Optimising District Heating Operations

Lars Stegman

Delft University of Technology

TU Delft

withthegrid

Optimising District Heating Operations

by

Lars Stegman

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on 16th October 2019

Project duration:	1st February 2019 – 16th October 2019	
Thesis committee:	Dr. M. de Weerdt,	TU Delft, supervisor
	Prof. P. Bosman,	TU Delft, EEMCS - Algorithmics
	R. Everhardt, MSc.	withthegrid, external expert

This thesis is confidential and cannot be made public until 16th October 2020.

TU Delft Delft University of Technology

Preface

During the writing of my thesis I have had great support from my supervisors Mathijs de Weerdt and Rob Everhardt. I would like to thank them for all the help, ideas and feedback they have given me during this project. Mathijs has guided me through the academic process very well and has often helped me see the bigger picture and the best next steps, while still allowing me to choose my own direction. Rob has always been accommodating and has helped me solve several problems I ran into by helping me find the mistakes in my reasoning.

I would also like to thank all the people at withthegrid for making me feel welcome and making me part of the day-to-day business at withthegrid. I wish them all the luck in expanding their business, which I am convinced will make a significant impact in the energy transition that is about to begin.

Finally, I would like to thank my family and friends for the support they have given me in both my academic and personal life. Without their unconditional love and support, it would not have been possible for me to accomplish this.

Lars Stegman
Barendrecht, 10th October 2019

Contents

List of Figures

List of Tables

List of Algorithms

1

Introduction

The Netherlands has set the goal for itself to reduce CO_2 emissions by 95% in 2050 [20]. To reach this goal, many fundamental changes are needed in the way society works. In 2017, buildings were responsible for 14.7% of CO_2 emissions, the majority of which was caused by burning natural gas for heating. Most buildings use individual gas boilers to generate their own heat. In 2016, more than 75% of the homes in The Netherlands used a boiler [13]. To reach the emission reduction goal set by the government, few homes can be heated using gas any more.

One of the alternatives would be to outfit homes with heat pumps and electric boilers for heating and warm tap water, respectively. However, district heating systems (DHSs) are considered a better, cheaper alternative as heat pumps and electric boilers use large amounts of electricity, which would require extensive investments in the electrical grid, despite homes being outfitted with photovoltaic (PV) panels.

With DHSs heat is produced in a (or multiple) central source(s) and distributed to consumers. The heat for a DHS can come from any source, renewable or otherwise. In the city of Rotterdam, residual heat from the chemical refinery Shell Pernis is going to be used for heating homes in the district of Pernis. In the city of Papendrecht, the use of a geothermal source as a heat source for a DHS is being researched. [27]

Currently, most DHSs are operated by choosing a temperature once every day depending on the weather. This setting is chosen such that peak demand can be satisfied. However, when demand is lower, the supply temperature will be higher than necessary and heat will be wasted. In addition to this, heat production costs can be dynamic over time, which gives room for more cost-efficient heat production scheduling. By choosing a dynamic temperature over the day losses and operating costs can be reduced. However, determining these dynamic temperatures is not easy, as there are several factors that need to be taken into account to ensure that demand can always be fulfilled. Thus, by optimising the heating schedule for the DHS both production cost and possibly CO_2 emissions can be lowered.

1.1. Research Questions

Optimisation of DHSs is not a new idea, as many approaches can be found in literature [2, 4, 9]. The novel approach of this research is the application of metaheuristics, whereas other research primarily focusses on mathematical optimisation.

As the optimisation problem is both non-linear and non-convex, the mathematical optimisation makes certain assumptions, e.g. approximating exponential terms by linearising them using a simulator. Most mathematical optimisations build a mathematical model of the DHS under optimisation, but with metaheuristics, this is not needed, as the fitness function can be optimised directly. This removes complexity and makes it possible to use elements of the DHS model that are too complex to model in the mathematical optimisation model.

In this report the following questions will be researched:

RQ 1 What metaheuristic optimisation approaches work well for solving the DHS optimisation problem?

RQ 1.1 What is needed to make the use of metaheuristics feasible?

RQ 2 How much improvement does an optimised heating schedule offer with respect to the existing operational methods?

RQ 2.1 Is a lower bound on the production costs available (in literature) or is it possible to determine a useful one?

RQ 3 How do the approaches compare to the existing mathematical optimisation approaches found in literature?

1.2. Outline

This report is laid out as follows.

Chapter 2 gives a comprehensive description of how DHSs work and the underlying physics required to understand the problem.

In Chapter 3 the model of the DHS operation optimisation problem is described, including the corresponding constraints, problem complexity, resulting objective function and a lower bound for the problem.

Chapter 4 gives an overview of existing approaches to optimise DHS operations and other related research, like demand forecasting and DHS system design.

Chapter 5 describes several methods which have been attempted to optimise the problem and a method to circumvent an expensive part of the simulation.

Then, in Chapter 6 several DHS scenarios are described. These scenarios are then used in the subsequently described experiments.

In Chapter 7 the results from the experiments in the previous chapter are discussed and the underlying causes for the results are investigated.

To conclude this report, Chapter 8 draws conclusions from the results, answering the research questions, and discusses possible future research directions.

2

Background

This chapter gives a description of how district heating systems work and the relevant underlying physics concepts.

Section 2.1 describes in detail how DHSs work, how they are operated and gives several examples to illustrate what effects choices in operation have.

Section 2.2 gives an overview of relevant physics concepts to give a more theoretical foundation for the earlier section. In addition to this, a description of heat exchangers is given, which is one of the most complex, but also most important, elements of calculations on a DHS.

2.1. District Heating Systems

A district heating system (DHS) consists of three components: producers, consumers, and a district heating network (DHN) to circulate heated water from producers to consumers and cooled down water back again. In Figure 2.1 an overview of the icons used in diagrams is given.

2.1.1. Network Topology

In Figure 2.2 a schematic view of a DHS is shown.

Transportation

A typical DHN consists of a primary and a secondary network. The primary network consists of transportation pipes which distribute large amounts of heated water from heat production facilities to heat transfer stations (HTSs) and/or consumers. From HTSs, heat from the primary network is transferred to

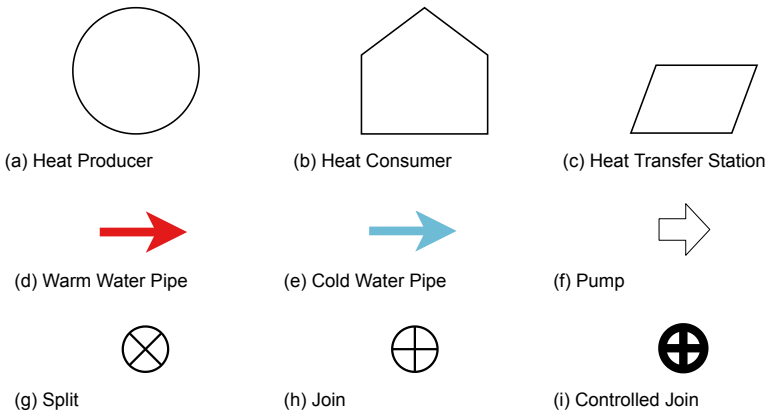


Figure 2.1: Diagram icons

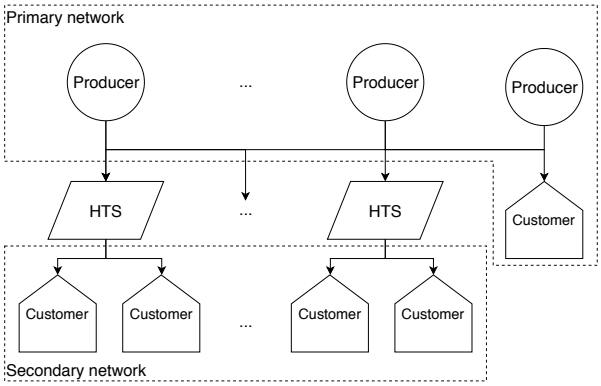


Figure 2.2: DHS schematic topology

the secondary network. In addition to HTSs, some large consumers, e.g. hospitals or shopping centres, can also connect to the primary grid directly.

The secondary network consists of smaller pipes than the primary network as the primary network transports heat for multiple secondary networks combined. As the pipe diameters are smaller, the surface area of the pipes in relation to their diameter is larger, which means heat is lost to the environment more easily. This means most energy losses occur in the secondary network.

In this research it is assumed that the DHN is tree-like. This means there is only one route from a producer to a consumer and that there are no bidirectional pipes in the DHN.

Connections

As explained above, the primary and secondary network consist of pipes to transport water. These pipes can be seen as edges of a graph and anything that connects (to) pipes as nodes. These nodes are either infrastructure nodes, like pumps, splits, joins, producers, HTSs, or heat sinks, like consumers and HTSs.

Every element of the network is assumed to have conservation of mass flow, which means that the sum of outgoing flows is equal to the sum of incoming flows. In other words, it is assumed there are no leaks in the network.

2.1.2. Operation

A DHS is operated by setting supply temperatures for heat sources, like producers and HTSs, and controlling pumps in the network. These pumps maintain pressure so consumers receive flow when they open their valves to consume heat.

Changes in production temperatures propagate through the DHN at a speed depending on the flow velocities in the network. Higher flow velocities spread water through the DHN faster, resulting in a low propagation delay. Since the warm water spends less time in the network before it arrives at a consumer, less heat will be lost to the environment, resulting in higher efficiencies. When flow velocities are low, water spreads through the network more slowly, leading to higher heat loss.

Central to calculations on DHSs is the heat equation (2.1), which states that the heat power taken from/put into the system is equal to product of the specific heat capacity of water[1], mass flow and the induced difference in temperature.[2]

$$Q = c\dot{m}\Delta T \qquad [\mathrm{W}]^{3} \tag{2.1}$$

Increasing the production temperatures at producers leads to lower mass flows in the network, since consumers need less water to receive the same amount of heat. As this lower mass flow leads to higher

[1] The amount of energy required to heat 1 kg by 1 K
[2] Note that heat and temperature are different things: heat is a form of energy that causes an object to have a certain temperature.
[3] Units: $\mathrm{J/(kg\,K)} \cdot \mathrm{kg/s} \cdot \mathrm{K} = \mathrm{J/s} = \mathrm{W}$

losses in the network, a simple solution would be to simply lower the temperature as much as possible until the maximum flow velocities are reached. Unfortunately, demand varies throughout the day and if an insufficient amount of heat is stored in the network ahead of time, it might become impossible to fulfil demand later in the day.

Both reheating the water to the set point supply temperatures and maintaining pressure in the network have costs associated. These costs vary depending on the time of day and the type of energy or production plant used. For example, electricity is usually more expensive in the morning when everyone is getting ready for the day, thus it might be more profitable to sell electricity instead of heat from the production plant.

Currently, heating schedules are determined once every day by using a predetermined heating curve. An example of a heating curve can be seen in Figure 2.3. In this example, the average outdoor temperature for the day is 8 °C, which leads to a set point supply temperature of 80.3 °C for the producer in the network. Essentially, the temperature settings are fixed to a set value for the entire day and only the pumping power is varied to fulfil demand at all times.

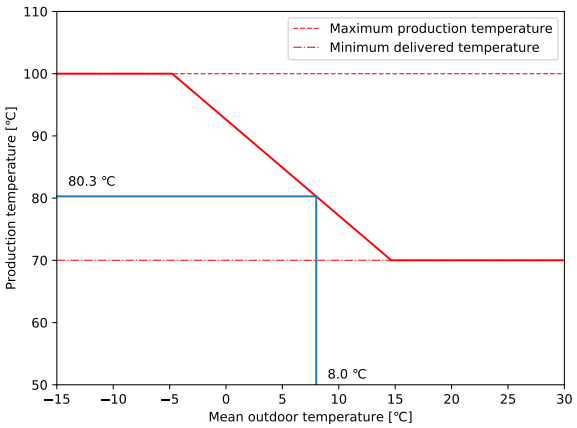


Figure 2.3: Example heat curve. A mean outdoor temperature of 8.0 °C results in a supply temperature of 80.3 °C at the heat producer

Because demand varies throughout the day, operating the DHS like this is not very efficient. When demand is low, a low flow velocity is needed to fulfil demand, which leads to higher losses in the DHN. In addition to this, as heating and pumping costs vary over time, it might be more efficient to heat more, earlier, than less, later, essentially using the DHN itself as a heat store.

Operating Costs

The operating costs of a DHS are determined by the costs for heat generation and pumping costs.

When $Q_t^p(\mathbf{T}_t^p)$ is defined to be the amount of energy produced by producer p at time t given output temperature $\mathbf{T}_t^p$ and $\dot{W}_t^{\dot{p}}$ is the pumping power by pump $\dot{p}$ at time t, then the overall operating cost is defined by:

$$c_{\text{op}}(\mathbf{T}, \dot{\mathbf{W}}) = \delta \sum_{t \in T} \left(\sum_{p \in P} c_t^p Q_t^p(\mathbf{T}_t^p) \right) + \left(\sum_{\dot{p} \in \dot{P}} c_t^{\dot{p}} \dot{\mathbf{W}}_t^{\dot{p}} \right) \tag{2.2}$$

where δ is the interval length, c_t^p the price of producing one joule of heat by producer p at time t, and $c_t^{\dot{p}}$ the cost of using one joule of electricity by pump $\dot{p}$ at time t. $\mathbf{T}$ is the set of output temperature settings of heat sources.

District Heating System Calculations

As described above, the control variables of a DHS consist of the heat source output temperature settings and the pumping power.

Three main variables are important to determine how a DHS behaves:

- Heat sources output temperatures
- Mass flows at every point in the network
- Heat consumed by consumers

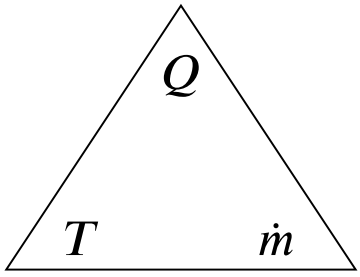


Figure 2.4: Variable triangle for DHS simulation. When two are known, the variable in the corner opposite the connecting side can be determined

If two variables values are known, the third can be calculated, which can be illustrated by the variable triangle shown in Figure 2.4. One combination can be easier to calculate than another, though, as certain combinations would require complex pressure calculations or calculations that require expansive stateful calculations.

One combination can be ruled out immediately: determining the output temperatures from the mass flows and consumed heat, as this is exactly the problem under optimisation for simple networks with a single heat source.

Examples

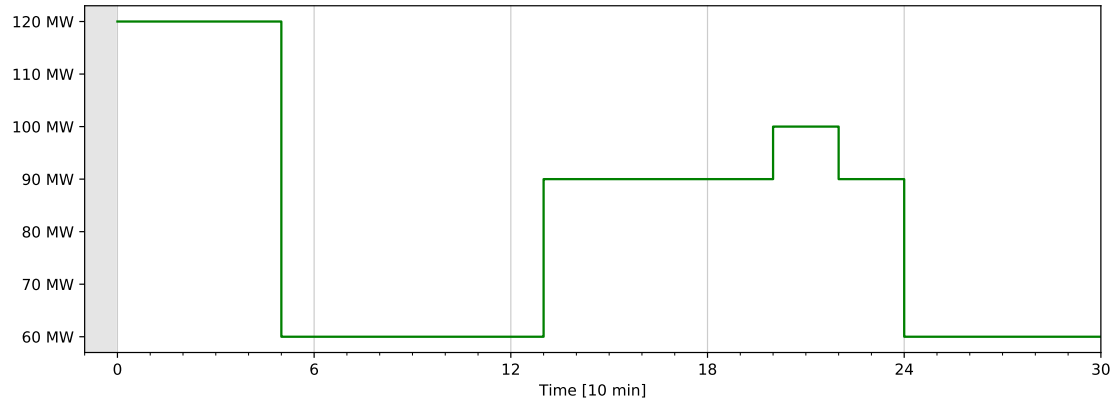
In this section, several small operation examples will be given to give a better intuition on the frequency response of a DHS network given changes in supply temperature or demand. In these examples, the mass flows are determined given the consumed heat and supply temperatures.

All examples consist of 1 producer and 1 consumer which are connected with two DN600[4] pipes with a length of 2704 m each. The simulation horizon covers 5 hours with an interval length of 10 min.

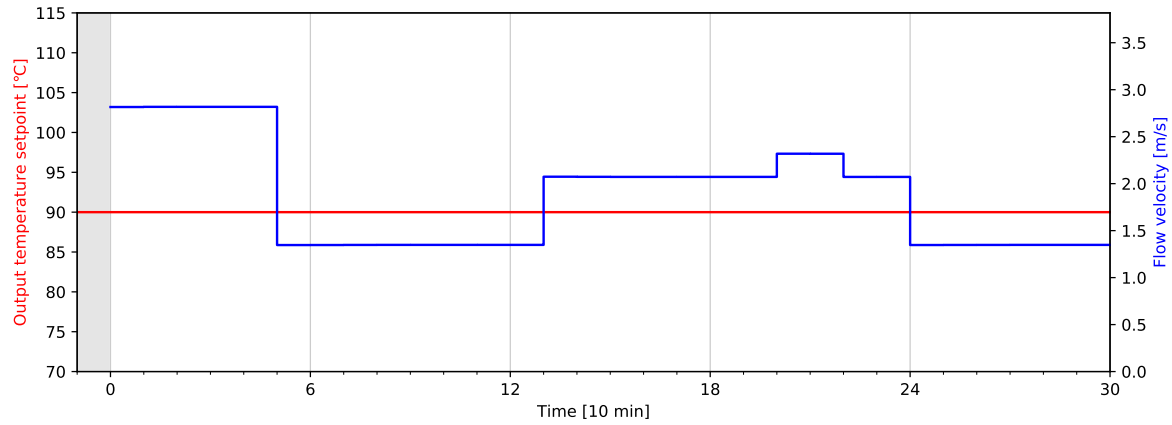
The vertical axis shows flow velocities (blue) and set point output temperature (red) of the producer. The horizontal axis is time in blocks of 10 min. Everything left of t = 0 min is history and can thus no longer be controlled. The maximum flow velocity for DN600 pipes is 3.0 m/s.

Satisfying Demand by Increasing Mass Flow For this first example the demand is dynamic and is shown in Figure 2.5a. The supply temperature is kept constant at 90 °C, which means temperature changes have no effect on the heating power for consumers. The resulting mass flows can be seen in Figure 2.5b. When demand drops (t=50 min), the flow velocity in the network drops as well. When the demand increases again at t=130 min, the flow velocity increases again as well. In reality there is a small delay between the change in demand and the change in mass flow at a producer, but this effect is several orders of magnitude smaller than the time scale at which the optimisation works.

