} \cdot \min\left(\frac{\dot{m}_{C_2,max}}{\dot{m}_{C_2}}, \frac{\dot{m}_{C_3,max}}{\dot{m}_{C_3}}\right) \tag{3.24}$$

In the same way, temperature and pressure difference constraints in aggregated customers should be chosen as the tightest bound, if not all customers have the same requirements on in- and outlet temperatures.

This way of modifying the variable constraints ensure that all customers are guaranteed to have their heat demand fulfilled by the optimization result (if the model error is assumed to be small enough). Looser bounds on the mass flow through aggregated customers may lead to solutions where some customers require a higher flow than what is possible, thus not receiving enough heat to satisfy their demand.

Branches

Branches consisting of a single node along a serially connected subnet (see Fig. 3.6) can be merged into that subnet. In order to get a proper structure with C_2 before C_3, an additional constraint on the time delays in the subnet has to be added:

$$\tau_{C_1,C_2} < \tau_{C_1,C_3}$$

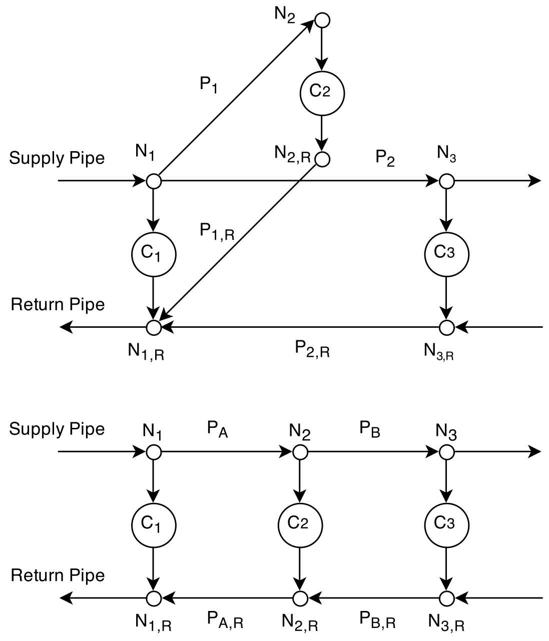


Figure 3.6 Aggregation of branch subnet.

The to maintain the delays between the different customers, the delays in the aggregated pipes must be

$$\tau_{P_A} = \tau_{P_1} \tag{3.25}$$

$$\tau_{P_B} = \tau_{P_2} - \tau P_1 \tag{3.26}$$

Calculations for the resulting pipes are derived in the same way as for serial subnet aggregation, resulting in

$$\dot{m}_{P_A} = \dot{m}_{P_1} + \dot{m}_{P_2} \tag{3.27}$$

$$\dot{m}_{P_B} = \dot{m}_{P_2} \tag{3.28}$$

$$V_{P_A} = V_{P_1}\left(1 + \frac{\dot{m}_{P_2}}{\dot{m}_{P_1}}\right) \tag{3.29}$$

$$V_{P_B} = V_{P_2} - V_{P_1}\frac{\dot{m}_{P_2}}{\dot{m}_{P_1}} \tag{3.30}$$

$$L_{P_A} = L_{P_2}\frac{\tau_{P_1}}{\tau_{P_2}} \tag{3.31}$$

$$L_{P_B} = L_{P_2} - L_{P_A} \tag{3.32}$$

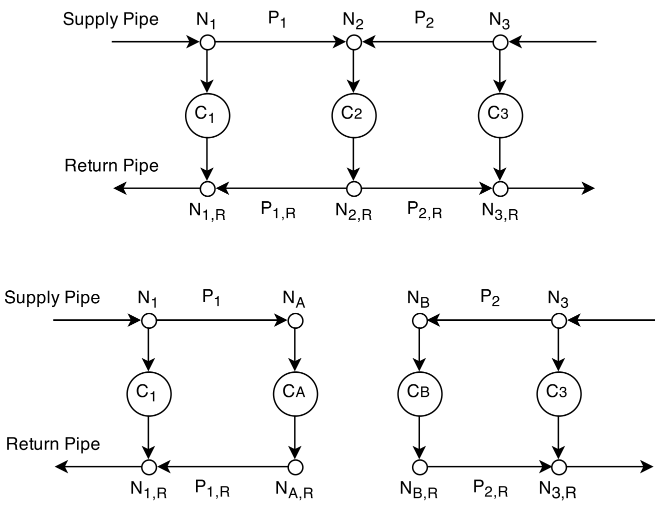


Figure 3.7 Splitting of a customer with multiple input flows.

Aggregating a branch subnet does not modify the customers, meaning that the supply and return temperatures remain the same. Because of this, the adjusted heat transfer coefficients for the aggregated pipes are found as in the serial case.

Another consequence of not modifying the customer is that branch aggregation does not significantly reduce the complexity of the optimization problem. What it does is to simplify the graph, in order to allow aggregation of the resulting serial customer.

Theoretically, a branch can be merged into a longer serial subnet, by placing the customer appropriately based on delay, but this operation is unnecessary, as the serial net can be aggregated instead to achieve a structure fulfilling the delay constraint.

Mixing flows

In the case of a customer with two input flows, the customer can be split into two smaller ones, each being supplied only by one input. The method could also be extended to split a customer with any number of inputs into the same number of separate customers, if such a situation should arise.

This structure appears both in loops in the network and in networks with multiple production sites. In loops, the flows come from the same supply pipe, which has been split at an earlier point, while in a scenario with multiple producers, the flows are from different production sites. Both cases can be present in a single network.

The mass flow split (in a case with two supply pipes) is given as

$$\dot{m}_{C_A} = \frac{\dot{m}_{P_1}}{\dot{m}_{P_1} + \dot{m}_{P_2}} \dot{m}_{C_2} \tag{3.33}$$

$$\dot{m}_{C_B} = \frac{\dot{m}_{P_2}}{\dot{m}_{P_1} + \dot{m}_{P_2}} \dot{m}_{C_2} \tag{3.34}$$

and the customer loads are split in the same way. The supply temperatures of the aggregated customers become the temperature delivered from their respective supply pipes, instead of the mixed temperature. No modifications are done to the pipes.

It is also possible to split customers downstream of C_2, if they exist, but to limit the amount of new components introduced by this split, it is preferrable to aggregate them into a single customer first. This customer, along with the pipe connecting it to C_2, both have their mass flows split by the same ratio as C_2.

As with branch aggregation, the model complexity is not reduced by splitting flows, but the split subnet can be reduced further by branch and serial aggregation.

Other cases

There is an alternative way to remove loops, where the loop is flattened into a single serial structure, maintaining all customers and their connections to the rest of the network. The endpoint of the resulting serial pipe is where the flows meet. While this may be a feasible approach to reduce loops, there is little it can do in a case with multiple production sites, and thus it has been left out.

It is also possible to extend the aggregation to subnets where the supply and return pipes are not laid out in parallell. As the vast majority of distribution networks are structured in the ordinary way, this, too, was considered unnecessary to implement.

3.4 Graph Representation

To represent the network layout, a representation as a directed graph is suitable. A directed graph $G : (E, V)$ is a set of edges E and vertices V (also known as nodes), where an edge $e_{ij} \in E$ goes from vertex i to j. Some definitions which will be useful to describe the aggregation procedure are:

- The *in-degree* of a vertex is defined as the number of incoming edges, and the *out-degree* is, similarly, the number of outgoing edges.
- In a directed graph, the *successors* of a vertex V are the vertices reached by following the outgoing edges, and the *predecessors* are the vertices to which V itself is a successor.
- A *root* is vertex which has no predecessors. A graph may contain multiple roots.

A district heating network can be considered as a directed graph with customers and production sites as vertices, and the pipes connecting them as edges. The direction of the edges is given by the direction of flow in the pipes. Technically, this gives rise to two different graphs for the supply and return sides of the network, but as the pipes are laid out side by side, either of the graphs can be obtained by simply reversing all edges of the other. Because of this, only the supply network will be considered as all aggregation is based on vertices, and thus affects the two graphs in the same way.

Each subnet type considered in aggregation can be defined as a subgraph $G_s \in G$ with some restrictions on the vertices included.

- In a serial subnet, all nodes except the first and last are required to have both in- and out-degree 1.
- Meeting flows are found at vertices with in-degree > 1. Vertices which are eligible to be split are also required to have at most one successor.
- A branch subnet is composed of a root v_1 (with respect to G_s), and two successors v_2 and v_3, where v_2 has out-degree 0 and in degree 1, and the delay from v_1 to v_2 is lower than from v_1 to v_3.