[4] The number after DN indicates the diameter of the pipe in mm, in this case 600 mm



(a) Consumer heat power demand



(b) Constant supply temperature of 90 °C and the resulting mass flows

Figure 2.5: Grid frequency response with constant supply temperature and dynamic demand from the point of view of the producer

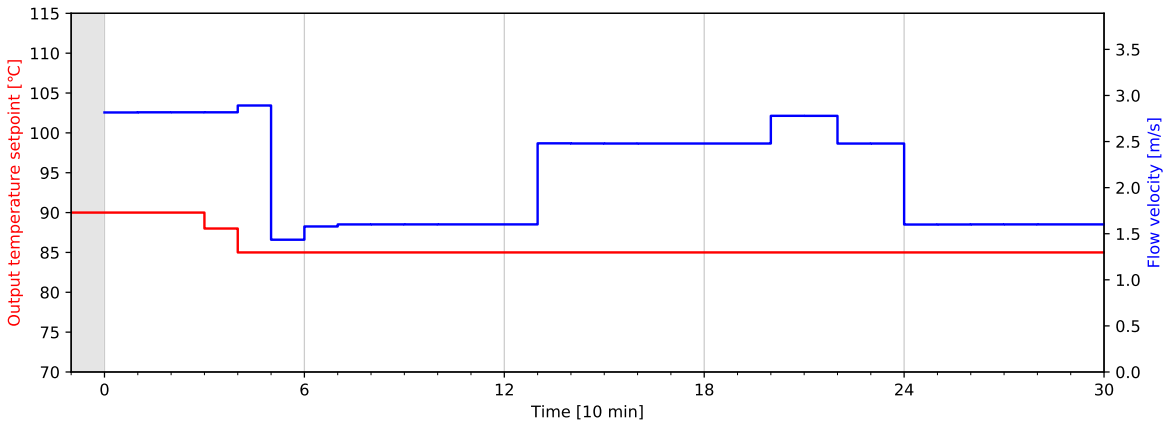


Figure 2.6: Changes in supply temperature lead to delayed changes in mass flow (variable demand)

Changes in Demand Lead to Instant Mass Flow Changes, Changes in Supply Temperature Do Not In this second example, both the demand and the supply temperature are varied over time. The demand is the same as in the previous example (Figure 2.5a).

The resulting mass flows are illustrated in Figure 2.6. Lowering the supply temperature, leads to higher mass flows in the network, but since there is a propagation delay, it does not happen instantly. When the supply temperature is dropped the first time (t = 30 min), nothing happens yet to the mass flow. Only in the next time block (t=40 min) does the consumer compensate for the lower supply temperature by increasing the mass flow.

Then, at t = 50 min, the heating power demand drops and the mass flow drops immediately as well. At t = 70 min the mass flow is increased a little, because at t = 60 min some warmer water was still left in the network, which meant less mass flow was needed to provide the same amount of heating power to the consumer.

High Temperatures Lead to Slower Frequency Response This final example shows that high supply temperatures have a longer lasting effect in the network than low temperatures. This effect is shown in Figure 2.7. At $t = 0$ min a block of water with temperature 105 °C is put into the network. As this water has more heat in it, the required mass flow for the consumers goes down. At $t = 180$ min a block of only 75 °C is put into the network. At time $t = 190$ min the mass flows starts to increase to compensate.

The higher temperature water has an effect for 90 min while the lower temperature water has an effect for only 60 min. This is because the total mass of water in the network does not change, but the mass flow in the network does change. As a low supply temperature leads to higher mass flows, it will take less time for the cooler water to pass through the consumer than it did for the warmer water.

2.2. Physics

To give a better understanding of why the effects described in the previous section occur, this section describes the relevant physics concepts.

Flow is the movement of a medium through a channel. In this case the medium is water and the channel are the pipes in the DHN. In this problem, two types of flow are used: mass flow and flow velocity. This is described in Section 2.2.1.

The second relevant concept is heat. This is a form of energy that causes a medium to have a certain temperature. Heat dissipates when there is a temperature differential between two points. This causes the water in the DHS to lose energy to the environment, which cools it down before it reaches consumers. Section 2.2.2 describes relevant heat equations and heat exchanger computations.

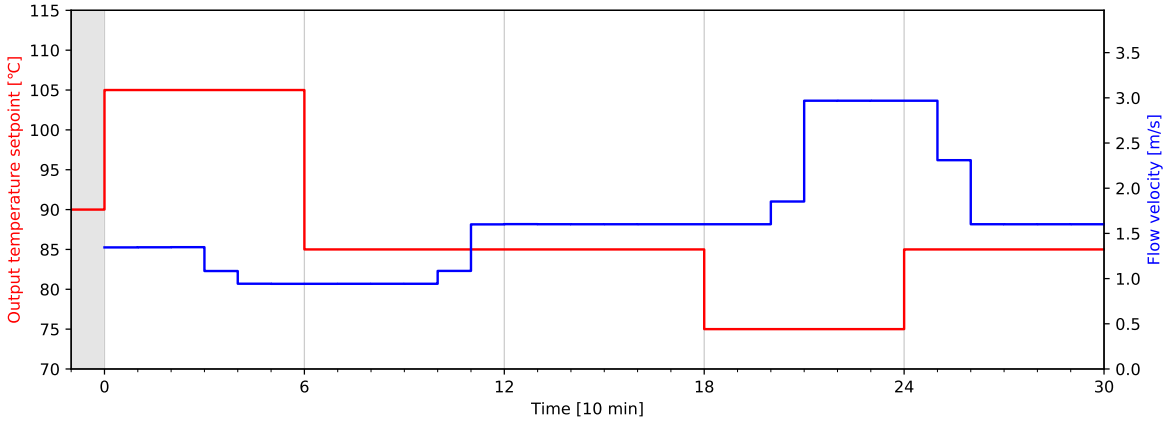


Figure 2.7: Temperature effect duration (constant demand)

2.2.1. Flow

Mass flow ($\dot{m}$ [kg/s]) is the amount of water weight that flows through a cross section of a pipe every second. At every point in a pipe the mass flow is the same. This is because the propagation of changes in the mass flow are effectively instant. The change in mass flow propagates as a pressure wave through the pipe at a speed of more than 1400 m/s[5], which means the change is propagated through a typical DHS in less than 10 s.

Some parts of the optimisation are easier to define when flow velocity (v [m/s]) is used instead of mass flow. Mass flow can be converted to flow velocity using Equation 2.3 where ρ is the density of water and A is the cross section area of the pipe in the direction orthogonal to the flow direction.

$$v = \frac{\dot{m}}{\rho A} \tag{2.3}$$

Energy

To create mass flow, energy is needed. This energy comes from electricity that is used by pumps to create a pressure difference. The amount of energy a pump uses at any time to create pressure can be derived using Equation 2.4.

$$\dot{W}^{\dot{p}} = \frac{\dot{m}\Delta P}{\rho \eta^{\dot{p}}} \tag{2.4}$$

where ΔP is the pressure difference between the inlet and outlet of pump $\dot{p}$, and $\eta^{\dot{p}}$ its efficiency.

From this equation is seems like the amount of energy can grow unbounded, but this is not true. Mass and pressure difference are related to each other, the combination of which results in a characteristic curve for the pump. This relation can be measured, but if this characteristic and either of two quantities is known, the third can be determined. An example characteristic is shown in Figure 2.8.

2.2.2. Heat

At the core of the problem is the heat balance equation 2.1 where c is the specific heat of the medium (in this case water) and ΔT is the difference in temperature between the inlet and the outlet of, e.g. a pipe or heat exchanger.

$$\dot{Q} = c\dot{m}\Delta T \tag{2.1}$$

where $\Delta T = T_i - T_o$.

[5]The speed of sound in water

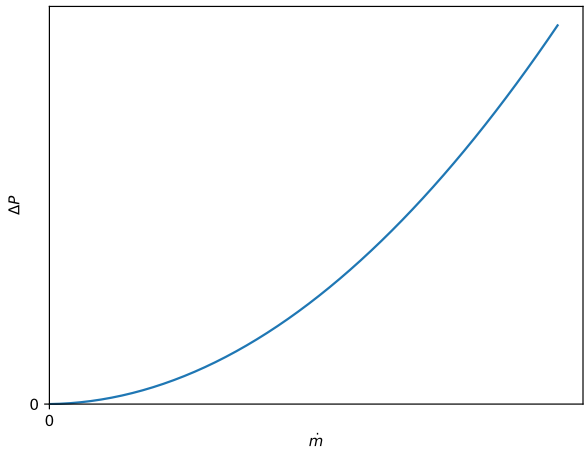


Figure 2.8: Pump mass flow versus pressure characteristic

Production

Using this formula, it is possible to calculate how much heat a producer has to generate. The mass flow is determined from the control variables and the consumer demands. The temperature difference is simply the difference between the return temperature from the grid and the set point supply temperature. Because the type of producers considered in this problem can usually increase production much faster than the time scale of the optimisation, it is assumed producers can respond to changes in mass flow instantly. In reality, they will temporarily output water of a lower temperature, but this is negligible for the optimisation.

Heat Losses

Because of the thermal conductivity of the pipes in the network, heat is lost during transport. This means that heat moves from the water to the pipes and their surroundings. This should be minimised as this reduces the efficiency of the DHS.

A high temperature differential between two points will cause heat to move faster between these two points. This means that by minimising the temperature in the pipes, the amount of heat lost to the environment is also minimised.

The temperature difference with respect to the environment is given by Newton's law of cooling:

$$\Delta T(t) = \Delta T_0 \exp(-t/\tau) \tag{2.5}$$

where $\Delta T_0 = T_i - T_{\text{env}}$ and $\tau = A\rho cR$. Here T_{env} is the environment temperature and R is the thermal resistance of the pipe. Note that τ is constant, but different for every type of pipe. Newton's cooling law thus gives how much the water has cooled down after t time in the pipe.

To minimise the amount of heat lost, either T_i or the amount of time water spends in the pipe could be minimised. It would also be better to have a high thermal resistance, but this is determined during the design of the DHS, well before this optimisation comes into play.

Heat Exchangers

To consume heat from a DHS or to move heat from the primary to the secondary network in a HTS, heat exchangers are used. These devices move heat from one medium to another without mixing them. A heat exchanger has 2 inlets and 2 outlets; a primary and secondary of both. Heat is moved from the primary side to the secondary side through a contacting wall between the two fluid streams. This contacting wall is made of a material with a low heat resistance so heat can easily pass through it.

In a typical DHS so-called counter-flow heat exchangers are used, where the primary flow flows in the direction opposite to the secondary flow. A diagram of this type of heat exchanger is shown in Figure 2.9. The outer mantel is the primary side as warm water flows into the heat exchanger here.

Rewriting Equation 2.1 to 2.6, the amount of heat required at the secondary side and the secondary supply (outlet) and return (inlet) temperatures, the secondary mass flow can be determined.

$$\dot{m} = \frac{Q}{c\Delta T} \tag{2.6}$$

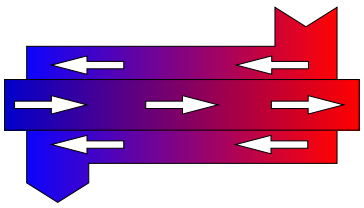


Figure 2.9: Counter flow heat exchanger diagram

In Equation 2.7 the primary return temperature for a heat exchanger is determined. The factor α is given in Equation 2.9, but a more detailed description is given in [9, section 2.6.1]. The factor α is determined using Equation 2.8. T_{pi} and T_{si} are respectively the primary and secondary inlet temperatures.

$$T_{po} = \alpha T_{pi} + (1-\alpha)T_{si} \tag{2.7}$$

$$LMTD = \frac{(T_{pi} - T_{po}) - (T_{so} - T_{si})}{\ln(T_{pi} - T_{po}) - \ln(T_{so} - T_{si})} \tag{2.8}$$

$$\alpha = \frac{T_{pi} - T_{so}}{T_{pi} - T_{si}} \exp\left(-\frac{UA}{c\dot{m}_s}\left(\frac{\dot{m}_s}{\dot{m}_p} - 1\right)\right) \tag{2.9}$$

$$UA = \frac{k}{\dot{m}_p^{-q} + \dot{m}_s^{-q}} \tag{2.10}$$

The following variables at the heat exchanger are needed in the logarithmic mean temperature difference (LMTD) method: primary and secondary mass flow, primary and secondary inlet temperatures, and the secondary target outlet temperature. Unfortunately, because the primary mass flow is exactly what needs to be determined for the simulation, no closed form of this equation exists and an iterative process is required to solve it.

In addition to these variables, several other constant properties of the heat exchanger are needed: the contacting wall (plate) area A, maximum primary mass flow $\dot{m}_{\text{max}}$ and k and q factors which can be determined experimentally.

Heat Exchanger Function Shapes

To make it clear what the heat exchanger functions look like exactly, they will be described here using an example heat exchanger.

Taking a typical heat exchanger with the following parameters:

- k factor = 6000
- q factor = 0.8
- Plate surface area $A = 1.5\,\text{m}^2$
- Maximum primary mass flow $\dot{m}_{p_{\text{max}}} = 300\,\text{kg/s}$
- Secondary return temperature $T_{si} = 45\,°\text{C}$
- Target secondary supply temperature $T_{so} = 70\,°\text{C}$

and following input domains:

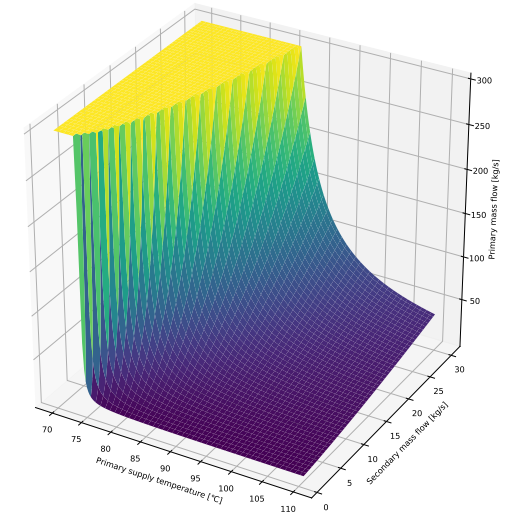
- Primary supply temperature $T_{pi} = 70\,°\text{C}$ to $110\,°\text{C}$
- Secondary mass flow $\dot{m}_s = 0\,\text{kg/s}$ to $30\,\text{kg/s}$

Depending on the secondary mass flow, which is linearly dependent on the demand, a minimal temperature is needed to deliver the requested heating power. If the primary supply temperature drops below

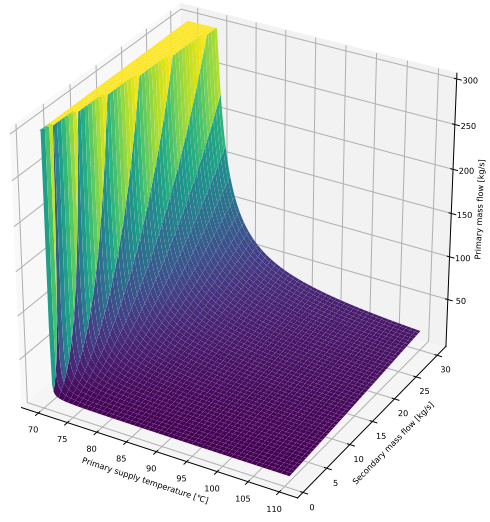
this temperature, the heat exchanger can no longer transfer enough power to its secondary side. This boundary can clearly be seen in Figure 2.10.

At this boundary, the primary mass flow (Figure 2.10a) goes to its maximum value to deliver as much heat as possible. The primary return temperature (Figure 2.10e) drops more steeply. The secondary supply temperature (Figure 2.10g) can no longer be kept at the target 70 °C and starts dropping. Finally, the delivered heat power (Figure 2.10c) can no longer be kept at the demand and the linear dependency no longer holds.

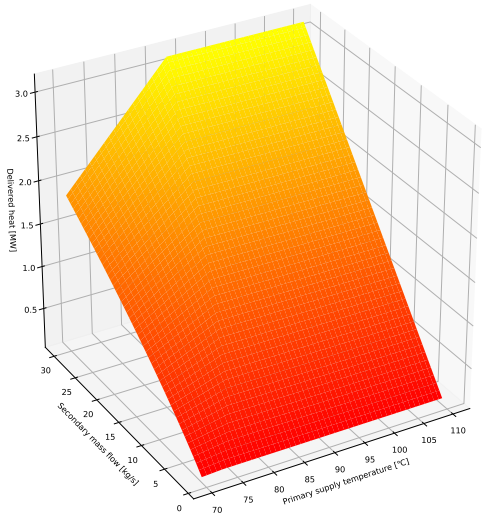
As can be seen in Figures 2.10b, 2.10d, 2.10f, and 2.10h, doubling the plate area A, drastically changes how efficiently the heat exchanger works. The boundary is shifted toward the lower bound of the primary supply temperature, i.e. the heat exchanger can deliver the same amount of heating power with a much lower primary supply temperature. It can also be seen that the primary return temperature will be much lower, which means the heat exchanger is more efficient. Note that doubling the k factor would have the exact same effect.



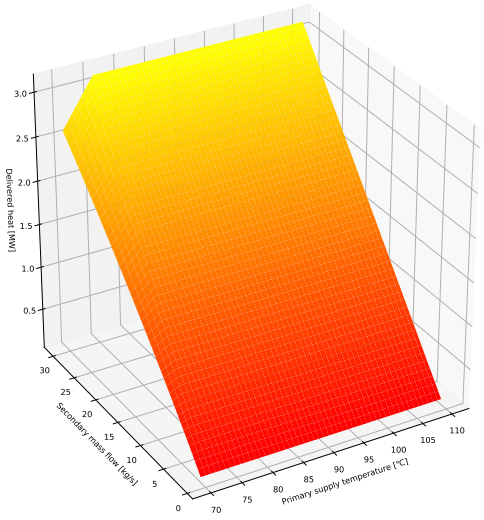
(a) Primary mass flow, $A = 1.5\,\mathrm{m}^2$



(b) Primary mass flow, $A = 3\,\mathrm{m}^2$

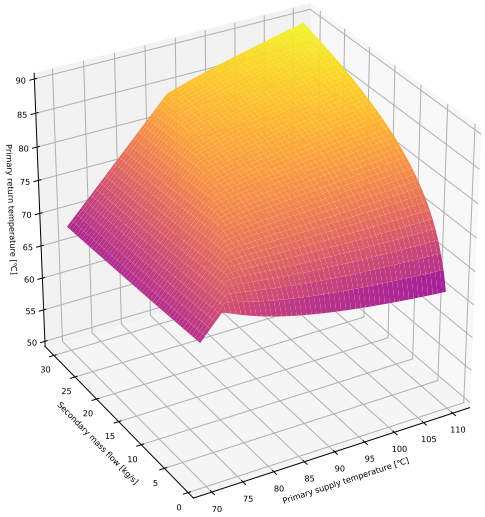


(c) Delivered heat, $A = 1.5\,\mathrm{m}^2$

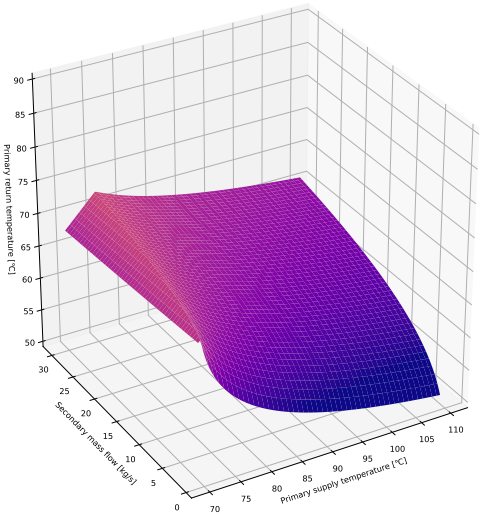


(d) Delivered heat, $A = 3\,\mathrm{m}^2$

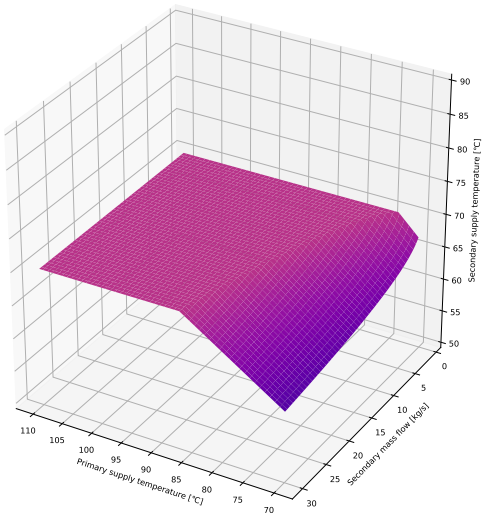
Figure 2.10: Heat exchanger functions for a heat exchanger with $k = 6000$, $q = 0.8$, $\dot{m}_{p\,\mathrm{max}} = 300\,\mathrm{kg/s}$, $T_{si} = 45\,^{\circ}\mathrm{C}$, $T_{so} = 70\,^{\circ}\mathrm{C}$. Figures at the top have $A = 1.5\,\mathrm{m}^2$, on the bottom $A = 3\,\mathrm{m}^2$. Note that the figures are rotated along the Z axis to show the functions' shapes better. Colours indicate the value on the Z axis.



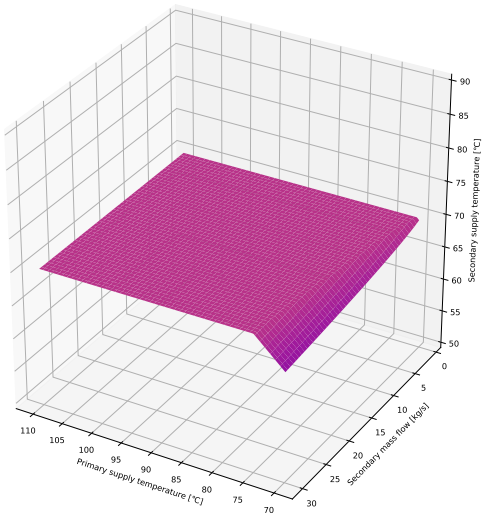
(e) Primary return temperature, $A = 1.5\,\mathrm{m}^2$



(f) Primary return temperature, $A = 3\,\mathrm{m}^2$



(g) Secondary supply temperature, $A = 1.5\,\mathrm{m}^2$



(h) Actual secondary supply temperature, $A = 3\,\mathrm{m}^2$

Figure 2.10: Heat exchanger functions (continued). Note that the figures are rotated along the Z axis to show the functions' shapes better.

3

Problem Description

This chapter gives a description of the optimisation model and the elements added or removed from the DHS problem as it was described in Chapter 2. In Chapter 2 the fundamental concepts behind the operation of DHSs were described. This chapter focusses on the actual optimisation problem, by describing the control variables, the (artificial) constraints of the problem, the problem complexity and the corresponding fitness function.

First, Section 3.1 gives a short description of what optimisation is. Section 3.2 gives a description of the control variables in this model and how they are used in the simulator behind the fitness function. Then, Section 3.3 gives a description of all constraints in the DHS problem, including artificial constraints added to improve the optimisation. The complexity of this problem is discussed in Section 3.4, which consists of an explanation of why this problem is not linear and of a convexity counter example.

Section 3.5 then describes the fitness objective for the DHS problem. Then, Section 3.6 describes an artificial constraint added to the optimisation that solves a problem for the optimisation and makes it possible to calculate an easy lower bound for the problem. Finally, using this new constraint, Section 3.7 describes a lower bound for this problem, by considering several of its facets.

3.1. Optimisation

Optimisation is the practice of finding the best possible solution to a problem from a (possibly infinite) solution space. More formally, Equation 3.1 minimises quality measurement function f by finding the solution $\mathbf{x}$ which minimises f. Equation 3.2 defines the solution space using function g to indicate that g applied to some $\mathbf{x}$ should be greater than or equal to some value vector $\mathbf{a}$.

$$\min_{\mathbf{x} \in X} f(\mathbf{x}) \tag{3.1}$$

$$\text{s.t. } g(\mathbf{x}) \geq \mathbf{a} \tag{3.2}$$

A simple example of a minimisation problem would be: find a value for $x \in \mathbb{N}$ that minimises x such that $x - 10 \geq 4$. In this case, f is the identity function, which simply returns the given x. The solution space is defined by all natural numbers larger than or equal to 14, because $g = x - 10$ and $a = 4$. The optimal solution is 14, as $g(13) = 3 \ngeq 4$, and $g(15) = 5 > 4$, which means 14 is more optimal.

Approaches

There are different possible approaches to optimise a problem, depending on the given problem.

Some problems are ”easy” to optimise, namely problems in the complexity class P. An example is the shortest path problem, which can efficiently be solved using, among others, Dijkstra’s algorithm.

The “hard” problems, from the class NP - P and up, include the travelling salesperson problem (TSP), for which no efficient algorithm exists (assuming P≠NP) if a guaranteed optimal solution is needed

within a reasonable amount of time for any instance. Problems like these are often solved using other methods, like heuristics, metaheuristics, or machine learning.

These problem classes encompass combinatorial optimisation problems, in which all decision variables have discrete values. There are also real-valued optimisation problems, which consist of finding the optimal values for real-valued decision variables. This type of problems are often solved using mathematical optimisation like derivative based optimisation, linear programming, non-linear optimisation or metaheuristics, like evolutionary strategies.

As the DHS problem is continuous, it might be solved using one of the methods.

3.2. Control Variables

As described in Section 2.1.2, three variables are important in calculations on how a DHS behaves: the mass flows, the consumed heat and the temperatures. When two are given, the third can be determined. In the simulation in this model, demand and temperatures are known and the mass flows need to be determined.

There are several methods of controlling a DHS simulator. One of the methods was described earlier in Section 2.1.2, in which the supply temperatures and pumping power is controlled, resulting in mass flows and consumer supply temperatures. This method is more difficult to simulate though, as complex pressure calculations are required for this. A different approach was taken based on the node flow method from Benonysson et al.

In this approach, consumers are leading. They are given their primary supply temperature and demand from which they can compute their primary return temperature and primary mass flow.

How water is routed through the DHN can now no longer be controlled by pressure, but by the mass flows resulting from the consumers. There are some areas in the network where more information is needed than just the consumer mass flows to determine how water is routed. This information comes from a decision variable that controls what fraction of the incoming flow goes where through the use of valves.

In Figure 3.1 a DHS with two producers and a controlled join is shown. There is no secondary network, so both consumers are directly connected to the primary network. The join is controlled to split the total mass flow from the consumers between the producers. This control consists of a parameter $\alpha \in [0, 1]$ which decides what fraction of the outlet mass flow comes from which producer. It would intuitively make more sense to have controlled splits, however the simulation is easier when joins are controlled through the control variables instead. This change can be made without loss of generality.

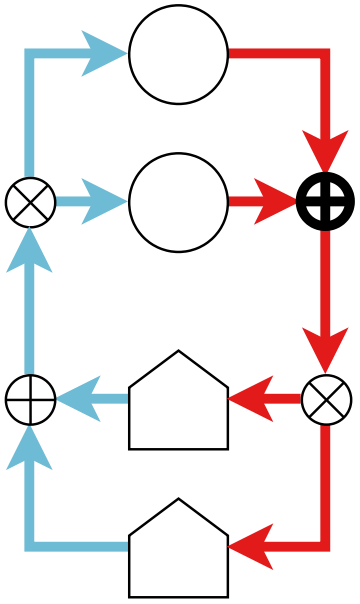


Figure 3.1: DHS with multiple producers, consumers and a controlled join

3.3. Constraints

The DHS problem has several constraints which are described in this section. The first subsection describes constraints that come from physical limitations on DHSs, e.g. the maximum mass flows through the network. The second section describes artificial constraints which are added to improve the optimisation.

3.3.1. Physics-Based Constraints

Fulfil Demand

The most important constraint to fulfil is that demand must be fulfilled at all times. This is a legal requirement in The Netherlands.

To be able to fulfil this demand it must be possible to deliver enough heating power to the consumers' secondary heat exchanger side. This can be done by either increasing the primary mass flow of the consumer or by increasing the primary supply temperature.

This constraint can be described by the following inequality:

$$Q_t^c \geq Q_t^{\text{dem},c} \qquad \forall c \in C, t \in T \tag{3.3}$$

which states the delivered power Q_t^c to consumer c at time t should be at least as much as the power $Q_t^{\text{dem},c}$ demanded by consumer c at time t for all consumers $c \in C$ at all times $t \in T$

Maximum Edge Flow Velocity

Every edge (pipe) in the network has a maximum amount of water that can flow through it every second. This mass flow can be translated to flow velocity using Equation 2.3.

Exceeding this limit can damage pipes or other equipment in the network. As every consumer in the network determines their required primary mass flows, these mass flows will eventually come from a common source pipe. This compound effect can lead to needing to exceed the maximum flow velocity in a source pipe. Should this happen, the only solution would be to increase the temperature of the water in the pipes so consumers will need a lower primary mass flow.

This constraint can be described by the following inequality:

$$v_t^e \leq v_{\text{max}}^e \qquad \forall e \in E, t \in T \tag{3.4}$$

which states that the flow velocity in pipe e at time t should be lower than the maximum flow velocity for pipe e for all pipes $e \in E$ at all times $t \in T$.

Minimum Supply Temperature

In addition to needing enough heat to fulfil demand, the supply temperature should also be higher than 70 °C to prevent bacterial growth in the network. This is also a legal requirement in The Netherlands, thus it might be possible this constraint could be omitted in other scenarios.

This constraint can be described by the following inequality:

$$T_{pi}^c \geq 70 \qquad \forall c \in C \tag{3.5}$$

Which states the primary supply temperature ${T_{pi}}^c$ for consumer c should be higher than 70 °C for all consumers $c \in C$.

Maximum Production Temperature Differential

The final physics constraint has to do with the production of heat. Heat producers should not need to change their output temperatures too much within a given time frame. Doing this would put excessive stress on the network and production equipment.

This can be expressed by the following inequality:

$$-T_{\nabla_{\text{max}}}^p \leq T_{t-1}^p - T_t^p \leq T_{\nabla_{\text{max}}}^p \qquad \forall p \in P, t \in T \tag{3.6}$$

which states that the temperature difference for producer p between time $t-1$ and time t should not be greater than the maximum temperature gradient $T_{\nabla_{\text{max}}}^p$ for all producers $p \in P$ at all times $t \in T$.

Note that T_0^p is the output temperature of producer p before the optimisation horizon starts. This means the output temperature before the start of the optimisation horizon is also a constraining factor.

3.3.2. Optimisation-Based Constraints

Energy Stored in the Network

As will described in Section 3.6, a constraint should be added to the optimisation which enforces a lower limit on the amount of energy present in the network at the end of the optimisation horizon.

This can be done by calculating how much energy is stored in the network at the end using equation 2.1:

$$Q_t = cm\Delta T \tag{2.1}$$
$$= c\sum_{e\in E} m^e T_t^e \tag{3.7}$$
$$= c\sum_{e\in E} \rho A^e L^e T_t^e \tag{3.8}$$

where A^e and L^e are the cross sectional area and length of pipe e respectively and T_t^e is the absolute temperature of the water in pipe e at time $t \in T$.

Using this we can then create the constraint:

$$c\rho \sum_{e\in E} A^e L^e T^e_{t=\text{end}} \geq \Omega \tag{3.9}$$

where Ω is the amount of energy that should be present in the network at the end of the optimisation.

3.4. Problem Complexity

This section describes the problem characteristics which make this problem hard to optimise with conventional optimisation methods. The first obstacle is the non-linearity of this problem. This is explained in Section 3.4.1. The second obstacle is the non-convexity of the problem, which is described in 3.4.2.

3.4.1. Non-Linearity

There are several factors in this problem which make the problem non-linear.

Heat Exchangers

The first is the heat exchanger functions, which are not smooth over their entire domain. Certain sections of the domain are smooth, but these sections are not always linearly dependent on the domain. The heat exchanger functions have been explored in Section 2.2.2.

Heat Losses

The second factor which causes non-linearity are the heat losses in the DHN. As described in Section 2.2.2, heat loss is calculated using Newton's cooling law (Equation 2.5), which contains an exponential dependence on the time water spends in the pipe. This time can be calculated using the mass flow in the pipes, which is calculated using the demand and the supply temperatures in the network.

As supply temperatures are decision variables, the input variable for the exponential Newton's cooling law depends on the decision variables.

3.4.2. Non-Convexity

A problem is relatively easy to optimise when it is convex. A problem is convex when both the goal function and the solution space are convex. Convex problems can be optimised using hill climbing, as there are no local optima to get stuck in. Unfortunately, the DHS problem is not convex. To show this, we show the solution space is not convex by means of a counter example.

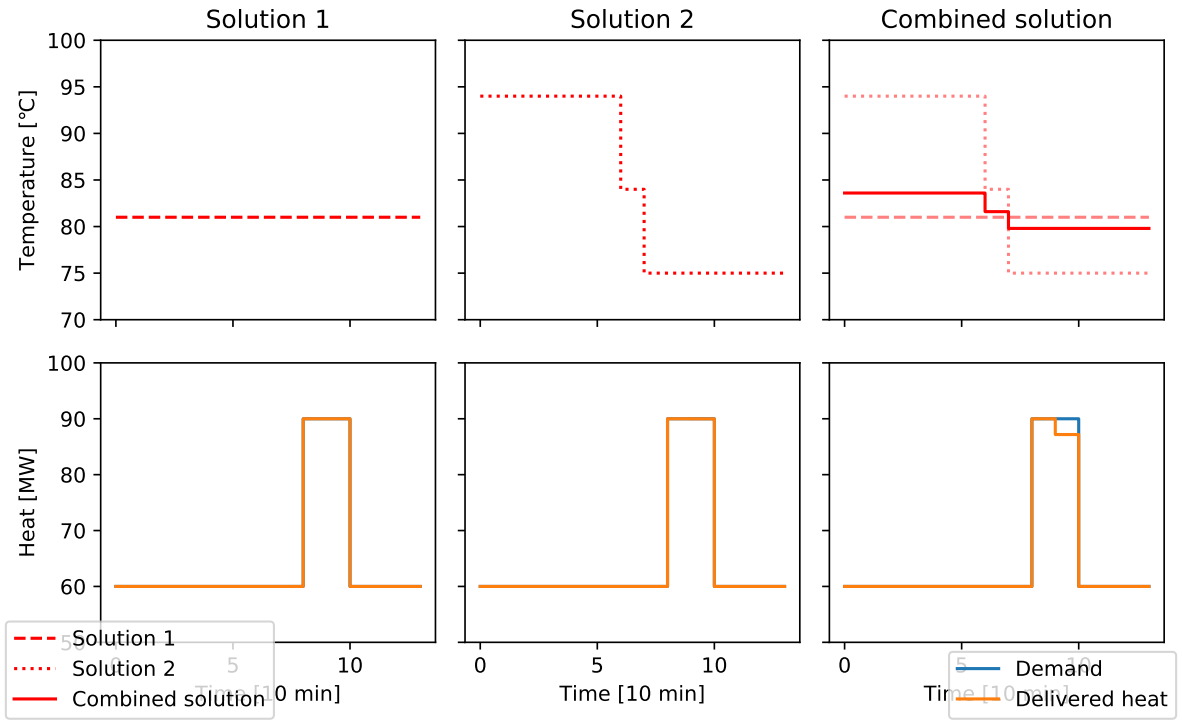


Figure 3.2: Three grid settings with their corresponding heat delivery. The third settings proves non-convexity.

A set S is convex if:

$$\forall \mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_r} \in S, \forall \lambda_k \in [0,1], \Sigma_{k=1}^{r} \lambda_k = 1. \quad \sum_{k=1}^{r} \lambda_k \mathbf{u_k} \in S \tag{3.10}$$

We use this definition to show that there exist $\mathbf{u_1}$, $\mathbf{u_2}$ and λ_1, λ_2 for which the convexity condition does not hold. In this case, the set S is the set of all valid solutions for the DHS problem. To show that this space is not convex, we make use of the constraint that enforces all demand to be fulfilled.

The grid from the examples in Section 2.1.2 is used again to show this. The demand for the consumer in the grid is shown in Figure 3.2 in the bottom left subplot.

For this grid $\mathbf{u_1}$, $\mathbf{u_2}$ are temperature settings for every time step. We choose a value of 81 °C for all time steps in $\mathbf{u_1}$.

For $\mathbf{u_2}$ we choose a starting temperature of 94 °C and at time t=60 min start dropping it down to 75 °C as it will no longer need to be as high to fulfil the demand in the peak. Note that this is before the higher demand peak has started. The demand peak will mostly be fulfilled by heat stored in the network.

These two solutions and their effects on the delivered heat are shown in the two left columns in Figure 3.2. Both solutions store enough heat in the network to fulfil demand at all times, which means they are valid solutions.

When the two solutions are combined linearly with $\lambda_1 = 0.8$ and $\lambda_2 = 1 - \lambda_1 = 0.2$ to create a new solution, the amount of heat stored in the network is no longer enough to fulfil the demand. This is shown in the third column. In the last time block of the demand peak, all water with enough heat in it has been consumed before the end of the peak and not all demand can be fulfilled.

This shows that the convexity condition 3.10 does not hold for all possible solutions and λ_r values, which means the DHS problem is not convex.

3.5. Fitness Function

A fitness function is a function that takes a solution and returns the fitness of this solution. The fitness of a solution is based on two factors: the feasibility and the desirability. A solution is feasible when it violates no hard constraints.

The desirability depends on what is being minimised. In this research the operating costs are minimised, but something else would be possible as well, like CO_2 emissions. If the operating costs of a solution are lower than those of another solution, the first will be more desirable, assuming it is a feasible solution.

3.5.1. Operating Costs

As described in Section 2.1.2, the operating costs (Equation 2.2) depend on the heat produced by the producers and the energy used to pump water through the DHN. As the operating costs is the main focus of the minimisation, this is an important part of the fitness function.

$$c_{\text{op}}(\mathbf{T}, \dot{\mathbf{W}}) = \delta \sum_{t \in T} \left(\sum_{p \in P} c_t^p Q_t^p(\mathbf{T}_t^p) \right) + \left(\sum_{\dot{p} \in \dot{P}} c_t^{\dot{p}} \dot{W}_t^{\dot{p}} \right) \tag{2.2}$$

3.5.2. Constraint Violations

For metaheuristic approaches it is not always possible to incorporate constraints into the optimisation directly. One possibility is to encode solution such that it becomes impossible to break certain constraints, but this is not always possible.

Another approach to add the constraints to the optimisation, is to add the constraint violations to the fitness values. As the optimisation attempts to minimise the objective value, it naturally minimises the constraint violations as well.

The penalty for every constraint type can be determined by either counting the number of constraint violations of that type or by using the relative violation. The relative violation method is preferred, as this allows a degree of ordering to be imposed on the solutions. A solution that never fulfils demand is worse than a solution that fulfils 90% of the demand, even if both solutions are infeasible.

It is also possible to define how much more important one constraint is than the other, by adding a punishment factor to every violation. A higher factor will lead to optimisers trying to solve one constraint violation earlier than others, as most progress can be made by fixing the more strongly punished violations first.