Implementation

The network graph implementation uses the NetworkX package for Python [Hagberg et al., 2008], and is based on the generic directed graph `networkx.DiGraph`. The network representation is bundled with code generation (described below) into a Python package named `dhn`, intended to provide the necessary functionality for transferring raw network data to an optimizable model with as much automation as possible, while still maintaining generality.

This package provides two classes for network representation, `Network` and `DemoNetwork`. `Network` provides all basic functionality, and `DemoNetwork` extends this class to provide visualization, without adding any further features. To the end user, they both provide the following methods:

add_customer Adds a customer to the network.

add_producer Adds a producer node to the network. Unlike customers, these are excluded from aggregation.

customers Returns a list of all customers. This is useful as the inherited method `DiGraph.nodes` will return both customers and producers.

producers Returns a list of all producers.

draw Plots the network. Assumes that all nodes have a position parameter.

aggregate Reduces the network to a specified number of nodes, or as much as possible if this number cannot be reached.

Aggregation of a network is done, as illustrated in Fig. 3.8, by iteratively aggregating first serial subnets and then branches until either a specified amount of nodes remain in the graph, or no more aggregation can be performed. No splitting of customers is done, due to modeling errors further discussed Section 5.1, which somewhat limits the ability to aggregate structurally complex networks. If this operation is needed, it can either be done by hand as it is less common than the other two, or quite easily added to the implementation.

A greedy algorithm is used to locate all subnets, which assumes that there is no preferable order in which to aggregate customers. This might not be the optimal approach, but it is the simplest one, and as such it is a good starting point.

NetworkX.DiGraph The `DiGraph` class provides a greneral directed graph implementation. All nodes and edges are accessible as through a Python dictionary, for example

```
from networkx import DiGraph

g = DiGraph()
g.add_node(1)   # node is named 1
g.node[1]['attribute'] = 1
print g.node[1]['attribute']
```

Similarly, edges are defined by the two vertices they connect. If one or both of the specified nodes do not exist when adding an edge, they will be automatically created

```
g.add_edge(1,2)      # implicitly create node 2
g.edge[1][2]['attribute'] = 'some attribute'
```

This functionality in itself is sufficient to represent a district heating network, but it is extended by `Network` to provide a more convenient and intuitive interface. Prime examples of this are `Network.add_node` and `Network.add_customer`, which contain little more than a call to `DiGraph.add_node` but still make the network declaration more readable and eliminates some repetition of code.

Locating Serial Subnets The method for finding serial subnets takes into account both the limitations on network layout presented above, and additionally ensures that the returned net does not contain more customers than should be aggregated. The procedure is divided into two steps. Firstly, a single serial subnet can be found given a starting point with a neighbour with in degree 1 and out degree 1.

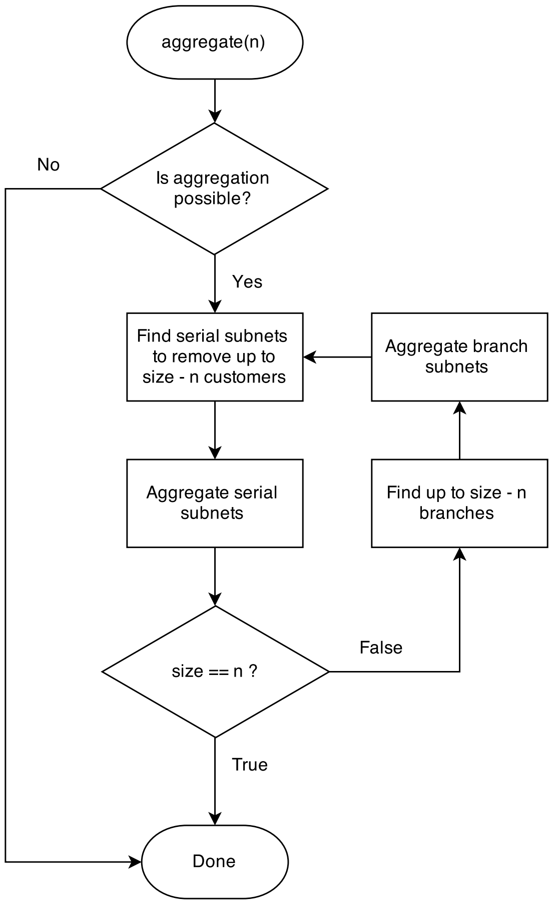


Figure 3.8 Overview of the aggregation algorithm.

```
get_serial_subnet
    input:    start - valid starting point
              next  - neighbour of start
              n     - the maximal number of customers to remove
    output:   serial subnet of at most n customers, from 'start'

    last <- neighbour of neigbour
    subnet <- list(start, neighbour, last)
    n <- n - 1
    while out_degree(last) == 1 and n > 0:
        add last to subnet
        last <- neighbour of last
        n <- n - 1

    return subnet
```

Once this method for retrieving subnets has been established, it can be applied to all valid starting points within the network.

```
serial_subnets
    input:    n - the maximal number of customers to remove
    output:   list of serial subnets in the graph, with at
              most n customers possible to aggregate

    subnets <- empty list
    for each customer in network:
        if customer is possible serial start:
            possible <- neighbours of customer with
                        in_degree == out_degree == 1

            for p in possible:
                s <- get_serial_subnet(customer, p, n)
                append s to subnets
                n <- n - length(subnet) - 2
                if n <= 0:
                    return subnets

    return subnets
```

These two operations combined will either return the requested number of customers to aggregate, or all available serially connected customers if there are not enough of them.

Locating Branch Subnets Branch subnets are of known size (three nodes), but have the limitation that the leaf node, which is to be merged into a serial configuration, must have a lower delay than the end node in that subnet. Finding such a structure is done as

```
get_branch_subnet
    input:  root - root node of the branch
    output: a list of the nodes in the branch, if a
            branch exists, otherwise an empty list

    branch <- empty list
    leaves <- empty list
    append root to branch

    for each neighbour of root:
        if in_degree(neighbour) == 1 and out_degree(neigbour) == 0:
            insert n into leaves

    sort leaves by delay (lowest first)
    min_delay <- lowest delay
    append leaf with lowest delay to branch

    for each remaining neighbour of root:
        if delay from neighbour to root > min_delay:
            append neigbour to branch
            return branch

    return empty list
```

This algorithm can be iteratively applied to all customers in the network, returning either the requested number of branches (as one customer per branch is made available for further aggregation), or all available branches. The iteration is simlilar to the one for serial subnets:

```
branch_subnets
    input:    n - the maximal number of branches to find
    output:   a list of branch subnets

    subnets <- empty list
    for each customer in network:
        s <- get_branch_subnet(customer)
        if s is not empty:
            append s to subnets
            n <- n - 1

            if n == 0:
                return subnets

    return subnets
```

3.5 Modelica Code Generation

Generating complete or almost complete models from the graph representation minimizes the manual labour involved in the creation of optimization and simulation models, as the network only has to be specified once, when the full network graph is created. For this purpose, `dhn` provides a hierarchy of translator classes, shown in Fig. 3.9. Although the current implementation only provides code generation for models using DHNLibrary, the aim is to provide easy extensibility to use other Modelica libraries as needed.

Translation Concepts

In a general setting, the components making up the network are considered in the same way as they are in the graph. This means that at the highest level, translation can be divided into code generation for vertices and edges, where the vertices can be subdivided into producers and customers. Furthermore, any one-off global components needed in a model (but not in the graph) can be added separately. While translation of these components will be library specific, it will always require a component declaration, as well as connections to other components in the model. For the purpose of code generation, the following two types of declarations are sufficient

```
    type name[(parameters)];
    type name = assignment;
```

Different models can be linked to each other using `connect` statements, which use special components known as connectors. These contain model values which are transferred to connected components, such as heat transfer and flow rates. Generating the component declarations for a network model is straightforward, given a

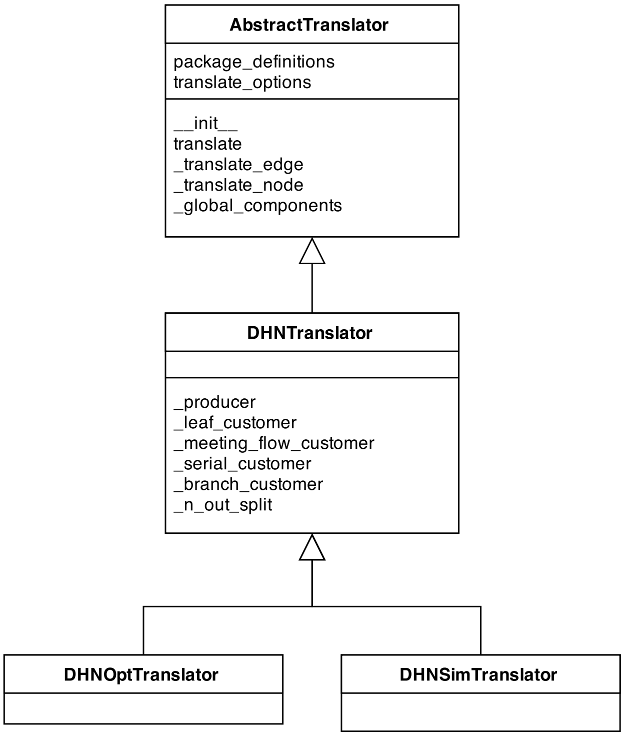


Figure 3.9 Class hierarchy for the graph-to-modelica translation.