This means that the feasibility function g for the DHS problem is defined by the constraints described in the previous section and the vector $\mathbf{a}$, the bounds on the constraints. The total violation term of the fitness function is define as follows:

$$\hat{g}(\mathbf{x}) = \psi \cdot \max\left(\mathbf{a} - g(\mathbf{x}), \mathbf{0}\right) \tag{3.11}$$

where ψ represents the punishment factor vector for every constraint, $\max$ is the element wise max operator and $\cdot$ the inner product operator. $\mathbf{a}$ is determined from the DHS model, $\mathbf{x}$ represents the decision variables like output temperatures and valve settings, and g consists of the relevant output values of the simulator, like edge velocities, delivered heat, etc.

3.5.3. Complete Fitness Function

By adding the operating costs, constraint violations and penalties together the complete fitness function for a solution $\mathbf{x}$ is given by:

$$f(\mathbf{x}) = c_{\text{op}}(\mathbf{x}) + \hat{g}(\mathbf{x}) \tag{3.12}$$

3.6. Energy Stored in the Network

As previously described in Section 2.1.1, the network of the DHS consists of pipes that transport water. As it takes time for water to reach consumers, heat is temporarily "stored" in the network until it is removed by consumers.

This storage effect results in two other effects that make it difficult to define the optimisation problem:

1. Optimisations can make most of their savings by decreasing heat production at the end as much as possible;
2. It is hard to find an easy to calculate lower bound.

To solve these problems, a constraint is added to the optimisation problem:

> The amount of energy in the DHS network at the end of the optimisation horizon is larger than some given constant.

3.6.1. Termination Focused Optimisation

As the optimisation is not constrained at the end of the horizon by consumer demand that will need to be fulfilled, optimisations will find solutions that turn down production as much as possible at the end. This often leads to many savings at the end of the horizon at the expense of more costs at the start. This would lead to unnecessarily high costs when the found solutions are applied in practice.

Adding the new constraint, forces optimisations to find solutions that produce enough heat at the end of the optimisation horizon for future consumption, which means they can no longer make big savings by producing very little heat at the end.

3.6.2. Lower Bound

Before the start of the optimisation horizon, heat is already stored in the network. Before consumers receive heat produced during the optimisation horizon, they first consume the heat that was already stored in the network.

Because of this, it is not possible to simply take the total demand as a lower bound on the heat that should be produced during the horizon. Subtracting the heat stored in the network from the total demand is also not possible, as the heat stored in the network is larger than the total demand, which would result in a lower bound of 0. It is also not possible for consumers to remove all heat from the water in the network, due to the physics of a heat exchanger.

By constraining the amount of heat stored in the network at the end of the horizon from below, a much smaller amount of energy remains that can be used to fulfil consumer demand.

A more detailed description on how this new constraint is used to calculate the lower bound is given in Section 3.7.1.

3.7. Lower Bound

To be able to assess how good a solution for the problem is and how much improvement is theoretically still possible, a lower bound can be used. This lower bound is a single number that is the absolute lowest amount of cost that can be made without violating constraints. If settings are given with costs lower than the lower bound, it can immediately be concluded that these settings are infeasible. However, because several unrealistic assumptions are made to enable the determination of the lower bound, it is not necessarily true that a feasible solution exists that is as good as this lower bound.

For the DHS problem, the following lower bound can be determined:

$$c_{\text{op}} \geq \text{GreedyHeatProductionDivision}\left(Q_{\text{produced}}\right) \tag{3.13}$$

$$\geq \text{GreedyHeatProductionDivision}\left(\max\left(0, Q_{\text{consumed}} + Q_{\text{loss}}^{\geq 0} + \gamma\right)\right) \tag{3.14}$$

where Q_{consumed} is the total consumption over the entire horizon, $Q_{\text{loss}}^{\geq 0}$ a lower bound on the losses in the network, γ the amount of additional heat that needs to be produced to have enough energy stored

in the network at the end of the horizon, and GreedyHeatProductionDivision a procedure that greedily assigns heat production to the cheapest producers.

The following subsections give a detailed description of how the different elements of the lower bound are determined and why. The first element is discussed in Section 3.7.1, which is the γ term representing how much additional energy needs to be produced to have a minimum amount of energy stored in the network at the end of the horizon. The second component, Q_{consumed}, is the total amount of heat that is consumed, which is discussed in Section 3.7.2. The third component, $Q_{\text{loss}}^{\geq 0}$ is the minimum amount of heat loss in the network and is discussed in Section 3.7.3. Finally, the GreedyHeatProductionDivision procedure and the reasoning behind it is discussed in Section 3.7.4. Then, Section 3.7.5 combines all these components together into the lower bound given above.

3.7.1. Energy Present at the End of the Horizon

This section explains how a lower bound can be determined on the amount of energy that needs to be produced to maintain a minimal amount of energy in the network at the end of the horizon.

Given the network energy balance equation

$$Q_{\text{start}} + Q_{\text{produced}} = Q_{\text{consumed}} + Q_{\text{loss}} + Q_{\text{end}} \tag{3.15}$$

which states the heat in the network at the start of the horizon (known) and the heat produced during the horizon is equal to the heat consumed during the horizon, the heat lost to the environment and the heat stored in the network at the end of the horizon combined.

Using the constraint from Section 3.3.2 and the assumption from 3.6:

$$c\rho \sum_{e \in E} A^e L^e T^e_{t=\text{end}} = Q_{\text{end}} \geq \Omega \tag{3.9}$$

where Ω is a constant amount of energy chosen beforehand, we can make the following argument.

$$\begin{aligned}
Q_{\text{start}} + Q_{\text{produced}} &= Q_{\text{consumed}} + Q_{\text{loss}} + Q_{\text{end}} && && (3.15)\\
Q_{\text{start}} + Q_{\text{produced}} &\geq Q_{\text{consumed}} + Q_{\text{loss}} + \Omega && \text{using 3.9} && (3.16)\\
Q_{\text{produced}} &\geq Q_{\text{consumed}} + Q_{\text{loss}} + \Omega - Q_{\text{start}} && && (3.17)\\
Q_{\text{produced}} &\geq Q_{\text{consumed}} + Q_{\text{loss}} + \gamma && \gamma = \Omega - Q_{\text{start}} && (3.18)\\
& && && (3.19)
\end{aligned}$$

Since Ω and Q_{start} are known beforehand, γ is a constant. This term represents the amount of energy that needs to be added after heat has been removed from the network by consumption and losses to satisfy the lower bound Ω. As γ and Q_{consumed} are known, we can determine a lower bound on the heat production.

$$Q_{\text{produced}} \geq \max\left(0, Q_{\text{consumed}} + Q_{\text{loss}} + \gamma\right) \tag{3.20}$$

In practice, Q_{loss} is not known beforehand, but $Q_{\text{loss}} \geq 0$ always holds, thus we can replace Q_{loss} with $Q_{\text{loss}}^{\geq 0}$ which still results in a valid (albeit less precise) lower bound.

The produced heat is constrained to be larger than or equal to 0, since producers are unable to extract heat back from the network.

To show the lower bound is correct we distinguish three cases:

$Q_{\text{start}} > \Omega$ i.e. more energy is stored at the start than at the end. We get: $\gamma = \Omega - Q_{\text{start}} < 0$. This makes sense as Q_{end} may be lower than Q_{start} and as such not all energy removed from the network needs to be refilled by the producers.

If $-\gamma \geq Q_{\text{consumed}} + Q_{\text{loss}}$, no energy will have to be produced in theory, thus the lower bound is equal to 0. In practice, energy will need to be produced to prevent breaking any hard constraints.

$Q_{\text{start}} = \Omega$ i.e. there should be at least as much energy in the network at the end as at the start. We get: $\gamma = \Omega - Q_{\text{start}} = 0$, thus all energy consumed or lost needs to replaced by the producers. In this case the lower bound on the produced heat will be equal to $Q_{\text{consumed}} + Q_{\text{loss}}$.

$Q_{\text{start}} < \Omega$ i.e. more energy should be stored in the network in the end than what was stored at the start. We get: $\gamma = \Omega - Q_{\text{start}} > 0$, which means that in addition to what is consumed or lost, an extra γ amount of energy needs to be stored in the network at the end of the optimisation horizon, which is exactly what the lower bound describes.

As shown, in all possible cases of Ω, the lower bound on the amount of heat produced holds.

3.7.2. Consumed Heat

The total amount of heat that is consumed is simply the sum of the total consumer demand, since all demand is fulfilled by a valid solution.

$$Q_{\text{consumed}} = \sum_{c \in C, t \in T} Q_t^c \tag{3.21}$$

3.7.3. Minimum Losses

In Section 3.7.1 it is mentioned that Q_{loss} is not known before hand, but that $Q_{\text{loss}} \geq 0$ always holds. To improve the overall lower bound, this lower bound on the losses can be improved. This lower bound represents the heat losses that are unavoidable despite any choices made in operation of the DHS.

As all demand must be fulfilled, producers must produce at least as much heat as is demanded. However, due to the loss of heat during transport, just producing the total demand will not be enough to fulfil demand. Assuming there is η percent heat loss, producers need to produce an additional η percent of heat to fulfil demand.

The amount of heat in the network is defined by $Q = cmT$. As it is assumed there are no leaks in the network, heat loss must must be due to a decrease in temperature of the water in the network during transport. By determining a lower bound on the temperature losses in the network, a lower bound on the heat losses can be determined.

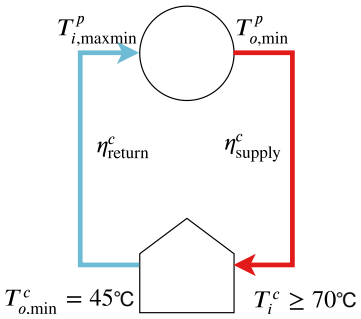


Figure 3.3: Minimum losses determination concept

Equation 3.22 can be used to determine the percentage temperature loss over a pipe path $\rightarrow$ in the network. The loss is expressed with respect to T_{env}, as no more heat can be lost when the water has reached this temperature.

$$\eta^{\rightarrow} = \frac{T_i^{\rightarrow} - T_o^{\rightarrow}}{T_o^{\rightarrow} - T_{\text{env}}} \tag{3.22}$$

Using this equation and constraints on the temperatures in the DHS, methods can be devised to find a lower bound on the heat losses in the supply and the return sides of the network.

The following equations will be used to determine the minimum temperature losses in the network:

$$\Delta T(t) = \Delta T_0 \exp(-t/\tau) \tag{2.5}$$

$$T_o = T_{\text{env}} + (T_i - T_{\text{env}}) \exp(-t/\tau) \tag{3.23}$$

$$\Rightarrow T_i = \frac{T_o - T_{\text{env}}(1 - \exp(-t/\tau))}{\exp(-t/\tau)} \tag{3.24}$$

$$\tau = A\rho cR \tag{3.25}$$

$$t = \frac{L\rho A}{\dot{m}} \tag{3.26}$$

To determine the minimum losses in the network, the supply side and the return side are considered separately. The losses in the supply side are also taken into account, as heat lost here will have to be refilled by the producers to get the water back to the right temperature.

Supply Side

As explained in Section 2.2.2, following Newton's cooling law (Equation 2.5), higher producer supply temperatures lead to higher heat losses in the network. Thus, by finding a lower bound on these supply temperatures, a lower bound on the heat losses in the supply side of the network can be found.

To find a lower bound on these supply temperatures, we can make use of the minimum consumer supply temperature constraints. By working backwards through the network, the minimum producer supply temperature for every consumer can be found. The difference between this supply temperature and the consumer's minimum supply temperature is the minimum amount of temperature loss in the network for this consumer.

Finding the minimum supply temperature of a producer for a consumer is done by finding the path through the supply side of the network with the smallest temperature losses. These losses can be determined using Equation 3.24 and the minimum consumer supply temperature constraint. Finding the path with the smallest temperature loss is done using a shortest path algorithm, where the path length is defined by the temperature losses along that path.

To determine the "length" of a supply side pipe, assume the outflow temperature is known and that the mass flow through the pipe is maximal. Then, determine the inflow temperature using Equation 3.24. The source of the path finding algorithm is the consumer, where the outflow temperature of the consumer's supply pipe is $T^c_{i,\text{min}}$.

The minimum temperature losses for a consumer can then be found by simply finding the path with the least losses over all producers.

The assumption of maximum mass flow is made to minimise the temperature loss in a pipe. Given that the outflow temperature of a pipe is fixed, the inflow temperature can be minimised by minimising the amount of time the water spends in the pipe. Following Equation 3.26, the amount of time water spends in the pipe is minimised by maximising the mass flow through the pipe. As temperature loss is simply the difference between the input and output temperatures, minimising the inflow temperature also minimises the temperature loss.

In short, finding the minimum temperature loss for a producer with respect to one consumer is done by determining a lower bound on the producer supply temperature. This lower bound is determined by assuming mass flow through the network is maximal and the consumer will receive water which exactly satisfies the minimum supply temperature constraint. As there can be multiple producers in the network, this shortest path finding should be done for all producers for the consumer.

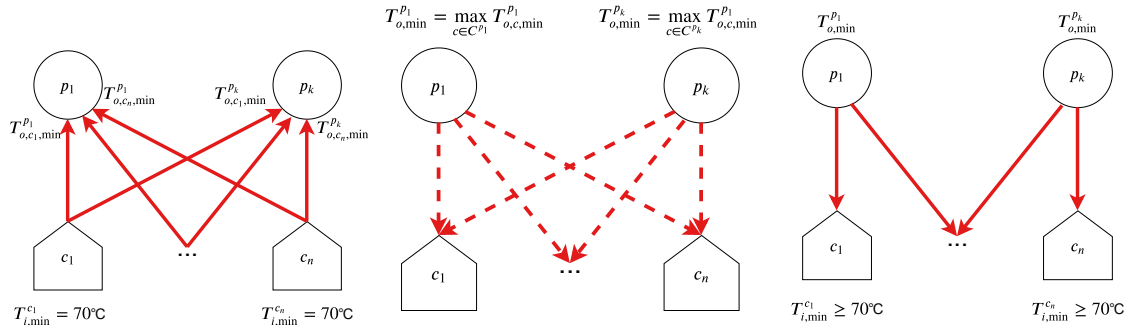
The lower bound on the percentage temperature loss for consumer c is determined by Equation 3.27 where $T^{p\to c}_{o,\text{min}}$ is the minimum supply temperature for producer p given that consumer c receives water of at least $T^c_{i,\text{min}}$. Using this, the lower bound on the supply side heat losses can be determined by Equation 3.28 where Q^c is the demand of consumer c.

$$\eta^c_{\text{supply}} = \min_{p\in P} \frac{T^{p\to c}_{o,\text{min}} - T^c_{i,\text{min}}}{T^c_{i,\text{min}} - T_{\text{env}}} \tag{3.27}$$

$$Q_{\text{min supply loss}} = \sum_{c\in C} \eta^c_{\text{supply}} Q^c \tag{3.28}$$

Shared Minimum Producer Supply Temperatures

By realising that producers provide heat to multiple consumers, another opportunity for improving the lower bound on the supply side losses becomes apparent. As producers provide heat to multiple consumers and all consumers have a minimum producer supply temperature, a producer must provide water at at least the maximum minimum producer supply temperature over its consumers, i.e. $T^p_o =$



(a) Phase 1: Backpropagation of minimum supply temperatures

(b) Phase 2: Generation of all possible temperature settings

(c) Phase 3: Determination of temperature settings with minimum losses

Figure 3.4: Minimum supply side heat loss algorithm phases visualisation

$\max_{c \in C^p} T^{p \to c}_{o,\min}$ where C^p is the set of consumers that receive their heat from producer p. As this temperature is higher, more losses will occur in the supply side of the network, leading to a better lower bound on the minimum supply side losses.

In Figure 3.4 three phases are shown of an algorithm to determine the shared minimum producer supply temperatures. A quick overview of the algorithm is given in Algorithm 3.1. The algorithm will be described in more detail next.

Phase 1 In phase 1, all consumers back propagate their minimum supply temperature through the supply side of the network to all producers like described before. Essentially, this phase is the entire previous section.

Phase 2 Once these temperatures have been collected at the producers, phase 2 begins. Producers' supply side temperatures are determined by choosing the maximum of its consumers' minimum producer supply temperatures. This ensures that the consumer that loses the most along the path through the network will still receive water of at least $T^c_{i,\min}$. Other consumers assigned to this producer will receive water with a temperature of more than $T^c_{i,\min}$.

If there is only one producer present in the network, this algorithm is quite simple: just take the maximum minimum producer supply temperature. However, when there are more producers present in the network, a decision has to be made on what producer a consumer should be assigned to. This is quite difficult as moving a consumer from one producer to another can have several effects: the original producer's minimum temperature drops or stays the same and the new producer's minimum temperature increases or stays the same.

Simply trying every assignment to find the best one is not feasible, as the number of assignments grows exponentially: $O(k^n)$. For example, when there are $n = 30$ consumers and $k = 2$ producers, this means there are already more than a billion possible assignments. By flipping the problem around, a more feasible algorithm is possible.

The idea behind this is: as every producer receives the minimum required producer supply temperature from all consumers, every producer has a limited set of $O(n)$ supply temperatures it can use. Using these lists as a basis to find the minimum assignment, only $O(n^k)$ possible settings have to be checked. With an instance of 2 producers and 30 consumers, this is less than a thousand possible temperature setting combinations.

To generate the possible temperature settings, the crossproduct of all producer minimum supply temperatures is used, e.g. the crossproduct of [75 °C, 79 °C] and [81 °C, 98 °C] is [[75 °C, 81 °C], [75 °C, 98 °C], [79 °C, 81 °C], [79 °C, 98 °C]].

Phase 3 In the third and final phase, all possible settings have been generated and the setting with the minimal minimal losses can now be found. In Algorithm 3.2 this process is described in more detail. By

Algorithm 3.1 Algorithm to find the minimum supply side losses

```
1: function MinimumSupplySideLosses
2:     Back propagate minimum consumer supply temperatures                ⇨ Phase 1
3:     Generate all O(n^k) possible temperature settings                  ⇨ Phase 2
4:     Q_supply loss ← ∞
5:     for every setting T_o in the possible settings do                  ⇨ Phase 3
6:         Q^{T_o}_supply loss ← SettingsMinLosses(T_o)
7:         Q_supply loss ← min{Q_supply loss, Q^{T_o}_supply loss}
8:     return Q_supply loss
```

Algorithm 3.2 Algorithm to find the minimum losses given fixed producer supply temperatures

```
1: function SettingsMinLosses(T_o)
2:     Q_loss ← 0
3:     for every consumer c do
4:         Q^c_loss, min ← ∞
5:         for every producer p where T^p_o ≥ T^p_{o,c,min} do
6:             Q^p_{c,loss} ← least heat loss path from p to c with T^p_o, assuming maximum mass flows
7:             Q^c_loss, min ← min{Q^c_loss, min, Q^p_{c,loss}}
8:         if Q^c_loss, min = ∞ then
9:             return ∞          ⇨ Infeasible as no producer has a high enough supply temperature
10:        else
11:            Q_loss ← Q_loss + Q^c_loss, min
12:    return Q_loss
```

iterating over all settings and determining the losses that would result from these settings, the minimal losses can be found. If it becomes apparent that a setting is not feasible due to no producer having a high enough supply temperature for a consumer, this setting is skipped.

The supply side temperature percentage heat loss is determined the same as before, except that the consumer supply temperature is likely higher than $T^c_{i,\text{min}}$ and the producer supply temperature higher than the consumer's minimum producer supply temperature.

Runtime Complexity The algorithm to find this lower bound is polynomial in the number of producers and as the number of producers is relatively small, running this algorithm is feasible.