naming scheme and the library specific types and parameters, and each component can be declared independently. The problem is further simplified by the acausal nature of Modelica, as this means that the ordering of declarations is unimportant. Connections require a bit more care, as they depend on other components and thus have to know the names of these. Even though it would be possible to let all components calculate each other's names, a cleaner solution is to keep the naming scheme in one place and instead store component connectors in a table indexed by name and connector type. In this way, all vertices can be processed first, and when subsequently going through the edges, the appropriate connections can be retrieved. The connectors used by customers and producers are *supply in* and *return in* for incoming flows from these lines, and *supply out* and *return out* for the outgoing. Any other connector which needs to be known in the translation of additional graph elements could also be listed in the connector table as appropriate. The translation procedure can then be abstracted as

```
translate
    input:    network - network graph to translate
    output:   Modelica code for network model

    declarations <- empty list
    connections <- empty list
    connector_table <- empty table

    for node in network:
        translate node

    for edge in network:
        translate edge

    write declarations to file
    write connections to file
```

All of this functionality is implemented in `AbstractTranslator`, which can be extended by translator implementations for specific libraries. Extending Abstract-Translator requires the subclass to implement two methods `_translate_node` and `_translate_edge`, which are called for code generation. These methods are assumed to add any connectors to the connector table, as well as add declarations and connect statements to the respective lists. It is also possible to override the `_global_components` method, which by default does nothing, to provide global components such as outdoor temperature models.

Aside from these methods, translators also have two properties (in the Python sense, i.e. a form of class attributes), `package_definitions` and `translate_options`. `package_definitions` is a string containing the necessary Modelica package imports. These imports serve to make the generated code more readable, and also make the translator implementations easier to write as component declarations become shorter. `translate_options` is a Python dictionary containing options which can be specified individually for each translator. The actual options are library specific, as they typically contain component parameters which are not used in the aggregation and thus not specified in the network description.

Translation Using DHNLibrary

A concrete example of code generation is the translation using DHNLibrary models which is used to generate most models considered in this thesis. The implementation is provided in the `DHNTranslator`, `DHNSimTranslator` and `DHNOptTranslator` classes (see Fig. 3.9), where the latter two are specialized to produce either simulation or optimization models as these vary slightly. These are essentially wrappers

Component	Supply in	Supply out	Return in	Return out
Producer	No	Yes	Yes	No
Leaf customer	Yes	No	Yes	No
Serial customer	Yes	Yes	Yes	Yes
Branch customer	Yes	Yes	Yes	Yes
Meeting flow customer	Yes	Yes	Yes	Yes

Table 3.1 Connector types for network elements.

for `DHNTranslator`, to which they delegate the majority of the work, only making minor changes or extensions where it is needed.

In order to translate the different parts of the network, it helps to identify their Modelica counterparts. The translator still considers each edge or vertex as a single element, but internally, this element may declare more than one Modelica component. Common to all customers is the customer model and a pump with a control input, as shown in Fig. 3.10. Other customer models extend it by adding different control volumes to the flow in- and outputs — and an additional pump in the case of meeting flows — giving the full range of network configurations considered in Section 3.3. A Dymola representation of a serially connected customer is shown in Fig. 3.11. Producer models are a single Modelica component, and they can be extended with volumes in the same way as customers. Pipes are also single components, which need no further extensions.

All elements provide connectors for some or all of *supply in*, *supply out*, *return in* and *return out*, depending on type (producer or customer) and location in the network. An overview of these connectors for all customer types needed for DHN-Library translation is presented in Table 3.1. Aside from the pipe connectors, both customers and producers also have a connector for control inputs.

This pattern shows that customer translation can be divided into two parts: first, the actual customer is generated, and then any additional components required to make it a special kind of customer can be added afterwards. Producers can be handled in a similar way. The approach also simplifies the exchange of customer models, as the customer type is only specified in a single place (the `_leaf_customer` method), although it requires the different models to have the same parameters to specify in translation.

The full set of functions in `DHNSimTranslator` and `DHNOptTranslator` is

translate Inherited from `AbstractTranslator`. Traverses the network graph, translating nodes and edges.

_global_components Inherited from `DHNTranslator`. Declares a global temperature model.

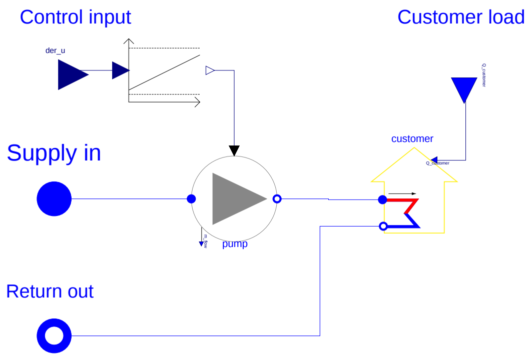


Figure 3.10 Dymola visualization of a leaf customer.

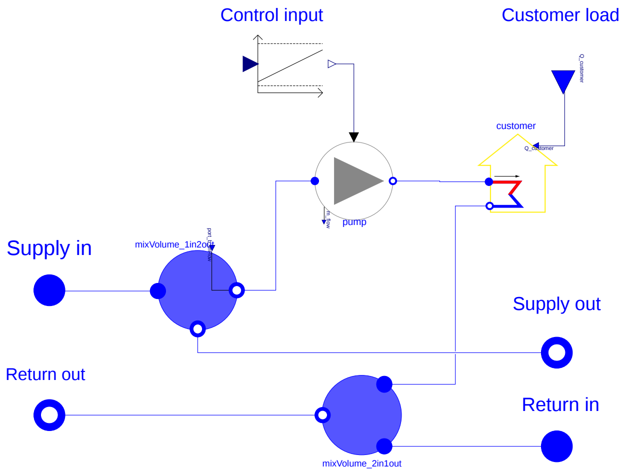


Figure 3.11 Dymola visualization of a serially connected customer. Note that most components are the same as in Fig. 3.10.

_translate_edge Translate a graph edge into a supply/return pipe pair.

_translate_node Inherited from `DHNTranslator`. Decides the appropriate method to call to translate a given node, based on its type (consumer/producer) and placement in the network (node, leaf etc.).

_leaf_customer Translate a leaf customer.

_serial_customer Translate a serial customer.

_branch_customer Translate a branch customer.

_meeting_flow_customer Translate a customer connected to two supply lines.

_producer Translate a producer with a single supply/return pipe pair.

_branch_producer Translate a producer with multiple supply/return pipes.

_n_out_split Inherited from `AbstractTranslator`. Generates volumes for branches. Called internally by `_branch_customer` and `_branch_producer`.§

Of these, an end user only has to call `translate`, as it delegates work to the other methods as necessary. Most methods are implemented higher in the class hierarchy (see Fig. 3.9), but additional details like which pipe model to use in optimization and simulation are added in these two classes.

Using the Translated Models

For numerical stability, especially in optimization, it is important that variables are properly scaled. Ideally, they are all scaled to the interval [0, 1], but some deviation can be acceptable. The simplest way to get scaling is to use the `nominal` attribute in Modelica, i.e.

```
Real big_value(nominal=1e8);
```

If a variable changes a lot, however, this may not be enough. For this purpose, JModelica.org allows variable scaling by a trajectory when optimizing. To attain such trajectories, a simulation of the system is done, under the same operating conditions as the optimization. The simulation result is then given as input to the optimization, where it is considered to be the nominal values needed.

The optimization model itself is different from the simulation model in that it has optimizable inputs where the simulation model requires predefined inputs. Also, in accordance with Section 3.1, the pipe models are different between the two models. The typical use of the optimization model is to let the optimization problem inherit it. This has the added benefit of allowing translation of the general model, though the optimization problem itself typically has to be tailored for each individual network. By extending a previously created model, the only thing left to do in the optimization problem is to specify the cost function and any constraints:

```
optimization Opt(startTime=..., finalTime=...,
                 objectiveIntegrand=cost)

    extends OptimizationModel;

    Real cost = ...;

constraint
    //constraints
end Opt;
```

Once translated, some finishing touches have to be added to the models. These are too network specific to be included in the general model, and are still small enough to be easy to add to the translated models. They are:

- Heat load models for customers
- A controller or set-point for the simulation model, used to find nominal trajectories for optimization.
- The optimal control problem specification, as discussed above.