This algorithm is polynomial in the number of producers because phase 1 runs in $O(knN \log N)$: the shortest path algorithm runs in $O(N \log N)$, where N is the network size, using Dijkstra's algorithm with a priority queue. This has to be repeated for every $O(kn)$ consumer/producer pair. The second phase then runs in $O(n^k)$ to generate all possible settings. The third phase runs in $O(n^k nk)$, as all $O(n^k)$ possible solutions need to be checked for every $O(kn)$ consumer/producer pairs. The shortest paths from phase 1 can be reused here, which means the shortest path algorithm does not need to be run again.

Thus, the overall complexity of the algorithm is: $O(knN \log N + n^k + kn^{k+1})$, which is polynomial in k, the number of producers.

Some additional possible improvements for the runtime are possible.

Because the percentage minimum temperature loss will not change when the demand in the network changes, it is possible to simply store the minimal percentage temperature loss for a network. When a new horizon is optimised, the stored percentage temperature loss can simply be multiplied with the demand, leading to the minimum heat loss for the given horizon.

To further decrease the runtime, the DHN can be preprocessed. When producers are placed right after each other, the producer that is closest to the consumers (in terms of losses) will always be the producer

consumers are assigned to. Doing otherwise would lead to more heat loss, which is the opposite of what Algorithm 3.1 does. Thus, producers that are behind the first producer can be removed, reducing the number of possible settings dramatically.

Finally, it is possible to combine phase 2 and 3 by immediately determining the losses of a solution when it is generated, however this was not done here to make the algorithm easier to understand.

Return Side

To further improve the lower bound on the overall heat losses, the minimum losses in the return side of the network can also be determined. A method similar to the method used for the supply side can be used again.

Here, instead of working backwards from minimum consumer supply temperature, we work forwards using the secondary return side set point temperature of the consumer's heat exchanger, e.g. 45 °C. This is the minimum possible primary return temperature of the consumer, as it is physically impossible to transfer more heat.

It is again assumed that the mass flows through the network are maximal. Now instead of determining the inflow temperatures for the pipes, we determine the outflow temperatures using Equation 3.23. Again, the path to the producer with the minimum temperature loss is found.

The lower bound on the percentage temperature loss on the return side is determined by Equation 3.29. This can then be used to determine the lower bound on the return side heat losses, i.e. Equation 3.30.

$$\eta^c_{\text{return}} = \min_{p \in P} \frac{T^c_{o,\text{min}} - T^p_i}{T^c_{o,\text{min}} - T_{\text{env}}} \tag{3.29}$$

$$Q_{\text{min return loss}} = \sum_{c \in C} \eta^c_{\text{return}} Q^c \tag{3.30}$$

Here, the effects of the combined consumer return temperatures are not taken into consideration.

Mass Flows

In the previous sections it was assumed that mass flows in supply (return) side pipes are maximal, but it can be physically impossible for the mass flow in a pipe to be equal to the pipe's limits. When the combined downstream (upstream) mass flows of a pipe are lower than the pipe's limits, the actual maximum mass flow is simply the sum of its downstream (upstream) mass flows.

By using the more realistic upper bound on the mass flows, a better lower bound on the temperature losses can be determined.

Of course, the maximum mass flows in a pipe should not be exceeded, so if the combined downstream (upstream) mass flows exceed the pipe's limits we simply use the pipe's limit as the mass flow used to calculate the lower bound temperature loss. This assumption breaks the conservation of mass flow invariant, but this is not a problem for calculations on the lower bound. In essence, the mass flows in the pipes along the lower bound temperature loss path are considered separately.

$d(e)$ is the set of immediately downstream pipes of pipe e and $u(e)$ the set of immediately upstream pipes of pipe e. If the node connecting e with its neighbours is a consumer or transfer, $d(e)$ and $u(e)$ are empty. The mass flows can then be determined using the following (recursive) equations:

$$\dot{m}_{\text{supply}}(e) = \begin{cases} \min\left(\dot{m}^e_{\text{max}}, \sum_{e' \in d(e)} \dot{m}_{\text{supply}}(e')\right) & \text{if } d(e) \neq \emptyset \\ \dot{m}^e_{\text{max}} & \text{otherwise} \end{cases} \tag{3.31}$$

$$\dot{m}_{\text{return}}(e) = \begin{cases} \min\left(\dot{m}^e_{\text{max}}, \sum_{e' \in u(e)} \dot{m}_{\text{return}}(e')\right) & \text{if } u(e) \neq \emptyset \\ \dot{m}^e_{\text{max}} & \text{otherwise} \end{cases} \tag{3.32}$$

Calculations

To determine the lower bound, it is assumed that the network is in a steady state. This means that if a temperature or mass flow measurement is taken at time t, a new measurement at the same point in the network at time t' will result in exactly the measurement values.

The overall algorithm to determine this lower bound on the heat losses is shown in Algorithm 3.3.

If a HTS lies along the minimum loss path, the temperatures on the secondary side are simply passed to the primary side of network. Since a heat exchanger primary supply temperature should always be at least as high as its secondary supply temperature, simply passing along the minimum found temperature so far is possible. This same argument holds for the return temperatures.

Algorithm 3.3 Algorithm to determine minimum heat losses in the network

1: Determine maximum mass flows in DHN using Equations 3.31 and 3.32
2: Determine minimum supply side heat losses using Algorithm 3.1
3: Determine minimum return side heat losses using Equation 3.30
4: Return the sum of the supply and return side losses

3.7.4. Maximum Produced Heat

In addition to lower bounds on the amount of heat that needs to be produced and the losses in the network, an upper bound on how much heat a producer can produce in one unit of time also exists.

This upper bound is defined by $Q^p_{\text{max}} = ({T_o}^p_{\text{max}} - T_{i_{\text{min}}}) \cdot \dot{m}^p_{\text{max}} \cdot c$, where ${T_o}^p_{\text{max}}$ is the maximum output temperature of producer p, $T_{i_{\text{min}}}$ the minimum return temperature, and $\dot{m}^p_{\text{max}}$ the maximum mass flow through producer p. The minimum return temperature is chosen such that it never occurs in practice, e.g. 30 °C.

As the maximum output temperatures differ per producer and the production costs differ per producer and over time, every producer has a maximum amount of costs it can make per block of time.

By dividing the total amount of heat that needs to be produced over the cheapest maximum cost time blocks, a lower bound on the total heating cost can be determined.

$\xi^p_t = c^p_t \cdot Q^p_{\text{max}}$ is the maximum cost at time t by producer p. ${}^{0:i}\xi^p_t$ is the ith cheapest producer time block to produce the maximum amount of heat.

This means that if Q amount of heat has to be produced, the total costs can then be determined using Algorithm 3.4, in which the heat to be produced is greedily divided over the cheapest producer time blocks.

Algorithm 3.4 Greedy algorithm to divide the heat to be produced over the cheapest producers time blocks and determine the lower bound on the costs

```
1:  function GreedyHeatProductionDivision(Q)
2:      c ← 0
3:      for i in 1, ..., |P| · |T| do
4:          p and t are equal to p and t in ^{0:i}ξ^p_t
5:          if Q^p_max > Q then
6:              c ← c + c^p_t · Q
7:              Q ← 0
8:              break
9:          else
10:             c ← c + ^{0:i}ξ^p_t
11:             Q ← Q − Q^p_max
12:     return c
```

3.7.5. Combining the Bounds

Finally, the overall lower bound can be determined.

The lower bound on the heat production is determined by taking into account how much heat is consumed, Q_{consumed}, how much loss there is in the network, $Q_{\text{loss}}^{\geq 0}$ and how much additional heat should be produced to have the enough heat stored in the network at the end of the horizon, γ.

By using the upper bound on heat production per producer, the lower bound becomes:

$$c_{\text{op}} \geq \text{GreedyHeatProductionDivision}\left(Q_{\text{produced}}\right) \tag{3.33}$$

$$\geq \text{GreedyHeatProductionDivision}\left(\max\left(0, Q_{\text{consumed}} + Q_{\text{loss}}^{\geq 0} + \gamma\right)\right) \tag{3.34}$$

This section has answered **RQ 2.1**. It is possible to determine a lower bound on the DHS problem, by making certain unrealistic assumptions that do not detract from the essence of the problem. This lower bound is determined by creating an additional constraint to ensure enough heat is stored at the end of the horizon. This constraint is not unrealistic to add, as the future beyond the horizon needs to be kept in mind while optimising for the horizon. In addition to this, a useful lower bound on the losses in the network has been determined as well. The precision of the lower bound will be further discussed in Section 7.7.

4

Related Work

Optimisation of district heating operations is not a new topic and much research has been done on it already. To give an overview of what work has been done already and which areas are still relatively unexplored, this chapter gives an overview of existing work and related problems. The existing works have different approaches to the problem, which are discussed in Section 4.1

This research focusses on optimisation of operations, but there are other problems that are related to this, like demand forecasting and the design of DHSs. These topics are discussed in more detail in Section 4.2.

Finally, Section 4.3 draws conclusions from the existing work and work on related problems.

4.1. Existing Work

Most of the existing work is focussed on mathematical optimisation techniques, like non-linear optimisation or linear optimisation by linearising certain non-linear components of the optimisation problem. These works are discussed in more detail in Section 4.1.1.

In more recent years, more research is being done on artificial intelligence based optimisation techniques, although this work mostly focusses on DHS design rather than operations. Some work exists where metaheuristics are used to optimise operations, but this work is quite limited. This work and other artificial intelligence based techniques are discussed in more detail in Section 4.1.2.

4.1.1. Mathematical Optimisation

Most of the existing work on the optimisation of district heating operations consists of mathematical optimisation, which is not unexpected, as the optimisation of this problem has already been under consideration since 1969 when Sazanov and Mil'man [30] researched how to optimally control the gas turbines of a DHS gas turbine plant using "a digital computer program […] using the gradient method to find the optimum". As climate change has climbed higher on the public agenda in the last decades, the amount of research done on DHSs has only grown.

Iterative Optimisation

One of the foundational papers on this subject from Benonysson et al. [4] uses a model of a DHS as part of a non-linear optimisation where the mass flows, and thus the transport delays, in the network are assumed to be constant. As transport delays are not constant in reality, this assumption leads to a difference in the actual costs when the DHS is simulated with the optimised settings, compared to the costs that the optimisation model determined. To solve this problem, the optimisation is run iteratively updating the mass flows in every iteration until the optimisation converges. The authors found that this algorithm is not always able to converge without tuning specific parameters for the DHS under optimisation. This algorithm did find solutions that reduce production peaks by utilising the heat storage capacity of the network more than a constant supply temperature setting would.

Many of the mathematical optimisation approaches researched after this paper were based on [4]. One of these is Giraud et al. [10]. Where several endogenous variables are again assumed to be constant during the optimisation and are updated using a simulator. This research follows a model predictive control (MPC) approach where the optimisation considers a horizon, but only the first interval of the solution is used. MPC is further discussed in the next section. When the first interval of the optimisation horizon is about to end, the optimisation is run again with new measurements and improved demand predictions which is supposed to lead to better results.
The decision variables are the supply temperatures and differential pressures at producers. In addition to this, unit commitment costs are also taken into account, which means the decision of when to turn on peak boilers is also under the optimisation's control. Peak boilers are smaller producers that can also be present in the secondary network to ensure the ability to fulfil demand even when there is a very high demand peak. These production sites are often very expensive to run relative to regular production sites. Compared to "expert law" operations, this approach resulted in 8% cost savings, and reduced the use of peak boilers significantly by utilising the network storage effect more effectively.

This work is used as one of the building blocks for Bavière and Vallée [2], which describes an MPC system that is used to advice plant DHS operators in real time. As the DHS in this work is quite large, the model of the network is changed by using a reduced set of consumers, called representative consumers. This work also explicitly analyses and uses the effect of network storage on heat load.

Model Predictive Control

Model predictive control (MPC) is a control process in which systems are controlled in real time using a model of the system. The predictive model makes predictions based on the current state of the system. Based on these predictions, the control system attempts to make the most optimal control decisions. Often, the predictive model is updated online to improve the model by incorporating new measurements from the system. These kinds of systems are often built in software, but some specialised hardware also exists, e.g. for use in electronic power conversion circuits.

In [12] by Grosswindhager et al. MPC is used to control the supply temperatures of a DHS. The MPC technique used is dynamic matrix control, which is one of the earliest MPC techniques. The model estimates supply temperatures at different nodes in the network by fitting the model coefficients to historic data. Given the most recent measurement data, a heuristic way of operating the network, mass flows predictions and the previous settings suggested by the controller, the controller predicts the most optimal supply temperature settings for the next time period. The method is called fuzzy because of the predictive component for the mass flows in the controller. This predictive component makes allows the controller to more precisely estimate the network state, which allows the controller to make more optimal choices.

4.1.2. Artificial Intelligence Based Optimisation

In addition to the mathematical optimisation approaches discussed in the previous section, some attempts have been made to optimise the operations using artificial intelligence based approaches.

Markov Decision Processes

With Markov decision processes (MDP) the system under optimisation is modelled as a set of possible states the system can be in and the possibility of moving from one state to another when a certain action is performed. [3] A policy is a mapping from states to actions, i.e. when the system is in a certain state, this action should be taken. Given the transition probabilities are not 1, the system stochastically ends in a new state. To optimise a policy for a system a technique called value iteration can be used. This technique uses a dynamic programming approach to determine the expected reward of performing an action for every state. Using these values it becomes possible to always perform the optimal action in any state.

In [17, 18], the energy/mass displacement in a hydraulic network is optimised by finding an optimal operation policy π^*. Although the work does not explicitly model the problem as an MDP, the terminology and concepts used are very reminiscent of MDP. The optimal policy is found through a method similar to value iteration on MDPs. Although the original paper [31] in which the authors describe the optimisation method was not accessible at the time of writing this research, a description is also given in [32]. This paper describes "a general framework for the optimal control of non-linear hydrodynamic systems under

uncertainties". The examples given use the mass flow through distribution stations as control variables, but a method is described to use "dummy control variables" on which the mass flow control depends. As explained in Chapter 2 the mass flow is dependent on demand and the supply temperatures, thus making it possible to use this approach. The research performed in [17] was a preliminary study to show the feasibility of this method on the DHS problem. In [18] it was reported that the approach is feasible and promising after applying it to a DHS in Kemi, Finland. Further research would focus on building a more detailed model of this DHS.

Metaheuristics

The use of metaheuristics for optimising DHS operations is surprisingly limited. Little work has been found and the work that has been done is quite limited in scope.

One of the most common metaheuristics is the genetic algorithm. Genetic algorithms encode a solution as a string of variables. These variables can represent a single bit, or something much more complex like the membership of a certain set. The algorithm maintains a population of candidate solutions, the members of which are mutated and bred, through something called crossover, to create a new offspring population. From this offspring population, only the fittest individuals are selected which causes unfit individuals to be removed from the population. The algorithm repeats this process until it reaches a certain termination criterion. The idea behind this is that as the algorithm iterates over the populations only the fittest solutions will survive selection. Through this selection, the overall fitness of the population will increase until the algorithm reaches an optimum solution.

One of the earliest works from Hori et al. [16] makes use of a genetic algorithm to optimise operational planning to minimise costs and dynamic load variation.

Later, Sakawa et al. [29] used a genetic algorithm to find solutions for the binary decision variables in an integer linear programming (ILP) formulation of district heating/cooling operations. The binary variables represent the decision of whether to use a producer or not as starting up and shutting down a producer has cost associated with it, the unit commitment cost. The network storage effect or heat losses were not taken into account, which enables the linearity of this problem. As computation power has dramatically increased and much progress has been made in the last decades, the integer linear programming formulation is likely solvable using conventional methods by now.

More recently, Fang and Lahdelma [7] use a genetic algorithm to determine the optimal heating schedule for an entire day. The decision variables are the producer supply temperatures and the producer mass flows. It is assumed that fuel, pumping costs and consumer return temperatures are static. The propagation delays and network storage effect are not taken into account and no mention is made of a time scale. This leads to the conclusion that the solution found by the optimisation, will be used over the entire day. The decision variable domain is discretised in intervals of 4% of the input range, e.g. with an input range of 67 °C to 115 °C, the possible settings are 67, 68.9, 70.8, …113.1, 115 °C. With the sample network used in the work, savings ranged from 0.6 % to 10.6 %.

4.2. Related Problems

The problem of optimising district heating operations has some properties in common with other problems, or assumptions are made that can be improved with other techniques.

The first of these is demand forecasting, which consists of predicting the demand of consumers in the network over a future horizon, which is discussed in Section 4.2.1.

What design decisions are made for a DHS influences how it is operated. Several works related to this are discussed in Section 4.2.2.

As described in Chapter 3, the goal in this research is to minimise operating costs. However, it is also possible to optimise for multiple goals at the same time and then making the trade-off on which solution is best. Work related to this is discussed in Section 4.2.3.

4.2.1. Demand Forecasting

Most optimisation assume that the demand profile is given, thus the stochasticity of the demand is not taken into account. As the demand is used as the basis on which the optimisation builds, bad demand

predictions will lead to suboptimal decisions or even decisions that might make it impossible to fulfil demand. For this reason, work has been done on forecasting the demand profiles of consumers.

One of the earlier papers to consider this problem is from Nielsen and Madsen [25]. In this work, a model is updated online to do an hour-by-hour prediction of the heat load in DHS. The model consists of several factors like wind speed, nominal solar radiation, air temperature, home insulation, time of day, day of the week, etc. This model has many coefficients which are fitted to measurement data of the past time frame using adaptive recursive least squares estimates. This work is reminiscent of manually crafting an artificial neural network (ANN) to predict the heat demand, which allows some domain knowledge to be added to the model.

Dotzauer [6] does something similar, but less complex. The only factors that are taken into account are the outdoor temperature, historical demand data and time of day/day of the week. Even this relatively simple model with little domain knowledge is able to predict demand fairly accurately. This makes it apparent that social factors and outdoor temperature are the most important factors in demand prediction.

In Laakkonen [21] the heat load and customer return temperatures are estimated using an actual ANN which uses historical demand and (forecast) weather data as input. The predicted demand is then fed into a "brute-force" optimiser which simply searches for a solution by checking every possible valid solution. The solution space is limited to 7 possible actions ($\Delta T_s = \pm 5\,°C,\ \pm 2\,°C,\ \pm 1\,°C,\ 0\,°C$) for every time step and the optimisation is required to end at the same end temperature. This means only $2 \cdot 7^{(T/2)}$ solutions exist, where T is the planning horizon. The ANNs models built to predict the demand and consumer return temperatures were quite accurate and improve runtime performance, as simulation is easier with the predicted return temperatures. The optimisation in this research resulted in 1.2% - 1.7% savings, which is lower than the authors expected. This is explained to be due to the fact that the case plant was already operated quite efficiently. The use of neural networks did allow the optimisation to run quite fast and was shown to be a useful tool.

4.2.2. System Design

Before it makes sense to optimise the operations of a DHS, the DHS first has to be designed. Designing a DHS consists of determining where pipes should be laid and how large they should be.

In [5] the urban area where the DHS will be built is modelled as a graph where edges are streets and nodes are intersections, dead ends, or heat sources. The problem is modelled as a selection problem and encoded with mixed integer linear programming (MILP). For every street it is decided whether a pipe should be built there. Consumers are grouped to the nearest street, their demand added to the street peak and total annual demand. The pipe diameters are determined by the peak demand and the profitability of laying a pipe in a street is determined by the total annual demand of a street. Interestingly, the optimisation goal is not to find the cheapest topology for a network where all consumers are connected, but to find regions where it would be profitable to connect consumers. This means it is possible not all consumers are connected to the eventual DHS.

Another facet of systems design is the selection of components like heat exchangers, pumps or pipes.

In [23] by Li et al. and [34] by Zeng et al. genetic algorithms are used to select which types of pipes should be used in a district heating/cooling network. The possible values for every gene is an integer indicating which type of pipe should be used at the location that is encoded by the gene.

4.2.3. Multi-objective Optimisation

In this work, the operation cost are the only objective to minimise. However, in other works multiple (contradicting) goals can be under optimisation.

In [8] by Fazlollahi and Maréchal the design and operation phases of an energy system are optimised simultaneously while keeping two goals in mind: minimising costs and minimising CO_2 emissions. Unfortunately, these goals are often contradictory. To solve this problem the optimisation model decision variables are split into two parts: the master set and slave sets. The optimisation first optimises the master set using an evolutionary algorithm. This master set consists of system design variables, e.g. pipe diameters or heat exchanger sizes. The slave sets are then optimised using a MILP solver to find

the optimal operation procedures. This is repeated over several iterations until the maximum number of iterations has been reached. The solutions found over all iterations are presented as the Pareto frontier, from which the system designers can make a choice. This technique can be used, for example, for choosing what kind of heat producer will be used in a DHS.

4.3. Conclusions

The field of DHS operations optimisation and related problems is extensive and much work has already been done, but there are still areas that are not yet explored. The optimisation of DHS operations so far has mostly been done using mathematical optimisation, like iterated linear programming (LP) or MPC.

Although metaheuristics are already used in the optimisation of DHS design, the use of them in optimisation of operations is still quite unexplored. Some research has already been done on this, but this is quite limited as many assumptions are made that remove most of the problem's complexity. Removing factors like the network storage effect, the non-linearity of heat exchangers and the non-convexity created by different temperature settings over time make the problem much easier.

Problems related to the optimisation of operations explored quite extensively. Work on demand forecasting is already quite advanced and can be done relatively cheaply. With the increase in smart metering, demand forecasting will become even more precise.

Finally, the goal of **RQ 2.1** was to find a method to determine a lower bound on the operating costs for the DHS problem. However, such a lower bound has not been found in literature.

5 Methods

Multiple different approaches to optimise the district heating system (DHS) operations could be used. This chapter describes several of these approaches and a method to avoid the need to solve the expensive heat exchanger equations.

All metaheuristic approaches need a starting point for their search. In Section 5.1 several methods for creating an initial solution for the DHS problem are described.

Section 5.2 describes what termination criterium is used in the algorithms.

Next, Section 5.3 describes a naive genetic algorithm which can make use of domain knowledge, like the maximum temperature gradient in the network.

As the concepts behind the genetic algorithm are more suitable for discrete combinatorial optimisation problems instead of real-valued optimisation problems, Sections 5.4 and 5.5 describe methods that are specifically designed to optimise real-valued problems.

In Section 5.4 a description is given of the self-adaptive differential evolution (SaDE) algorithm. This algorithm is designed to work with real-valued non-convex optimisation problems by maintaining a population of target vectors which are mutated and combined into new target vectors.

Section 5.5 gives a description of the covariance matrix adaptation evolutionary strategies (CMA-ES) algorithm, which maintains a covariant normal distribution that is iteratively updated to move toward the optimal solution.

Finally, in Section 5.6, a method is discussed to approximate the heat exchanger functions described in Section 2.2 as solving these functions iteratively is quite slow.

5.1. Initial Solution

Most metaheuristic optimisation techniques require a starting point for their search toward the optimal solution. For the DHS problem several possibilities for generating an initial solution are reasonable.

5.1.1. Random

The first and simplest method of generating an initial solution is to initialise all variables randomly. For temperature variables, this means choosing a value from a uniform distribution between the upper and lower temperature boundaries. The valve setting variables, if needed, are chosen uniformly in $[0, 1]$.

As no constraints are taken into account with this method, the initial solution will most likely violate all constraints. However, as the variables are initialised randomly, the chance of biasing an algorithm toward a local optimum is small as well.

5.1.2. Heating Curve

As described in Chapter 1, a DHS can be controlled using a heating curve which, given the outdoor temperature, indicates at what temperature the producer should output all day. The pumps in the network pick up the changes in demand by varying the mass flows in the network. The heating curve produces feasible solutions, thus it might be a good starting point for optimisations.

To find a heating curve for a network, it is assumed that the DHS problem is convex when time is removed as a variable factor. This means only temperature is still a relevant variable.

A simple binary search algorithm can now be used to find the minimum required temperature to satisfy all demand without breaking constraints. Algorithm 5.1 gives short a description of this algorithm.

Algorithm 5.1 Heating curve search algorithm where f is the fitness function indicating constraint violations and operating costs

```
1:  function FindHeatingCurve(T_min, T_max, ε)
2:      T_l, T_r ← min, max
3:      T_m ← (T_r − T_l)/2
4:      while |T_l − T_r| > ε do
5:          T_m ← (T_r − T_l)/2
6:          if f(T_l) < f(T_m) then
7:              T_r ← T_m
8:          else if f(T_r) < f(T_m) then
9:              T_l ← T_m
10:         else
11:             T_lm ← (T_l + T_m)/2
12:             T_mr ← (T_r + T_m)/2
13:             if f(T_lm) < f(T_m) then
14:                 T_r ← T_m
15:             else
16:                 T_l ← T_lm
17:             if f(T_mr) < f(T_m) then
18:                 T_l ← T_m
19:             else
20:                 T_r ← T_mr
21:     return T_m
```

5.1.3. Existing Solution

The final method to specify the initial solution is to simply use a previously found solution and shifting it up to the start of the horizon to optimise.

This solution is already optimised, however the information used during the optimisation might be outdated. This situation is not uncommon in MPC, where measurements are used to decide on a policy for the near future. Given the new information, the optimisation will adapt the existing solution to fix suboptimal behaviour or constraint violations.

Of course, when the solution is shifted, no values are available for the end of the horizon. This is solved by repeating the last settings until the end of the new optimisation horizon. Should this result in constraint violations or suboptimal control, the optimisation will fix this.

5.2. Termination

To decide whether an algorithm should terminate different decisions can be made.

One of the most conventional methods to decide this is to terminate the algorithm after a certain amount of iterations without improvement or when a maximum number of iterations is reached.

The maximum number of iterations without improvement ensures that an algorithm is terminated when

it has converged. Running the algorithm for a longer amount of time is useless as it is unlikely the algorithm will improve any further.

The maximum number of iterations is to ensure that the algorithm actually converges. Should this limit be reached, other measures should be taken as the algorithm is converging too slowly.

5.3. Genetic Algorithm

As a naive implementation, a simple genetic algorithm has been created. This algorithm does not explicitly make use of the fact that the DHS problem is real-valued in its methods, but it does find real-valued solutions.

A genetic algorithm maintains a population of solutions and mutates and combines individual solutions in the population to find new and better solutions. A genetic algorithm consists of 5 main elements. The initial solution, mutation, combination, selection and termination. How these components work together is described in Algorithm 5.2.

The previous section describes methods to create initial solutions and the termination criterion. This section describes the other three components and the solution encoding used in this algorithm (Section 5.3.1). The mutation method (Section 5.3.2) used in this algorithm. In Section 5.3.3, multiple crossover operations are described. Then, Section 5.3.4 describes how this algorithm selects the solutions for the next generation. Finally, Section 5.3.5 describes how the algorithm is initialised when an existing initial solution is used.

5.3.1. Solution Encoding

For the genetic algorithm (GA) solutions are encoded by creating two matrices of variables. In both matrices columns represent time. For the first matrix, rows represent the temperature settings for one heat source. In the second matrix, rows represent the settings for one valve.

5.3.2. Mutation

To explore the search space, the GA uses mutation. Individuals in the population are changed (mutated) by changing variable values.

Whether a solution is mutated or not is decided randomly. This can be influenced through an input of the algorithm, the mutation probability. In every iteration of the algorithm a random value is drawn from a uniform distribution in $[0, 1]$ for every individual in the population. If this number is lower than the mutation probability, the individual is mutated.

The mutation in this algorithm randomly replaces settings with random values. These values are between the upper and lower temperature limits for the temperature variables and in $[0, 1]$ for valves. The probability of a variable value being randomly replaced in a solution is 0.05, but at least one variable is changed to ensure the solution differs from the original solution. This algorithm is described in Algorithm 5.3.

5.3.3. Crossover

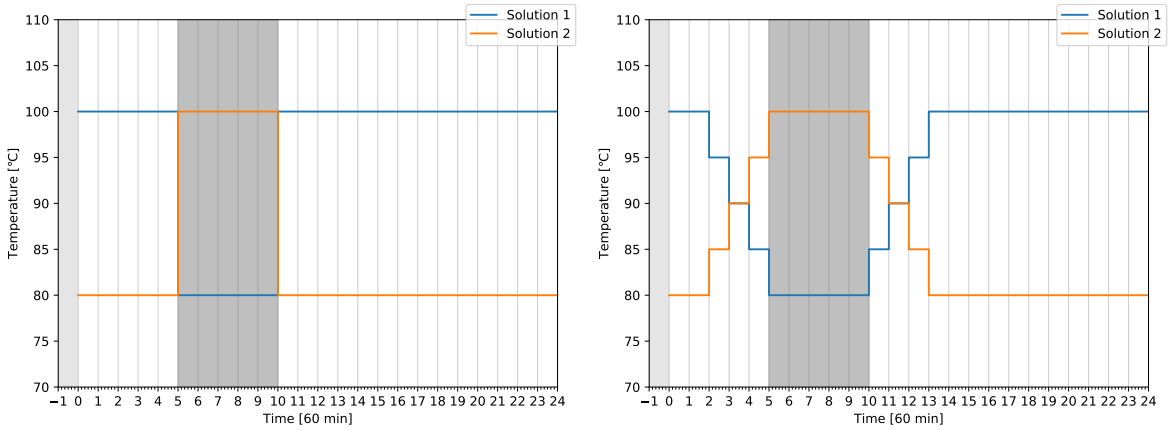
With crossover, two individuals are picked from the population and combined into two new offspring solutions. For the DHS problem two crossover operators are quite natural, which will be explained

Algorithm 5.2 Genetic algorithm overview

1: $g \leftarrow 0$
2: $\mathbf{X}^{(0)} \leftarrow$ initial solution
3: **while not** should terminate **do**
4: $\quad \mathbf{M}^{(g)} \leftarrow$ Mutate($\mathbf{X}^{(g)}$, mutation probability)
5: $\quad \mathbf{O}^{(g)} \leftarrow$ Crossover($\mathbf{X}^{(g)}$)
6: $\quad \mathbf{X}^{(g+1)} \leftarrow$ Selection($\mathbf{X}^{(g)}$, $\mathbf{X}^{(g)}$, $\mathbf{X}^{(g)}$)
7: $\quad g \leftarrow g + 1$
8: **return** $\mathbf{X}^{(g)}$

Algorithm 5.3 Genetic algorithm mutation operation

```
1:  function Mutate(X, p)
2:      M ← ∅
3:      for X_i in X do
4:          if rand[0, 1] ≤ p then
5:              j_r ← rand[0, |X_i|]
6:              for X_ij in X_i do
7:                  if rand[0, 1] ≤ 0.05 or j = j_r then
8:                      M_ij ← rand[j_min, j_max]
9:                  else
10:                     M_ij ← X_ij
11:     return M
```



(a) Time

(b) Fade, with a maximum temperature gradient of 5 °C/h

Figure 5.1: Crossover operations

here. The crossover operation concepts are also shown in Figure 5.1

Time

The first crossover is over time. The two solutions are each split at two points over time, i.e. the temperature and valve settings for one time block are not separated, but the settings for time block b and $b+1$ might be separated. This is basic 2-point crossover. The centre parts of the solutions are switched, which creates two new solutions.

With this crossover operation, if one solution is good for time $t : t + \Delta$ and the other for time $t' : t' + \Delta'$, they can be combined into one better solution. This crossover is described in Algorithm 5.4 and illustrated in Figure 5.1a.

Algorithm 5.4 Genetic algorithm time crossover

```
1:  function Crossover(X)
2:      O ← ∅
3:      for i = 0, 2, ..., N do
4:          t_1, t_2 ← rand[1, |X_i|], rand[1, |X_i| - 1]
5:          t_1, t_2 ← min(t_1, t_2), max(t_1, t_2)
6:          O_{i,t_1:t_2}, O_{i+1,t_1:t_2} ← X_{i+1,t_1:t_2}, X_{i,t_1:t_2}
7:      return O
```

Fade

The second crossover is similar to the time crossover, but instead of just inserting the values from the other parent, the values are inserted and the existing surrounding temperature values are smoothed

out so inserting the new values does not create temperature gradient constraint violations. Figure 5.1b illustrates this crossover operation.

The crossover operation smooths out the temperature settings until the first two consecutive settings do not violate any constraints. For example, if the temperature at time $t = 13\,\text{h}$ would be 70 °C for solution 2 before the crossover, there would be a constraint violation between $t = 12\,\text{h}$ and $t = 13\,\text{h}$ of 5 °C. During crossover, the fade over operation would pull that temperature up to 75 °C so no gradient violation would exist any more. However, should such a violation exist at time $t = 18\,\text{h}$, the fade over operation would leave that untouched, as the fade over would not have touched that time.

5.3.4. Selection

To decide whether a new solution should be placed in the new population, the fitness of the individual is used. The algorithm uses k-tournament selection.

This procedure picks k solutions from the current population, the mutated individuals and the crossover offspring. From these three solutions, the best solution is chosen to be entered into the new populated. k-tournament selection allows the average fitness of the population to improve while still maintaining diversity.

In this algorithm $k = 3$ is chosen.

5.3.5. Existing Initial Solution

To use an existing solution as a starting point for the optimisation, a random population is generated and the existing solution is added to this population. This solution will be the best in the population, and might be improved through mutation and crossover.

5.4. Self-Adaptive Differential Evolution

The self-adaptive differential evolution (SaDE) algorithm is designed to work with continuous values. It is based on differential evolution (DE) [26] where certain meta-parameters have to be specified manually by the user. In self-adaptive differential evolution (SaDE), these parameters are automatically adapted during the optimisation procedure. This allows the algorithm to adapt itself based on how the optimisation has progressed so far and removes the complexity of manually tuning the meta-parameters.

Just like the genetic algorithm, SaDE maintains a population of solutions which are mutated and combined to create new, and hopefully better, solutions. In SaDE the population consists of a matrix where rows are solutions and columns the parameters of these solutions.

How solutions are encoded is described in Section 5.4.1. Then, Section 5.4.2 discusses how mutation is done in this algorithm. Then, in Section 5.4.3, the crossover procedure is discussed. In the genetic algorithm, crossover and mutation were independent. In differential evolution (DE) the mutants created by the mutation procedure are used as one of the parents in the crossover operation. Then, in Section 5.4.4, the selection procedure is discussed. Finally, Section 5.4.6 describes how this algorithm is initialised when an existing initial solution is used.

Many different versions of SaDE exist. The one described here is by Qin and Suganthan [28]. Algorithm 5.5 describes the algorithm as described in [28].

5.4.1. Solution Encoding

For this algorithm, solutions are encoded as a single vector of values. The temperature settings for heat sources are concatenated. The valve settings are then concatenated as well. Thus, the solution is represented by an $(n+v)\cdot T$-dimensional vector where n is the number of heat sources, v the number of valves and T the number of time blocks in the horizon. The temperature settings for producer $0 \leq p < n$ can be found in elements $p \cdot T$ until $(p+1)\cdot T - 1$ and the valve settings for valve $0 \leq j < v$ in elements $n \cdot T + j \cdot v$ until $n \cdot T + (j+1) \cdot v - 1$.

5.4.2. Mutation

The set S is the set of all mutation strategies. Five strategies are described in the original, but only 2 are used in the SaDE algorithm by Qin and Suganthan. These two strategies are:

- DE/rand/1: $\mathbf{V}_i^{(g)} = \mathbf{X}_{r_1}^{(g-1)} + \mathbf{F}_i^{(g)} \left(\mathbf{X}_{r_2}^{(g-1)} - \mathbf{X}_{r_3}^{(g-1)} \right)$
- DE/best/2: $\mathbf{V}_i^{(g)} = \mathbf{X}_i^{(g-1)} + \mathbf{F}_i^{(g)} \left(\mathbf{X}_{\text{best}}^{(g-1)} - \mathbf{X}_i^{(g-1)} \right) + \mathbf{F}_i^{(g)} \left(\mathbf{X}_{r_1}^{(g-1)} - \mathbf{X}_{r_2}^{(g-1)} \right)$

where $\mathbf{X}_{\text{best}}^{(g)}$ is the best individual in the population of generation g and $r_1 \neq r_2 \neq r_3 \neq i$ are three random indices and the current index. DE/rand/1 explores the search space by combining random solutions, whereas DE/best/2 exploits progress made so far resulting in convergence toward the optimum.

In DE F is a user specified constant, whereas in SaDE, $\mathbf{F}^{(g)}$ is drawn from a normal distribution $\mathcal{N}(0.5, 0.3)$ and clamped to $(0, 2]$ for every individual and generation.

To choose between which mutation strategy is used, a distribution is created. Every strategy $z \in S$ has an associated probability of being chosen, $\mathbf{p}_z$. This probability is updated after a certain number of iterations, the learning period, by recording how many of the mutants created by the strategy are used in selected new solutions. The number of accepted solutions is recorded in $\mathbf{s}_z$ and the number of rejected solutions in $\mathbf{f}_z$. How the new distribution is created is described in Lines 17 and 18 in Algorithm 5.5. After the new distribution has been created, the recorded numbers are reset.

5.4.3. Crossover

Just like the crossover operation in standard DE, the mutant vector $\mathbf{V}_i^{(g)}$ and the target vector $\mathbf{X}_i^{(g-1)}$ are combined into a trial vector $\mathbf{U}_i^{(g)} = \left(\mathbf{U}_{i1}^{(g)}, \ldots, \mathbf{U}_{ij}^{(g)}, \ldots \mathbf{U}_{in}^{(g)} \right)$ following:

$$\mathbf{U}_{ij}^{(g)} = \begin{cases} \mathbf{V}_{ij}^{(g)} & \text{if rand}[0,1] \leq CR \text{ or } j = j_{\text{rand},i} \\ \mathbf{X}_{ij}^{(g-1)} & \text{otherwise} \end{cases} \tag{5.1}$$

where $j_{\text{rand},i}$ is a random index in $[1, n]$ to ensure the trial vector differs from the target vector.

In the standard DE algorithm the crossover probability *CR* is a user defined constant. In the SaDE algorithm this value is drawn from a normal distribution $\mathcal{N}(CRm, 0.1)$ and clamped to $(0, 1]$. *CRm* is updated every so often depending on how well previously sampled values for *CR* worked.

5.4.4. Selection

The SaDE algorithm uses a simple hill climbing acceptance strategy where a trial vector is only accepted if it improves on its target vector.

$$\mathbf{X}_i^{(g)} = \begin{cases} \mathbf{U}_i^{(g)} & \text{if } f\left(\mathbf{U}_i^{(g)}\right) < f\left(\mathbf{X}_i^{(g-1)}\right) \\ \mathbf{X}_i^{(g-1)} & \text{otherwise} \end{cases} \tag{5.2}$$

In Algorithm 5.5 the selection procedure returns the new population, the sum of the *CR* values of trials that were accepted $\dot{CR}_+^{(g)}$, the number of accepted trial vectors $n_+^{(g)}$ and the number of accepted, $\mathbf{s}^{(g)}$, and rejected, $\mathbf{f}^{(g)}$, trials for every mutation strategy.