3.6 From Data to Aggregated Model

The model generation producedure can be divided into three steps: specification, aggregation and translation. Out of these, the specification part is definitely the most verbose, but if data is given in a consistent format it can likely be automated.

As described in Section 3.4, the Network class provides two methods, `add_customer` and `add_producer`. These, in combination with the `add_edge` method inherited form `networkx.DiGraph`, are sufficient to specify an entire network. Parameters to these methods are node names and a collection of Modelica specific parameters needed in translation, along with the nominal operating conditions used in aggregation (T^{in}, T^{out}, $\dot{m}$ etc.). Below is the Python code for creating a network with a single production plant and four customers.

```
from dhn import Network, DHNSimTranslator, DHNOptTranslator

def declare_network():
    net = Network()

    net.add_producer('P',
                     {
                        'maxHeatToWater': 10e6,
                        'min_efficiency': 0.5,
                            'max_efficiency': 0.9,
                            'k': 40,
                            'delta_T_opt': 30,
                            'load': 50,
                            'pos': (0, 0)   #only needed for draw()
                        })

    net.add_customer('C1',
                        {
                            'm_flow': 10,
                            'T_in': 105 + 273.15,
                            'T_out': 75 + 273.15,
                            'load': 1.25e6,
                            'm_flow_max': 15,
                            'pos': (0.5, 0)
                        })

    net.add_customer('C2',
                        {
                            'm_flow': 10,
                            'T_in': 104 + 273.15,
                            'T_out': 74 + 273.15,
                            'load': 1.25e6,
                            'm_flow_max': 15,
                            'pos': (1, 0)
                        })

    net.add_customer('C3',
                        params={
                            'm_flow': 10,
                            'T_in': 103 + 273.15,
                            'T_out': 73 + 273.15,
                            'load': 1.25e6,
                            'm_flow_max': 15,
                            'pos': (1.5, 0)
                        })

    net.add_customer('C4',
                        {
                            'm_flow': 10,
                            'T_in': 103 + 273.15,
                            'T_out': 73 + 273.15,
                            'load': 1.25e6,
                    'm_flow_max': 15,
                    'pos': (1.3, 0.3)
                })

net.add_edge('P', 'C1',
             {
                'm_flow': 40,
                'length': 1000,
                'volume': 50,
                'T_in': 106 + 273.15,
                'T_out': 105 + 273.15,
                'T_out_return': 71.5 + 273.15,
                'T_in_return': 72.5 + 273.15
             })

net.add_edge('C1', 'C2',
             {
                'm_flow': 30,
                'length': 1000,
                'volume': 35,
                'T_in': 105 + 273.15,
                'T_out': 104 + 273.15,
                'T_out_return': 71.7 + 273.15,
                'T_in_return': 72.7 + 273.15
             })

net.add_edge('C2', 'C3',
             {
                'm_flow': 10,
                'length': 1000,
                'volume': 15,
                'T_in': 104 + 273.15,
                'T_out': 103 + 273.15,
                'T_out_return': 72 + 273.15,
                'T_in_return': 73 + 273.15
             })

net.add_edge('C2', 'C4',
             {
                'm_flow': 10,
                'length': 1000,
                'volume': 10,
                'T_in': 104 + 273.15,
```

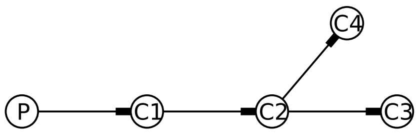


Figure 3.12 Example network before aggregation.

```
            'T_out': 103 + 273.15,
            'T_out_return': 72 + 273.15,
            'T_in_return': 73 + 273.15
        })

    return net
```

To use this network in aggregation, a call to `aggregate` is enough. The aggregated network can then be translated. Continuing on the previous script, this requires the following code

```
net = declare_network()
net.aggregate(3)    # 3 total components, producer + 2 customers

sim_translator = DHNSimTranslator()
opt_translator = DHNOptTranslator()

sim_translator.translate(net, outfile='SimulationModel.mo')
opt_translator.translate(net, outfile='OptimizationModel.mo')
```

The original and aggregated networks, as presented by `Network.draw`, are shown in Figures 3.12 and 3.13, respectively.

This is the entire automated part of the workflow. Before the model can be optimized, the manual steps described in Section 3.5 are also required.

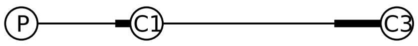


Figure 3.13 Aggregated example network.

4

Results

In this section, experiment setups and results are presented both for subnet reductions and application on a full network. The economic dispatch problem solved for for the different networks was a minimization of the production cost

$$J = \int_{t_0}^{t_f} (P_{pump}\, \dot{m} + P_{fuel}\, Q_{load} + \sum_i w_i \dot{u}_i) dt \tag{4.1}$$

over a time of 12 hours, discretized at a 5 minute resolution. Q_{load} represents the requested output of the producer, before the efficiency is applied, as this determines the actual amount of fuel needed. The prices used were relative, such that $P_{pump} = 0.05 P_{fuel}$, and were assumed to be constant during the considered 12 hour period. Furthermore, any revenue is disregarded, or can be considered to be a known constant, already subtracted from the production prices. The derivatives, while not part of the actual production cost, are included to give a well behaved solution to the optimization problem, but are given small weights to ensure that they do not interfere too much with the real cost. To get the actual cost, the contributions from the derivatives should be subtracted, and the result multiplied by a fuel price, giving the total cost in the currency of this price.

Apart from the constraints on each component (see Section 3.1), bounds were added on the control signal derivatives $\dot{u}$ to give the system more realistic response times. End point constraits on the energy contained in each pipe were also included, as

$$E_{pipe}(t_0) \leq E_{pipe}(t_f) \tag{4.2}$$

These constraints represent the fact that operation of the system should continue beyond the 12 hour horizon, and without them, the optimal solution would be to shut down all production towards the end of the considered period.

4.1 Accuracy of Aggregation Operations

Accuracy in each subnet aggregation is important, as the final model is constructed by repeatedly performing these operations. To determine which operations to include in the aggregation algorithm, 3-customer subnets and their aggregated counterpards were optimized for the following layouts:

- Three serially connected customers with one production unit.
- Three customers in a branch layout, with a single producer connected to the root customer.
- Three customers and two producers, with the two supply lines meeting at the middle customer.

Comparing the optimization results between the original and aggregated subnets gives an indication of the similarity of the model characteristics, and thus the usefulness of the operation if an accurate approximation is to be made.

Serial Subnet

Results for a serial subnet are shown in Figure 4.1, and the network itself is depicted in Fig. 4.2. Clearly, the operation preserves most of the system dynamics — an essential feature, as it is the only one to directly reduce the network size, and thus has to work for the approach to be worthwile.

Important network parameters for the original network are shown in Table 4.1. η_{max} and α are parameters for the producer efficiency as described in Section 3.1. Customer loads are very large, and loads of this size will typically only occur in late stages of aggregation. As cost function, (4.1) was used with the load control derivative weight $w_u^Q = 0.001$ and the weights of all pump control derivatives $w_u^{\dot{m}} = 0.1$. The cost functions for the aggregated and original networks are the same, apart from the the additional pump control derivative in the original network. Such an additional term is unavoidable, as the this network contains more customers, but due to the small contributions of all derivatives to the total cost, the error resulting from this should be negligible.

Branch Subnet

Trajectories for load, mass flow and supply temperature for a branch subnet before and after aggregation are shown in Figure 4.3. The network, shown in Fig. 4.4, was parametrized according to Table 4.2. (4.1) was used as cost function, and the derivative weights were the same as for the serial subnet test case, i.e. $w_u^Q = 0.001$ and $w_u^{\dot{m}} = 0.1$. As no customers are removed when merging a branch, the cost function is not affected by the aggregation.

While the approximation appears to be slightly less precise than for the serial subnet, the result is still acceptable. As is the case with aggregation of serial subnets, proper handling of branches is also important to for the method to have any practical use, as the structure is common in essentially all networks.

Parameter	Value
Producer Q_{max} (theoretical)	100 MW
η_{max}	0.9
α	$1.1 \cdot 10^{-3}$
Base load	60 MW
Peak load	70 MW
Load fraction, C_1	0.3
Load fraction, C_2	0.2
Load fraction, C_3	0.5
τ_{P_1}	5.6 min
τ_{P_2}	8 min
τ_{P_3}	11.8 min

Table 4.1 Parametrization of serial test case.

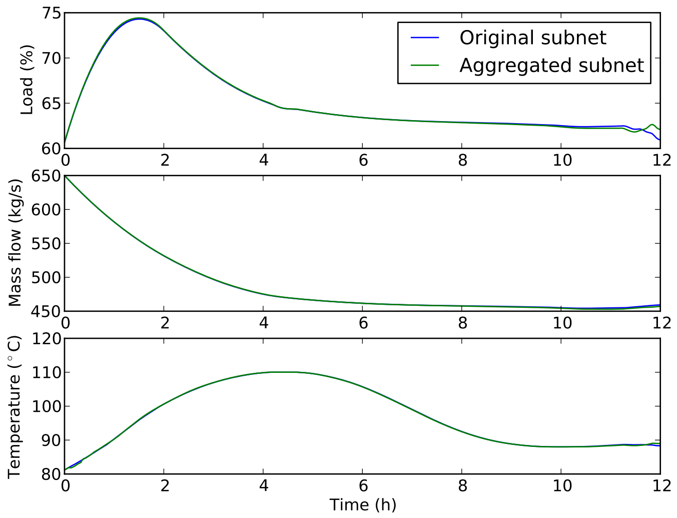
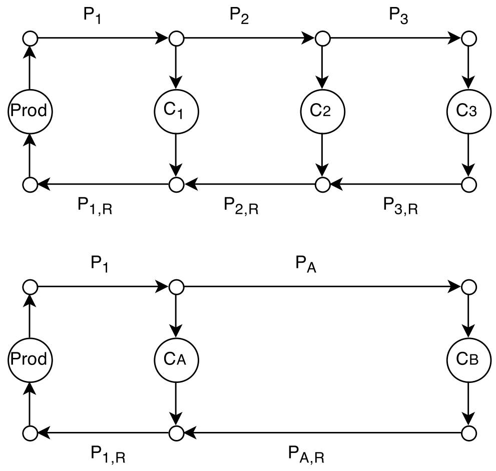


Figure 4.1 Optimal trajectories for a serial subnet and its aggregated counterpart.


P_1
P_2
P_3
Prod
C_1
C_2
C_3
$P_{1,R}$
$P_{2,R}$
$P_{3,R}$
P_1
P_A
Prod
C_A
C_B
$P_{1,R}$
$P_{A,R}$

Parameter	Value
Producer Q_{max} (theoretical)	100 MW
η_{max}	0.9
α	$1.1 \cdot 10^{-3}$
Base load	60 MW
Peak load	70 MW
Load fraction, C_1	0.3
Load fraction, C_2	0.4
Load fraction, C_3	0.3
τ_{P_1}	6.4 min
τ_{P_2}	13.9 min
τ_{P_3}	24.1 min

Table 4.2 Parametrization of branch test case.

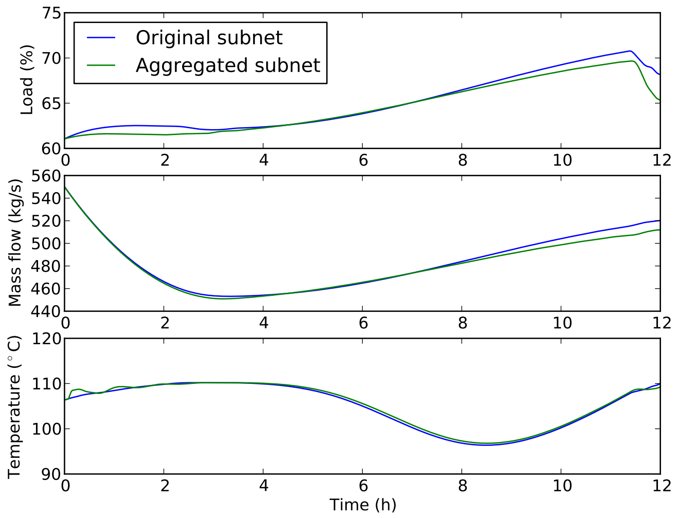


Figure 4.3 Optimal trajectories for a branch subnet before and after aggregation.

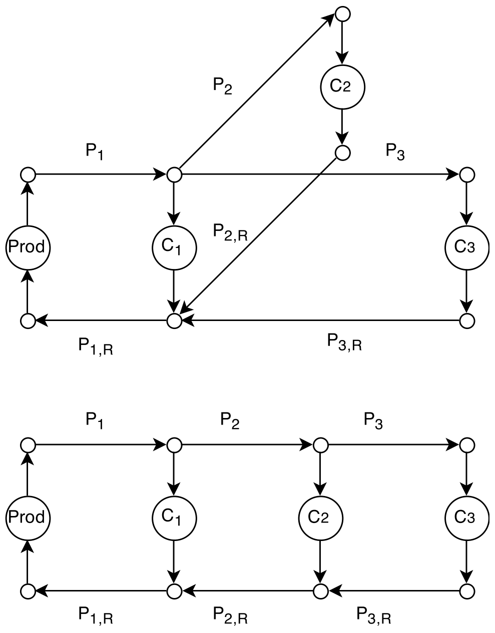


Figure 4.4 Structure of the branch subnet test case.

Meeting flow split

The splitting of customers at meeting flows is less successful than the other operations, as can be seen from Figures 4.5 and 4.6. Due to this, the operation was left out of the full aggregation procedure to improve the accuracy of the resulting model.

The operation was tested on the network shown in Fig. 4.7, with the parameters shown in Table 4.3. The two production units are identical. Similar to the other subnet test cases, (4.1) was used as cost function, with derivative weights $w_u^Q = 0.001$ and $w_u^{\dot{m}} = 0.1$. When splitting customers with meeting flows, the resulting network will be larger than the original one, thus introducing another derivative term into the cost function of the aggregated subnet. However, as all derivatives have small weights, the difference between the solution should not be affected in any major way by the extra term.

The lack of accuracy in the aggregated model is caused by the decoupling of the return temperatures of C_A and C_B. Assuming steady state conditions, it holds that $T_{C_A}^{out} = T_{C_B}^{out} = T_{C_2}^{out}$, but in general, this will no longer be true. The cause behind this is that C_A and C_B can have different inlet temperatures, naturally giving rise to different outlet temperatures as well, while the C_2 receives a single, mixed supply temperature, resulting in a single return temperature as well. This phenomenon can be exploited to independently change the supply temperatures of the producers in

Parameter	Value
Producer Q_{max} (theoretical)	60 MW
η_{max}	0.9
α	$6.25 \cdot 10^{-4}$
Base load	60 MW
Peak load	70 MW
Load fraction, C_1	0.2
Load fraction, C_2	0.4
Load fraction, C_3	0.4
τ_{P_1}	13.9 min
τ_{P_2}	26 min
τ_{P_3}	38.3 min
τ_{P_4}	10.2 min

Table 4.3 Parametrization of meeting flow split test case.

ways which would not be possible in the original, fully connected network.

4.2 Application to a Full Network

To check the validity of optimal control signals calculated for an aggregated network, these can be applied to a simulation model of the full network. As the optimization is done on a network with less customers than the full one, the customer control inputs for the individual customers have to be reconstructed from the optimal inputs. This reconstruction is done based on the fact that each aggregated customer contains fractions of the original customers

$$\dot{m}_{C_A} = \sum_i k_i \cdot \dot{m}_{C_i} \tag{4.3}$$

Conversely, this allows reconstruction of an original customer as

$$\dot{m}_C = \sum_i w_i \cdot \dot{m}_{C_A,i} \tag{4.4}$$

where k_i is the fraction of each aggregated customer provided by the original customer considered. This assumption only holds if all customers share the same load profile, and due to this the total load of the network is modeled, with each customer receiving a fraction of this load rather than having an individual model.

Since both mass flows and customer loads are fixed when validating, these do not deviate from the values found in the optimzation. Instead, all errors will express

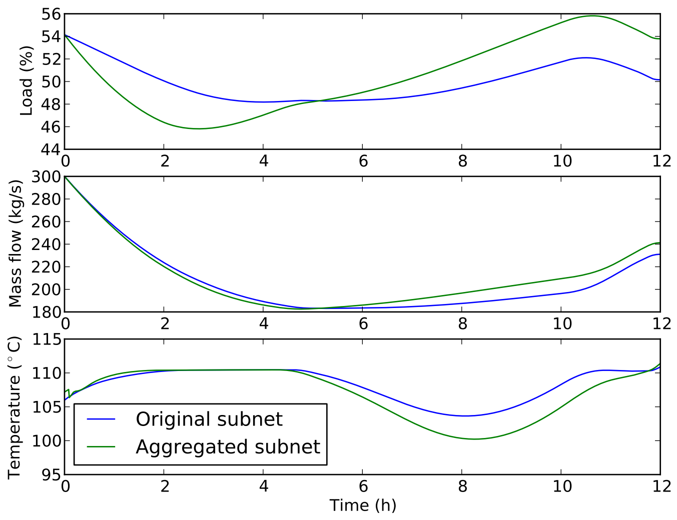


Figure 4.5 Optimal trajectories for P_1 before and after splitting a customer with meeting flows.