5.4.5. Parameter Adaptation

The adaptive part of the SaDE algorithm consists of adapting the mutation probability and the strategy selection distribution.

The mutation probability is drawn from $\mathcal{N}(CRm, 0.1)$ for every target vector once every 5 generations. In each generation, the cumulative moving average of *CR* values that lead to accepted trial vectors is updated. Every 5 iterations new values for CR are drawn from the distribution with *CRm* as the mean.

Every 25 iterations *CRm* is updated to the cumulative moving average which is then reset. The initial value for *CRm* is 0.5. The original paper stores a collection of all *CR* values, but this is not necessary as only the mean of these values is needed.

The strategy selection distribution is used to select which strategy to use for mutation. In each generation, it is recorded how often the use of a strategy led to a selected solution. This is then used to update the strategy distribution every 50 generations. The updated distribution is created by dividing the success ratio for every strategy by the sum of all success ratios. The selection probability for every strategy $\boldsymbol{z}$ is calculated according to formula 5.3.

$$\mathbf{p}_z = \frac{\frac{\mathbf{s}_z}{\mathbf{s}_z + \mathbf{f}_z}}{\sum_{z' \in S} \frac{\mathbf{s}_{z'}}{\mathbf{s}_{z'} + \mathbf{f}_{z'}}} \tag{5.3}$$

Algorithm 5.5 Self-Adaptive Differential Evolution algorithm as described in [28]

$CRm \leftarrow 0.5$
$\mathbf{s} \leftarrow \mathbf{0}, \mathbf{f} \leftarrow \mathbf{0}, \mathbf{p} \leftarrow \frac{\mathbf{1}}{|S|}$
$\overline{CR_+} \leftarrow 0, n^+ \leftarrow 0$ ⇨ Cumulative moving average of accepted CR

$\mathbf{X}^{(0)} \leftarrow$ initial solutions
$l \leftarrow \left|\mathbf{X}_0^{(0)}\right|$
$\mathbf{CR} \leftarrow \mathcal{N}(CRm, 0.1) \in (0, 1]^l$
while not should terminate **do**
    $\mathbf{F}^{(g)} \leftarrow \mathcal{N}(0.5, 0.3) \in (0, 2]^l$ ⇨ Draw *F* values
    $\mathbf{V}^{(g)} \leftarrow$ GenerateMutants($\mathbf{X}^{(g-1)}$, $\mathbf{F}^{(g)}$, S, $\mathbf{p}$)
    $\mathbf{U}^{(g)} \leftarrow$ GenerateTrials($\mathbf{X}^{(g-1)}$, $\mathbf{V}^{(g)}$, $\mathbf{CR}$)
    $\mathbf{X}^{(g)}, \dot{CR}_+^{(g)}, n_+^{(g)}, \mathbf{s}^{(g)}, \mathbf{f}^{(g)} \leftarrow$ Select($\mathbf{X}^{(g-1)}$, $\mathbf{U}^{(g)}$)
    ⇨ Update learning parameters
    $\overline{CR_+} \leftarrow \frac{\overline{CR_+} \cdot n_+^{(g)} + \dot{CR}_+^{(g)}}{\max(1, n_+^{(g)} + n_+^{(g)})}$ ⇨ Update cumulative moving average of accepted *CR*
    $\mathbf{s} \leftarrow \mathbf{s} + \mathbf{s}^{(g)}, \mathbf{f} \leftarrow \mathbf{f} + \mathbf{f}^{(g)}$
    if $g \mod 50 = 0$ **then** ⇨ Update strategy probabilities
        $\mathbf{r} = \frac{\mathbf{s}}{\mathbf{s}+\mathbf{f}}$ ⇨ Element wise division
        $\mathbf{p} = \frac{\mathbf{r}}{\sum_{z \in S} \mathbf{r}_z}$
        $\mathbf{s} \leftarrow \mathbf{0}, \mathbf{f} \leftarrow \mathbf{0}$

    if $g \mod 25 = 0$ **then** ⇨ Move *CR* mean to accepted average
        $CRm \leftarrow \overline{CR_+}$
        $\overline{CR_+} \leftarrow 0, n_+ \leftarrow 0$

    if $g \mod 5 = 0$ **then** ⇨ Redraw *CR* values
        $\mathbf{CR} \leftarrow \mathcal{N}(CRm, 0.1) \in (0, 1]^l$

    $g \leftarrow g + 1$

return $\mathbf{X}^{(g-1)}$

5.4.6. Existing Initial Solution

To use an existing initial solution, the existing solution is taken and the population is randomly initialised based on this solution. This is done by taking the initial solution and varying the variable values by 10% of the variable bounds window size.

For example: we have an initial solution of 95 °C. The minimum and maximum supply temperatures are 70 °C and 110 °C. This means that the randomised initial variable value can fall between 93 °C and 97 °C.

5.5. Covariance Matrix Adaptation Evolutionary Strategy

Another method to optimise real-valued problems is covariance matrix adaptation evolutionary strategies (CMA-ES) [14]. This method maintains a normal covariant distribution for the decision variables.

By iteratively sampling from this distribution, the fitness landscape is explored. Every iteration the distribution is updated with the some of the sampled solution. If certain samples are in a better part of the solution space, they are more likely to be used to update the distribution, moving the distribution toward better areas of the solution space.

This concept is shown in Figure 5.2 [33]. The dots are the samples taken during the current generation, the orange dotted ellipse is the step size, its centre the distribution mean and the covariance is represented by the rotation of the ellipse with respect to the axes.

5.5.1. Solution Encoding

Just like for SaDE, solutions are encoded as a single vector of values. The temperature settings for heat sources are concatenated. The valve settings are then concatenated as well. Thus, the solution is represented by an $(n+v)\cdot T$-dimensional vector where n is the number of heat sources, v the number of valves and T the number of time blocks in the horizon. The temperature settings for producer $0 \leq p < n$ can be found in elements $p \cdot T$ until $(p+1)\cdot T - 1$ and the valve settings for valve $0 \leq j < v$ in elements $n \cdot T + j \cdot v$ until $n \cdot T + (j+1) \cdot v - 1$.

For CMA-ES an additional step has to be taken, as the variable scales have to be about the same. However, the valve settings and the temperature settings are about one to two orders of magnitude apart from each other. For this reason, the variables are rescaled between 0 – 10. This means that when a producer has a maximum production temperature of 110 °C and a minimum production temperature of 70 °C, that a CMA-ES variable with value 5.5 represents a temperature setting of $\frac{110\,°C - 70\,°C}{10-0} \cdot 5.5 + 70\,°C = 86.5\,°C$.

For valves, the settings are also scaled between 0 – 10, which means a CMA-ES variable of 5.5 represents a valve setting of $\frac{1-0}{10-0} \cdot 5.5 = 0.55$.

5.5.2. Meta-Parameters and Variables

The CMA-ES algorithm has several meta-parameters:

- λ – Population size
- μ – The number of parents used for recombination, value used in this research: $\mu = \left\lfloor \frac{\lambda}{2} \right\rfloor$
- c_m – Learning rate for the mean, value used in this research $c_m = 1$
- c_μ – Learning rate for covariance rank-μ update of the covariance matrix, usually $c_\mu = 1$
- c_c – Learning rate for cumulation of the rank-one update of the covariance matrix, usually $c_c = 1$
- d_σ – Damping parameter for step-size update, usually $d_\sigma = 1$
- w_i – Weighting factor for sample $\mathbf{x}_{i:\lambda}^{(g)}$ for all $1 \leq i \leq \mu$
- $\mu_{\text{eff}} = \left(\sum_{i=1}^{\mu} w_i^2\right)^{-1}$
- $c_\sigma = \frac{\mu_{\text{eff}}+2}{n+\mu_{\text{eff}}+5}$ – Learning rate for the step size,

The following variables are used in the algorithm:

- $\mathbf{x}_k^{(g)}$ – Solution $1 \leq k \leq \lambda$ of generation g
- $\sigma^{(g)}$ – Step size at generation g

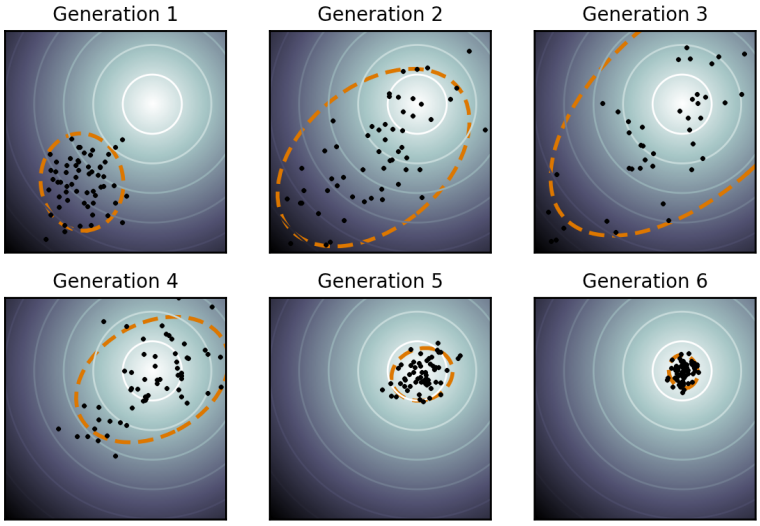


Figure 5.2: CMA-ES directional search concept for a 2D problem. The fitness landscape is represented by the background. White indicates a higher fitness than blue. The black dots are the samples taken from the current distribution. The orange dotted ellipse indicates the variance for the distributions. Rotation with respect to the vertical and horizontal axis represents the covariance. [33]

- $\mathbf{m}^{(g)}$ – Mean in generation g
- $\mathbf{C}^{(g)}$ – Covariance matrix at generation g
- ${\mathbf{C}^{(g)}}^{-1/2}$ – A transformation matrix to equalise the length of the axes
- $\mathbf{x}_{i:\lambda}^{(g)}$ – The ith best solution in generation g when the solutions are ordered in non-decreasing order of fitness, i.e. $f\left(\mathbf{x}_1^{(g)}\right) \leq \ldots \leq f\left(\mathbf{x}_\lambda^{(g)}\right)$
- $\mathbf{p}_\sigma^{(g+1)}$ – Evolution path, which is the path of the mean has taken over multiple generations. This is used to update the step size.

5.5.3. Algorithm Description

In Algorithm 5.6 a very high overview of the CMA-ES algorithm is given. For a more in depth description, the reader is referred to [14].

On line 7 λ samples are taken from the current distribution. Then on line 8, the new weighted mean is computed using the learning factor c_m.

Next, the covariance matrix is updated. On line 10, the covariance for the current samples is computed. This is then used in 11 to compute the covariance matrix for the next generation.

Finally, the step size is updated on lines 13 and 14. This is done by computing the conjugate path $p_\sigma^{(g+1)}$ using the path learning rate c_σ, a normalisation constant $\sqrt{c_\sigma(2-c_\sigma)\mu_{\text{eff}}}$ and a transformation matrix ${\mathbf{C}^{(g)}}^{-1/2}$ that rescales all axes to be equally sized. This conjugate path is used to compute the new step size. By dividing the length of the conjugate path by the expected length of a $\mathcal{N}(\mathbf{0},\mathbf{I})$ distributed vector and some scaling, the new step size is computed.

The idea behind this is: if the conjugate path length is larger than expected, many small steps in one direction are taken over the generations. The same can be achieved by taking one big step, thus the step size should be larger. When the conjugate path length is smaller than expected, the optimisation is taking steps back and forth over the optimum, thus the step size should be decreased.

To use this algorithm, the `pycma` library, version 2.7.0, from [15] is used.

Algorithm 5.6 Covariance Matrix Adaptation Evolutionary Strategy algorithm

$g \leftarrow 0$
$\mathbf{m}^{(0)} \leftarrow$ initial solution
$\mathbf{C}^{(0)} \leftarrow \mathbf{I}$
$\mathbf{p}_\sigma^{(0)} \leftarrow \mathbf{0}$
$\sigma^{(0)} \leftarrow$ initial step length
while not should terminate **do**
    $\mathbf{x}_k^{(g+1)} \sim \sigma^{(g)} \mathcal{N}(\mathbf{m}^{(g)}, \mathbf{C}^{(g)})$ for $k = 1, \ldots, \lambda$
    $\mathbf{m}^{(g+1)} \leftarrow \mathbf{m}^{(g)} + c_m \sum_{i=1}^{\mu} w_i \left(\mathbf{x}_{i:\lambda}^{(g+1)} - \mathbf{m}^{(g)}\right)$
    ⇨ Update the covariance
    $\mathbf{C}_\mu^{(g+1)} \leftarrow \sum_{i=1}^{\mu} w_i \left(\mathbf{x}_{i:\lambda}^{(g+1)} - \mathbf{m}^{(g)}\right)\left(\mathbf{x}_{i:\lambda}^{(g+1)} - \mathbf{m}^{(g)}\right)^\top$
    $\mathbf{C}^{(g+1)} \leftarrow (1 - c_\mu)\mathbf{C}^{(g)} + c_\mu \frac{1}{\sigma^{(i)^2}} \mathbf{C}_\mu^{(i+1)}$
    ⇨ Update step size
    $\mathbf{p}_\sigma^{(g+1)} \leftarrow (1 - c_\sigma)\mathbf{p}_\sigma^{(g)} + \sqrt{c_\sigma(2 - c_\sigma)\mu_{\text{eff}}}\mathbf{C}^{(g)^{-1/2}} \frac{\mathbf{m}^{(g+1)} - \mathbf{m}^{(g)}}{\sigma^{(g)}}$
    $\sigma^{(g+1)} \leftarrow \sigma^{(g)} \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\left\|\mathbf{p}_\sigma^{(g+1)}\right\|}{E\left\|\mathcal{N}(\mathbf{0}, \mathbf{I})\right\|} - 1\right)\right)$

    $g \leftarrow g + 1$
return $\mathbf{m}^{(g)}$

5.5.4. Existing Initial Solution

To use an existing initial solution, the initial mean of the distribution is set to be equal to the initial solution. The initial step size and covariance is not changed.

5.6. Heat Exchanger Approximation

One of the most expensive parts of the DHS simulation is solving the heat exchanger equations. More than half of the simulation time is spent solving heat exchanger equations, which are usually reasonably smooth functions with the exception of certain boundaries. These functions are described in detail in Section 2.2.2.

The boundaries where the functions are no longer smooth, is exactly where the optimisation works toward. By lowering the primary supply temperature, the primary mass flow will increase. Because pumping costs are lower overall than heating, more savings can be made by lowering the supply temperatures than by lowering the pumping power. This means that an approximation of the heat exchanger function needs to be reasonably precise at this boundary.

Given the primary supply temperature, secondary return and target supply temperatures and secondary mass flow, the heat exchanger returns the primary return temperature, primary mass flow, actual secondary supply temperature and the delivered heat.

There are multiple approaches to approximating the heat exchanger: fitting a function to every returned value, using machine learning, or interpolating the values. Fitting a function is not easy as the heat exchanger functions are not smooth over their entire input domain. Using machine learning is an option, but it is more complex than needed, because interpolation can solve the problem more easily.

5.6.1. Interpolation

Interpolating a function consists of using a limited amount of known function evaluations to estimate unknown points. Thus, by taking samples of the real heat exchanger functions, values that are not yet known can be interpolated.

This is done by sampling over a grid of input values that are predefined. The ranges of these variables should be chosen such that all realistic values are within it, e.g. a primary supply temperature of 40 °C would not be realistic for the type of DHSs in this research.

Three different interpolation approaches are explored. The first method uses a single interpolator for the entire domain. The second uses a non-regular grid interpolator with subspaces that have a higher sample density if the interpolation error is too large in that subspace. The third is a combination of the previous two but uses an explicit tree structure of regular grid interpolators.

To determine whether an interpolator has a high enough precision, regular samples are taken over the entire domain and the root mean squared error (RMSE) is determined for every output value.

Regular Grid Interpolator

The regular grid interpolation uses a regular grid of measurement points and bilinear interpolation to find values for points that have not been solved for. Regular grid bilinear interpolation can be done in $\mathcal{O}(1)$ time which makes it much faster than the iterative process needed to solve the heat exchanger functions.

The algorithm for $\mathcal{O}(1)$ bilinear interpolation can be seen in Algorithm 5.7 and is based on the unit square method. X and Y are strictly ascending ordered vectors which elements are evenly spaced. `data` is a $(|X|, |Y|)$ matrix of data points and `point` is the point for which the interpolated value has to be determined.

Algorithm 5.7 $\mathcal{O}(1)$ bi-linear interpolation based on the unit square method

```
1: function interpolate(X, Y, data, point)
2:     i ← |X|· (point.x - X.first) / (X.last - X.first)
3:     j ← |Y|· (point.y - Y.first) / (Y.last - Y.first)
4:     i⊥, i⊤ ← ⌊i⌋, ⌈i⌉
5:     j⊥, j⊤ ← ⌊j⌋, ⌈j⌉
6:     return data[i⊥, j⊥] ·(i⊤ − i) · (j⊤ − j)+
7:            data[i⊤, j⊥] ·(i − i⊥) · (j⊤ − j)+
8:            data[i⊥, j⊤] ·(i⊤ − i) · (j − j⊥)+
9:            data[i⊤, j⊤] ·(i − i⊥) · (j − j⊥)
```

Arbitrary Point Interpolation

Using arbitrary sample points means it is no longer easy to find out which points are closest to the to be interpolated point. To do the interpolation with arbitrary points the SciPy[1] library was used. This uses Qhull[2] to do Delaunay triangulation and then uses Barycentric interpolation on the found triangles to do the interpolation.

To create the samples for the interpolator, a regular grid is used. If the RMSE is too large, the grid is divided into subspaces and the procedure repeats until the precision is high enough or until the maximum sample density is reached. The entire procedure is described more detailed in Algorithm 5.8.

On line 3 `subspaces` is defined, which is a queue that contains all the subspaces that need more precision. Then, while there are still subspaces in the queue, samples are taken, regular interpolators built based on these samples and it is checked whether there is enough precision now. If not, the subspace is split into smaller subspaces again, which are added to the queue. If there is enough precision, the points and associated values are added to the data that is used for the overall interpolator.

Unfortunately, because of the irregular data, construction of the interpolator takes an unreasonable amount of time and is quite memory intensive. Because of this, no analysis has been performed of this interpolator, as the construction of even a single function interpolator never finished running.

Constructing this arbitrary point interpolator in reasonable time cannot be done, due to the high number of sample points required to reach the desired precision. For the typical heat exchanger described above, more than 30 000 points are required. Even though Delauney triangulation runs in polynomial time [1], the input size is too high to construct the triangulation within a reasonable amount of time. It might be possible to generate the sample points in a more efficient manner to reduce the amount of points needed, but this was not further explored.