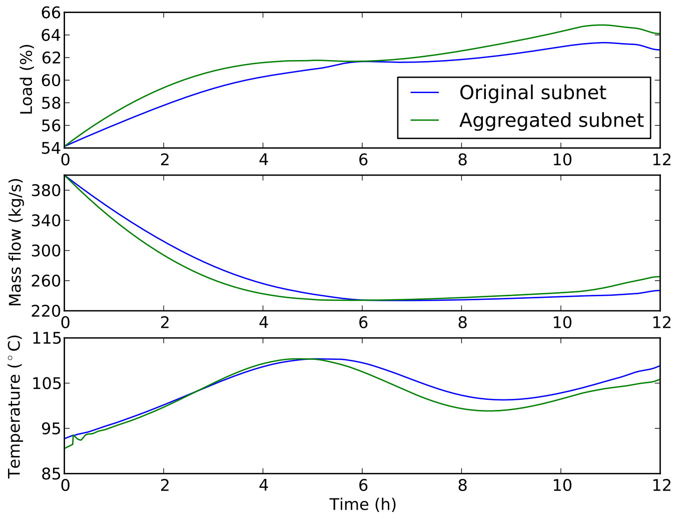


Figure 4.6 Optimal trajectores for P_2 before and after splitting a customer with meeting flows.

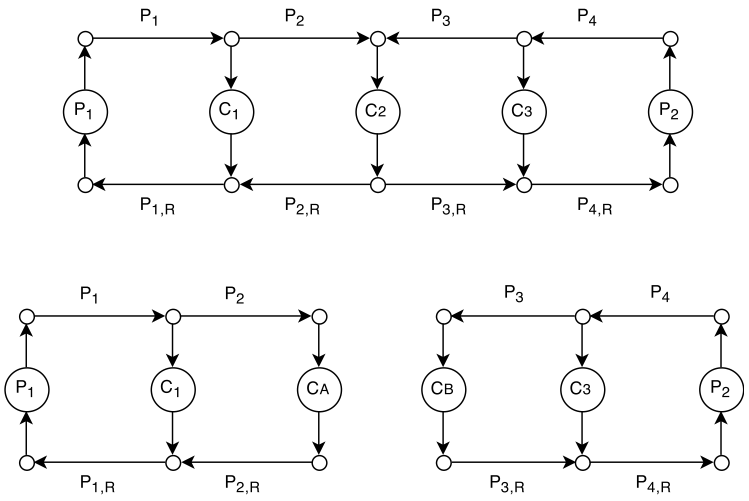


Figure 4.7 Structure of the meeting flow split test case.

themselves in the network temperature. Too high mass flow for a given supply temperature will result in a lower ΔT at a customer than expected, while a lower mass flow will increase it. The heating at the producer follows the same pattern, i.e. a positive temperature error means that more heat than necessary was supplied and a negative error indicates under-production. Because the water is circulated in the distribution network, the error will also be cumulative, with an error in production will result in a temperature difference that remains unless another error cancels it out.

Test Network

The network used in all tests of the aggregated models is based on the Ishøj DHN, using the data given in [Larsen et al., 2004]. The network, shown in Fig. 4.9 has 23 customers, a single producer and no loops. As the article does not provide the full specification of the network the layout, load distribution and average supply and return temperatures are used. Where temperature data for customers is unavailable, a temperature drop of 30 °C is assumed. Pipe diameters are chosen to achieve delays in the range of ca 5–20 minutes between customers.

The network is optimized for 12 hours, from midnight to noon, with a 5 minute resolution in the discretization. The base demand is 16 MW, with a peak demand of 22 MW at 07:00. The outdoor temperature begins at -5°C and goes toward a top temperature of 5°C at 14:00, though the peak itself is beyond the optimization horizon. The total load and outdoor temperature are shown in Fig. 4.8 Trajectories for variable scaling are generated by keeping all mass flows at their nominal values,

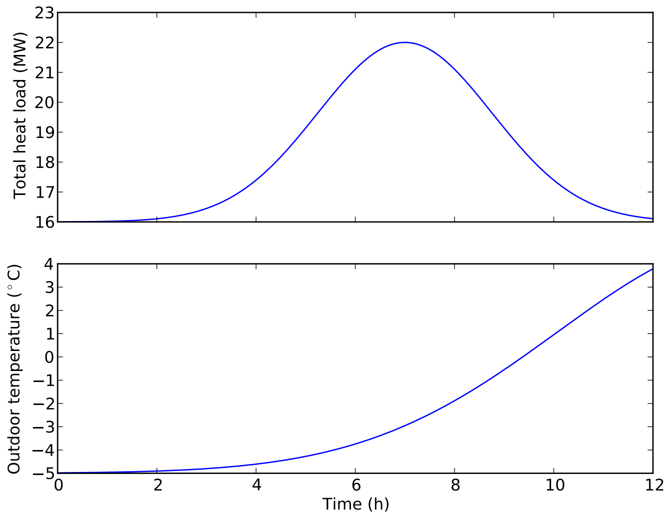


Figure 4.8 Total heat load and outdoor temperature during the optimization period.

and determining the production level at the power plant using a PI-controller with a gain of $5 \cdot 10^{-4}$ and time constant 1. The controller acts to keep the supply temperature of a customer close to the plant (S39) at a reference temperature of 105°C. It does this with rather low precision, but this does not matter as it keeps the load and all temperatures within reasonable ranges, making them useful as nominal values.

All optimization results contains trajectories for how the system dynamics change over time, given the optimal input signals. To validate these results, a model of the full network is simulated using the optimal trajectories as inputs. The error for a variable x is caclulated as the difference between the result from the optimization and the validation result found by simulation

$$x_{error} = x_{validation} - x_{expected} \tag{4.5}$$

Aggregated Models

Three different aggregated models are optimized, with 3, 4 and 5 customers, respectively. The network graph, prior to aggreagtion, is shown in Fig. 4.9, and the models, along with an intermediate aggregation step of 10 customers, is shown for comparison in Fig. 4.10. 3 customers is the maximal degree of aggregation, as customer S39 is connected to the producer by a separate supply pipe. In the current implementation, it is impossible to aggregate this customer, as producer nodes in the graph are not considered in aggregation. A comparison between values in the full network and the 3-customer approximation is given in Table 4.4 to show the effect of the aggregation. As S39 is not changed in any way, the other two remaining customers reperesent the rest of the network, and both their mass flow and the

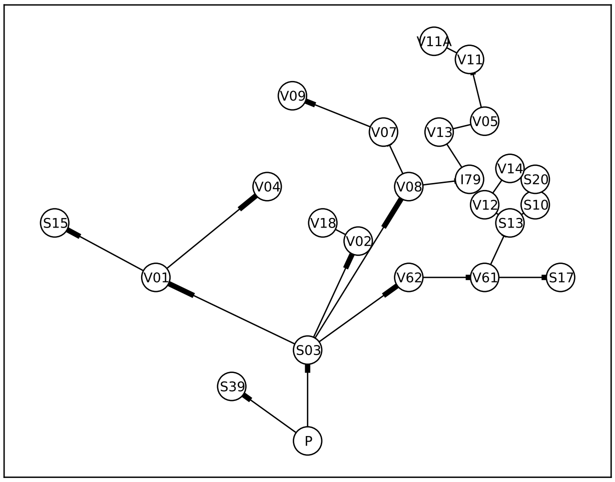


Figure 4.9 Graph representation of the full Ishoej DHN. The producer is denoted by P, and all other nodes are customers.

	Original		Aggregated	
Customer	Load fraction (%)	$\dot{m}$ (kg/s)	Load fraction (%)	$\dot{m}$ (kg/s)
S39	2.8	4.40	2.8	4.4
S03	73.1	116	33.7	49.0
V11A	24.1	0.60	0.4	38.4

Table 4.4 Comparison between original network and an aggergated version with 3 customers.

fraction of the total heat load that they make up have increased significantly. This is especially true for V11A, which sees an increase in both mass flow and heat demand by ca 600%.

Results

A plot of the supply temperature error at the producer is presented in Fig. 4.11, and average and RMS errors for production plant and customers are given in Table 4.5. The corresponding error trajectories are shown in Fig. 4.11, and Fig. 4.12 shows the relative error in the delivered heat. To find this error, the missing load is calculated from the supply temperature error, as defined in (4.5):

$$Q_{error} = c_p \dot{m}_{producer} T_{error} \tag{4.6}$$

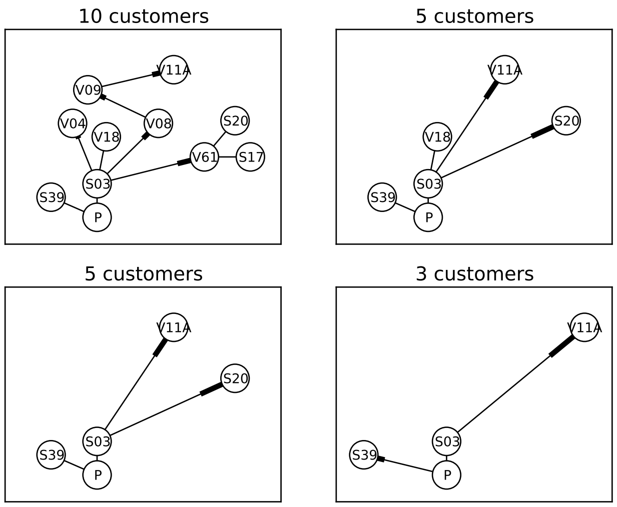


Figure 4.10 Graph representation of steps in the aggregation. The node labelled P is the producer, other nodes are customers.

Customers	Error	
	RMS	Average
3	7.4674	-5.0241
4	4.7591	-2.7104
5	3.9151	-2.0130

Table 4.5 Supply temperature error at producer.

All models display large relative errors, though they decrease with the size of the aggregated model. An interesting characteristic of all results is that the error at the end of the production plan is smaller, which, with the cumulative nature of the error in mind, implies that a relatively accurate amount of heat is delivered, but at the wrong time, allowing the error to mostly cancel out over the course of the simulation.

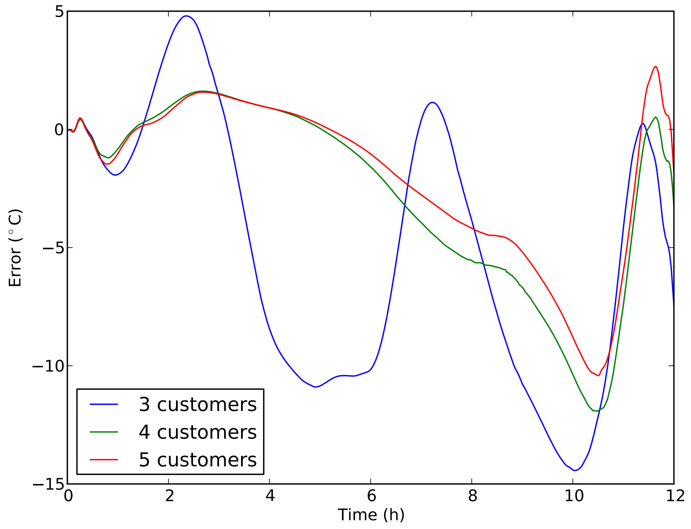


Figure 4.11 Supply temperature error at production site for different optimization results.

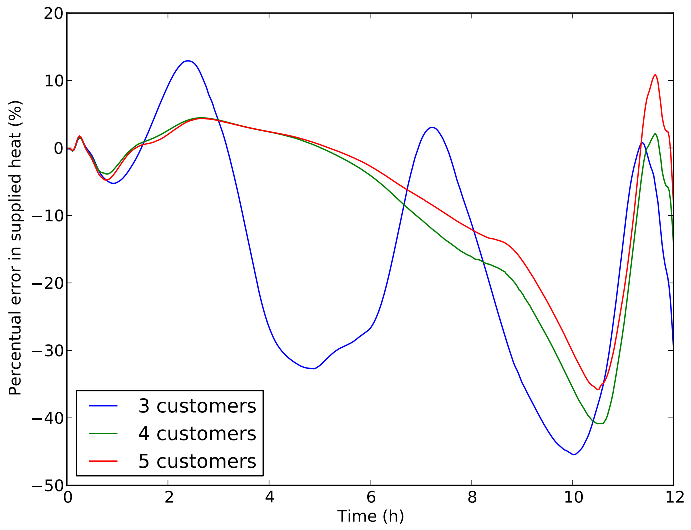


Figure 4.12 Percentual error in delivered heat for different aggregation levels.

5 Discussion

The experiment results show two major problems with the aggregated models. The first one is the inaccuracy that arises when splitting customers with multiple incoming flows, and the other is the large errors in supplied heat. Probable causes for both of them are based on violations of the hydraulic steady state assumption made in aggregation. More such pitfalls may exist, as the original method was developed for simulation, and not optimization. Typically, optimization algorithms are more likely to find and exploit these violations, making them a larger problem in optimization.

5.1 Meeting Flow Split

As seen in the results for the subnet where meeting flows are separated, the discrepancy between the optimization results are larger than for serial or branch subnets. The inaccuracy can be explained by the decoupling of the return temperatures caused by the split. Under the steady state assumption of the aggregation method, this does not happen, but whenever the network state deviates from this point, the return temperatures of the two customers created in the split will likely differ.

In the original network, if one of the incoming supply temperatures is significantly lowered, this will affect both producers as the return temperature is dependent on the mixed supply temperatures. Once the customers are split, this dependency is removed and the two production sites are thus not forced to cooperate as much.

The problem could be avoided by adding a constraint stating that

$$T_{C_A}^{out} = T_{C_B}^{out} \tag{5.1}$$

but if this is done, the model with the split will actually be more complex than the original, as it has an additional customer, while retaining the coupling between the two meeting flows which was supposed to be removed.

The consequence of not being able to handle meeting flows in loops or from several producers is that these customers cannot be aggregated, thus limiting the minimal size of the aggregated networks. If the network is too complex, it will be impossible to aggregate to an optimizable size unless the extra error is acceptable.

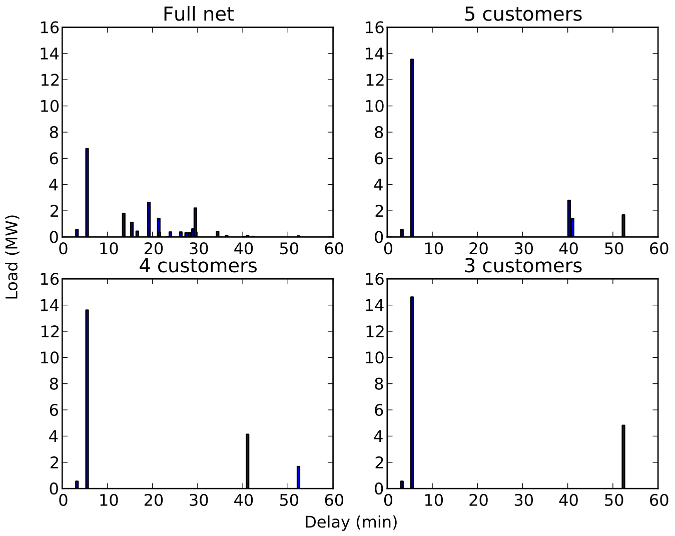


Figure 5.1 Load distributions of original and aggregated networks.

The more production sites there are in a network, the greater the risk for an impossibly large model becomes, as there will be one customer which is impossible to aggregate along each pipe between two production sites.

5.2 Distortion of the Load Distribution

The large errors in heat delivery can be attributed to distortion of the load distribution, caused by the aggregation procedure. Each time a customer is removed, its load is split between the neighbouring customers. Under the steady state assumption of the aggregation method, this does not affect the network characteristics, but as soon as the delays vary (i.e., the mass flows change), the heat demand associated with the removed customer will receive the wrong delay. As a result, the producer will see part of the load demand of the customer too early, and part of it too late. For large degrees of aggregation, this will change any original load distribution into two large loads, one at the customer closest to the producer, and one at the the farthest one. Fig. 5.1 shows a comparison of the load distributions of the full test network and the aggregated models used, clearly displaying this behaviour. There are (at least) two possible solutions to this problem. Either the individual load profiles are shifted in time, or the aggregation algorithm can be modified to maintain a more similar load distribution.

Time-shifted Customer Loads

One way to address the problem is by maintaining the original load profiles for each customer, shifted by the delay between the original and the aggregated customer. In the current formulation, the load Q_{C_A} of an aggregated customer is the sum of the loads of the original customers it contains, i.e.

$$Q_{C_A} = \sum_i k_i Q_{C_i} \tag{5.2}$$

This would be modified by introducing a time shift τ_i, corresponding to the delay, into the demand profiles of the original customers, as

$$Q_{C_A}(t) = \sum_i k_i Q_i(t - \tau_i) \tag{5.3}$$

The delays τ_i are not constant, but can, as most other parameters, be derived from the nominal values, utilizing the fact that

$$\tau = \frac{\rho V}{\dot{m}} \tag{5.4}$$

Such a solution would also scale well to networks with multiple producers, as long as the currently made assumption on constant flow directions is kept, since this ensures that all customers affected by aggregation will only be supplied by a single producer

Selective Aggregation

An alternative approach is to change the aggregation algorithm itself to be more selective when removing customers. Currently, a greedy algorithm is used, which simply goes through all customers and selects them for aggregation in the order they are encountered. A more careful selection of customers could be done, keeping customers at certain delays from the plant to maintain the overall shape of the load distribution. In the distribution shown in Fig. 5.1, this would correspond to keeping one or more customers around 20-30 minutes to decrease the shifting of the load distribution towards higher delays.

For high degrees of aggregation, this modification will not be able to counteract the effects on the load distribution, as customers in the middle of the network will sooner or later have to be aggregated into ones closer to or farther from the production plant. To prevent this, some customers could be excluded from aggregation, limiting the minimum size of the model but ensuring more accurate time delays.

5.3 Future Work

Though the foundation for the aggregation procedure has been laid out, both in theory and implementation, much remains to be done. A major point is adjusting the aggregated customer loads as discussed in the previous section to provide a better approximation of the full network. If this modification has the desired effect, a more thorough validation of the method is in order. Some areas which could be investigated are:

- Validation of results using a more complicated model. The current approach uses a feed forward control, and ideal components. A more realistic simulation model could give greater insights into the model accuracy. A step further in this direction is to automate the generation of such models.
- Evaluation of the performance of aggregation in networks with a more complex structure. The test network which is currently used has a tree structure with a single producer, which is the most aggregation friendly layout. To perform well in real applications, the aggregation method has to work for networks with loops and/or power plants as well.
- Including pressure differences in the aggregation. The possibility to do this was one of the reasons for choosing the aggregation method, but pressure modeling is not currently available in the Modelica library that is used.
- Use of aggregated networks in a more complete setting, with e.g. more detailed power plant or cost models.
- Implement additional functionality to automatically generate a netork given only the topology and measurement or simulation data, rather than the explicit steady state to use in aggregation. This would allow the aggregated model to be easily changed when the network dynamics change, something that is not possible with the current approach of hard coding the network specification.

Any further application of the aggregation algorithm can also help with gaining insight into when the steady state assumptions do not hold, and if adjustments can be made to avoid those problems.

6

Conclusions

The goal was to implement and evaluate an approach to generate approximate models of the distribution network in district heating systems. As a basis, a previously developed method for aggregating models for simulation was used, with added extensions to handle problems related to optimization. The strength of this method lies in its detailed knowledge of the original network, which is utilized to find an aggregated network that is equivalent in a nominal case. The use of a nominal case is also a weakness, as deviating from it can introduce various errors in the created model.

The modified aggregation method has been developed along with an abstract network representation and Modelica code generation, and these are implemented in a Python package which allows automation of the workflow from network data to nearly complete optimization and simulation models.

Initial experiments indicate that the method holds promise, but needs further refinement to be accurate enough for real real world applications, especially in an optimization setting. Currently, the most pressing issue is errors in the network time delays, for which a solution is proposed but not implemented. If it works as expected, the aggregation procedure should be ready for testing in a general production planning scenario.

Bibliography

Åkesson, J. (2008). “Optimica—an extension of Modelica supporting dynamic optimization”. In: *In 6th International Modelica Conference 2008*. Modelica Association, Bielefeld, Germany.

Åkesson, J., K.-E. Årzén, M. Gäfvert, T. Bergdahl, and H. Tummescheit (2010). “Modeling and optimization with Optimica and JModelica.org—languages and tools for solving large-scale dynamic optimization problems”. *Computers and Chemical Engineering* **34**:11, pp. 1737–1749.

Andersson, C., C. Führer, and J. Åkesson (2014). *Assimulo: A Unified Framework for ODE Solvers*. eng. Tech. rep. LUTFNA-5005-2014. Centre for Mathematical Sciences, Lund University, p. 24.

Bøhm, B. and H. Larsen (2005). *Simple models of district heating systems for load and demand side management and operational optimisation*. Technical University of Denmark. ISBN: 87-7475-323-1.

Frederikssen, S. and S. Werner (2014). *District Heating and Cooling*. Sudentlitteratur. ISBN: 978-91-44-08530-2.

Grosswindhager, S., A. Voigt, and M. Kozek (2012). “Predictive control of district heating network using fuzzy DMC”. In: *Modelling, Identification Control (ICMIC), 2012 Proceedings of International Conference on*, pp. 241–246.

Hagberg, A. A., D. A. Schult, and P. J. Swart (2008). “Exploring network structure, dynamics, and function using NetworkX”. In: *Proceedings of the 7th Python in Science Conference (SciPy2008)*. Pasadena, CA USA, pp. 11–15.

Larsen, H. V., H. Pálsson, B. Bøhm, and H. F. Ravn (2002). “Aggregated dynamic simulation model of district heating networks”. *Energy Conversion and Management* **43**:8, pp. 995 –1019. ISSN: 0196-8904.

Larsen, H. V., B. Bøhm, and M. Wigbels (2004). “A comparison of aggregated models for simulation and operational optimisation of district heating networks”. *Energy Conversion and Management* **45**:7–8, pp. 1119 –1139. ISSN: 0196-8904.

Larsson, P.-O., H. Runvik, S. Velut, S. M. Razavi, M. Bohlin, and J. Funkquist (2014). *Decision Support for Short-Term Production Planning of District Heating using Non-linear Programming*. Tech. rep. 1262. Värmeforsk.

Loewen, A. (2001). *Entwicklung eines Verfahrens zur Aggregation komplexer Fernwärmenetze*. PhD thesis. Universität Dortmund, IV, 142, 18 : Ill. ISBN: 3-8167-5909-2 and 978-3-8167-5909-6.

Magnusson, F. and J. Åkesson (2012). "Collocation methods for optimization in a Modelica environment". In: *Proceedings of the 9th International Modelica Conference*. Munich, Germany.

Nielsen, T. (2002). *Online prediction and control in nonlinear stochastic systems*. PhD thesis. Technical University of Denmark.

Pinson, P., T. Nielsen, H. Nielsen, N. Poulsen, and H. Madsen (2009). "Temperature prediction at critical points in district heating systems". *European Journal of Operational Research* **194**:1, pp. 163 –176. ISSN: 0377-2217.

Saarinen, L. and K. Boman (2012). *Optimized district heating supply temperature for large networks*. Tech. rep. 1227. Värmeforsk.

Sandou, G., S. Font, S. Tebbani, A. Hiret, and C. Mondon (2005). "Predictive control of a complex district heating network". In: *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, pp. 7372–7377.

Wächter, A. and L. T. Biegler (2006). "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". English. *Mathematical Programming* **106**:1, pp. 25–57. ISSN: 0025-5610. DOI: `10.1007/s10107-004-0559-y`. URL: `http://dx.doi.org/10.1007/s10107-004-0559-y`.

Wigbels, M. (2002). "Oberhausen district heating system". In: Bøhm, B. (Ed.). *Simple models for operational optimisation*. IEA District Heating and Cooling, Annex VI: Report 2002:S1. Netherlands Agency for Energy and the Environment. ISBN: 90-5748-021-2.

Lund University **Department of Automatic Control** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER´S THESIS *Date of issue* June 2015 *Document Number* ISRN LUTFD2/TFRT--5975--SE
Author(s) Henning Larsson	*Supervisor* Pontus Giselsson, Dept. of Automatic Control, Lund University, Sweden Anders Rantzer, Dept. of Automatic Control, Lund University, Sweden (examiner) *Sponsoring organization*

Title and subtitle

District Heating Network Models for Production Planning

Abstract

In this thesis, a methodology for aggregating district heating networks for use in dynamic optimization with the JModelica.org platform is investigated. Several methodologies for this network reduction are reviewed, and one, based on a detailed physical model of the system, is presented in greater detail. This method is implemented along with an abstract network representation and Modelica code generation to automate as much as possible of the workflow from network data to simulation and optimization models.

Experiments are presented both for aggregation for typical cases with 3–4 conusmers in a sub-network, and for a full district heating network. The tests on the smaller networks are mostly accurate, and while the results from the full network show the need for further improvement of the method, a likely source for the error is presented, along with solution proposals.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280-5316		*ISBN*
Language English	*Number of pages* 1-67	*Recipient's notes*
Security classification		

http://www.control.lth.se/publications/