[1] https://www.scipy.org
[2] https://www.qhull.org

Algorithm 5.8 Non-Regular Grid Interpolation

```
1:  function build_interpolated_htx(space, heat_exchanger, rmse_boundaries)
2:      interpolation_points, interpolation_values ← {}, {}
3:      subspaces ← {space}
4:      while subspaces is not empty do
5:          subspace ← pop(subspaces)
6:
7:          sample_points ← generate_regular_samples(subspace)
8:          sample_values ← heat_exchanger(*sample_points)
9:          regular_interpolators ← build_regular_interpolators(sample_points, sample_values)
10:
11:         verification_points ← generate_verification_points(subspace)
12:         verification_values ← heat_exchanger(*verification_points)
13:         actual_values ← regular_interpolators(*verification_points)
14:                                   ⇨ rmse is a 4-tuple, one element for every return value
15:         rmse ← sqrt((1/n) ∑(verification_values - actual_values)^2)
16:         if any rmse > rmse_boundaries and can_split(space) then
17:             subspaces ←_U split(subspace)
18:         else
19:             interpolation_points, interpolation_values ←_U sample_points, sample_values
20:     return build_interpolators(interpolation_points, interpolation_values)
```

Tree-Based Regular Grid

This approach is a combination of the previous two. It uses an explicit tree structure to store the data in regular grid sub-interpolators. The use of regular grid interpolators means building and using the interpolators is fast, while the tree structure allows the interpolators to use regular grids.

The algorithm to build this interpolated heat exchanger is described in Algorithm 5.9. This function calls itself recursively until every subspace has enough precision or until the subspace can no longer divided. Just like in the previous section, samples are taken of the space and regular grid interpolators are built. The RMSE of these interpolators is then checked. If the RMSE is too high, the space is split up into subspaces and new child nodes are constructed for every subspace. If the error is low enough, the regular grid interpolators that were constructed are stored in the node and the recursion terminates.

Experiments

To measure performance of the interpolators, samples are taken on a regular grid over the entire domain and the error and run times are measured for every sample. Of course, lower errors and run times are better.

Every experiment is repeated 10 times and the measurement results are averaged when necessary.

Precision To compare the precision of the different interpolators, the heat exchanger described earlier is used. 300 measurements are taken on every axis, e.g. as the primary supply temperature ranges from 70 °C to 110 °C there are 7.5 samples per unit. As there are two variable axis, 300^2 regular samples are taken over the entire input domain.

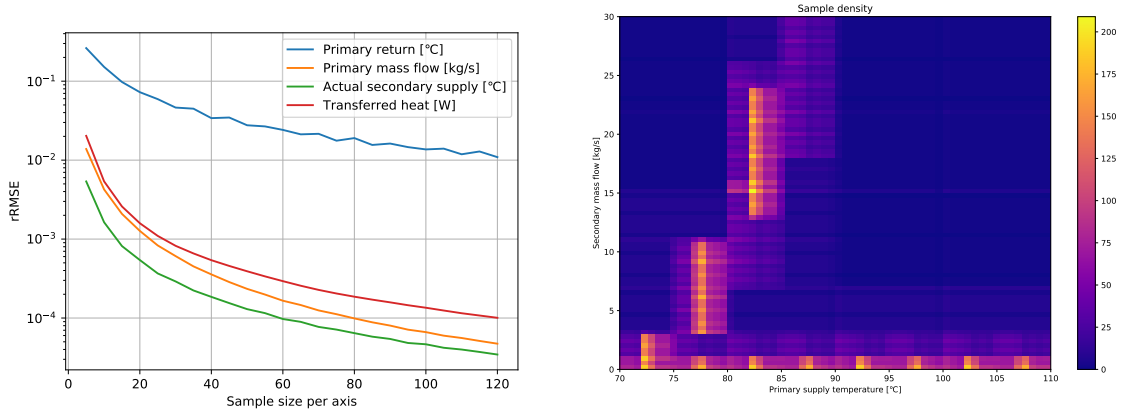
For the regular grid interpolator the sample size can be varied. To explore what sample size leads to an acceptable error, the resulting error can be measured. In this case the error is defined as the relative RMSE (rRMSE), which is the RMSE divided by the mean.

The adaptive interpolation precision can be controlled by setting the desired precision. The interpolator will have this precision, unless the region could no longer be split into subregions. This can be controlled by the `can_split` function. For this experiment, regions will no longer be split when the sample density will exceed 30 samples per unit.

In Figure 5.3a the overall rRMSE of the regular interpolator is shown relative to the sample size. As can be seen, the error goes down when the sample size increases, although the primary return temperature

Algorithm 5.9 Tree-Based Regular Grid Interpolation

```
1:  function build_interpolated_htx(space, heat_exchanger, rmse_boundaries)
2:      node ← {}
3:
4:      sample_points ← generate_regular_samples(space)
5:      sample_values ← heat_exchanger(*sample_points)
6:      regular_interpolators ← build_regular_interpolators(sample_points, sample_values)
7:
8:      verification_points ← generate_verification_points(space)
9:      verification_values ← heat_exchanger(*verification_points)
10:     actual_values ← regular_interpolators(*verification_points)
11:                                  ⇨ rmse is a 4-tuple, one element for every return value
12:     rmse ← sqrt((1/n) ∑(verification_values - actual_values)^2)
13:     if any rmse > rmse_boundaries and can_split(space) then
14:         subspaces ← split(space)
15:         node.children ← build_interpolated_htx(*subspaces, heat_exchanger, rmse_boundaries)
16:     else
17:         node.interpolators ← regular_interpolators
18:     return node
```



(a) rRMSE

(b) The adaptive interpolator sample density

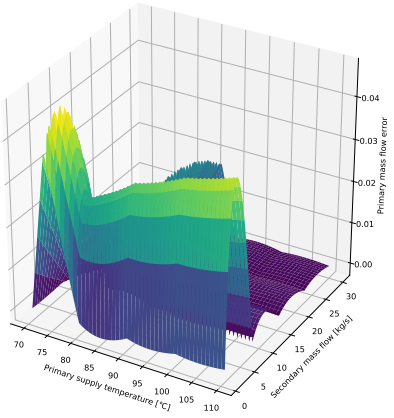
Figure 5.3: Interpolator samples

error remains significantly higher than the other variables. This is likely due to the fact that this function is much less smooth than the other functions. When a sample is taken right at the boundary, values used in the interpolation are much farther apart from each other than when this happens for the other heat exchanger functions.

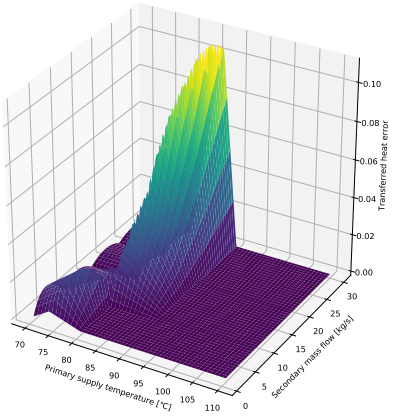
The areas where errors are high can be seen in Figure 5.4. As expected, the areas where the functions are not smooth have a high error, because values used for interpolation at the boundary are further apart. By increasing the sample size, the errors become orders of magnitude smaller. The adaptive interpolator has the smallest error, which is to be expected as it takes as many samples as it needs to get the error below a certain setting.

The adaptive interpolator takes more samples where the function is harder to interpolate to reduce the error in that area. In Figure 5.3b the sample density of the interpolator is shown.

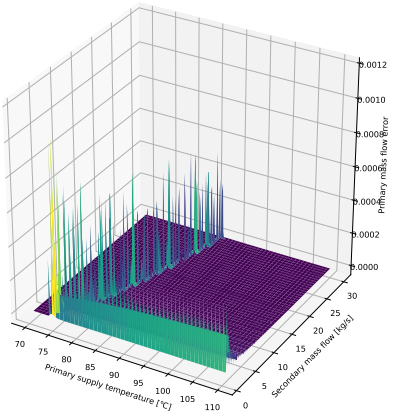
As can be seen, most samples are around the boundary and the region close to the secondary mass flow being 0 m/s. Although it is not shown in Figure 2.10, the primary return temperature is of course equal to the primary supply temperature if the secondary mass flow (demand) is 0, which means more samples are needed in that area to ensure the interpolation is precise enough.



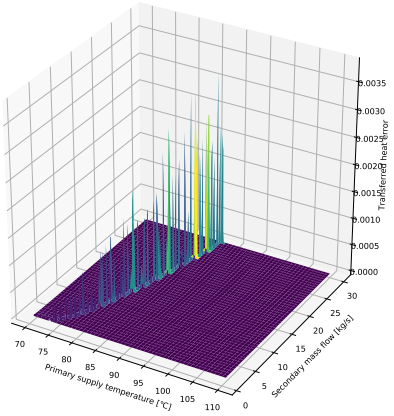
(a) Primary mass flow, sample size: 5



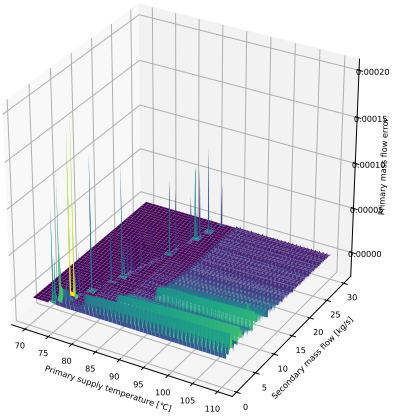
(b) Delivered heat, sample size: 5



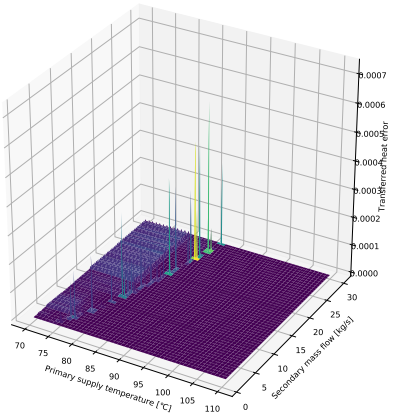
(c) Primary mass flow, sample size: 120



(d) Delivered heat, sample size: 120

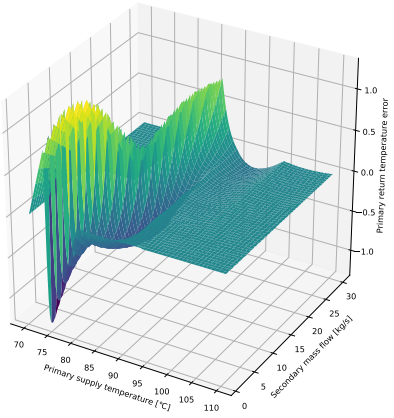


(e) Primary mass flow, adaptive

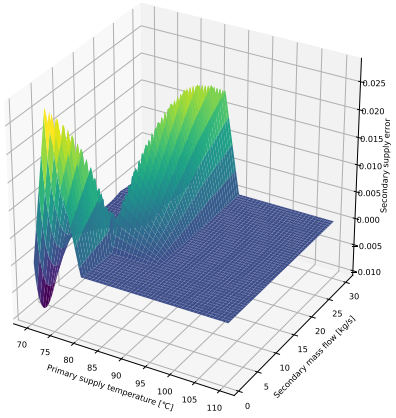


(f) Delivered heat, adaptive

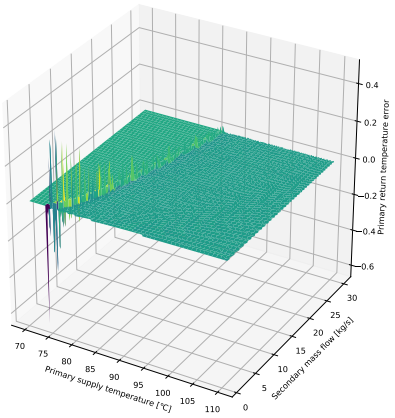
Figure 5.4: Regular interpolated heat exchanger error for sample size of 5 and 120 and the adaptive interpolator. Note the different y-axis scales.



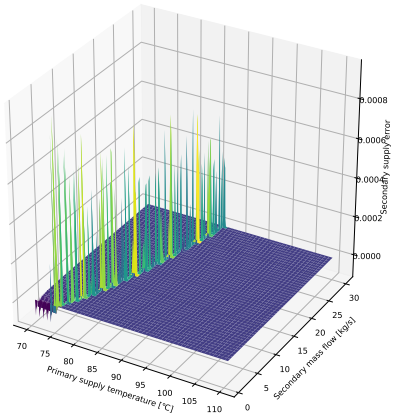
(g) Primary return temperature, sample size: 5



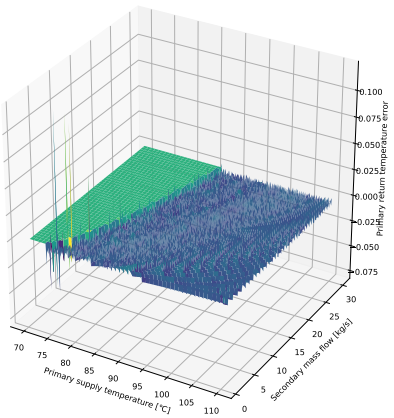
(h) Secondary supply temperature, sample size: 5



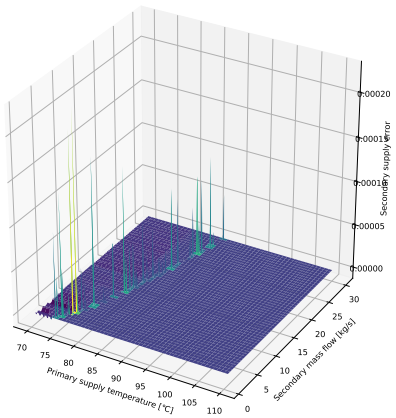
(i) Primary return temperature, sample size: 120



(j) Actual secondary supply temperature, sample size: 120



(k) Primary return temperature, adaptive



(l) Actual secondary supply temperature, adaptive

Figure 5.4: Regular interpolated heat exchanger error for sample size of 5 and 120 and the adaptive interpolator. Note the different y-axis scales. (Continued)

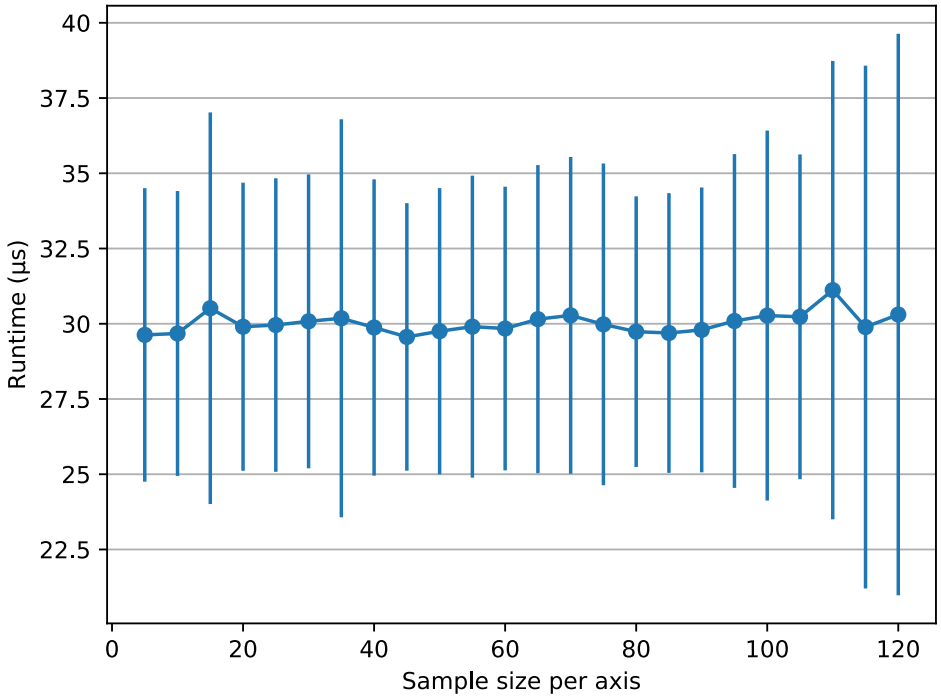


Figure 5.5: Runtime for the regular grid interpolator with respect to an increasing sample size

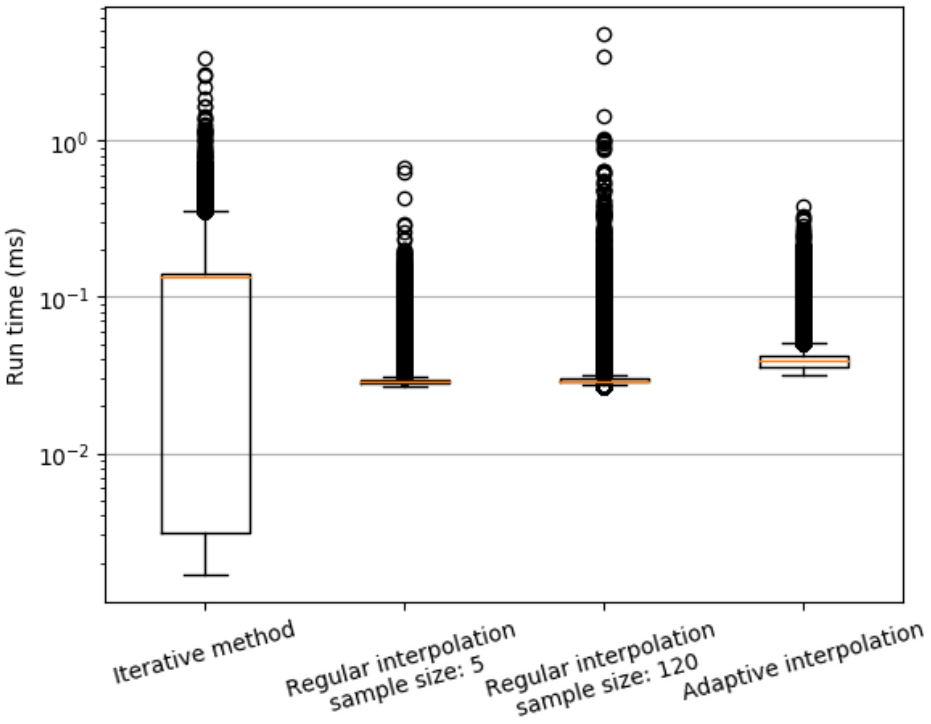


Figure 5.6: Interpolation run time for the different interpolation methods.

The samples around the minimum supply temperature boundary are of course needed to ensure the data around this boundary is correct. The adaptive interpolator, thus behaves as expected.

Runtime As can be seen in Figure 5.5 the runtime of sampling the heat exchanger interpolator is constant when the sample size increases, which is to be expected as the interpolation algorithm runs in $\mathcal{O}(1)$.

Something that stands out is that the standard deviation goes up when the sample size is larger than 100. This is likely due to the large number of interpolation points which might outgrow the CPU cache size, leading to more cache misses when looking up data for interpolation.

The most important reason to create an interpolator is to improve the simulator runtime performance. Figure 5.6 shows the run time for every interpolator type. As can be seen the performance increase gained by creating an interpolated heat exchanger is not as large as was expected. However, since the relative time spent solving the heat exchanger function in the simulation is a significant portion, the performance increase is still useful to have.

As can also be seen, the run time for interpolators is more stable than for the iterative method. This is due to the fact that the iterative method can finish early in some cases, which leads to a higher variance in run time. To further improve the performance of the interpolations, the early exit from the iterative method might be added to the interpolators.

The average run time for the adaptive interpolator is slightly higher than for the regular interpolator, which can be explained by the fact that the latter can immediately interpolate the value, whereas the other first has to traverse down the tree. The adaptive interpolator run time depends on the depth of the tree which can be controlled with the `can_split` method and the desired accuracy parameter.

Another interesting thing to note for the regular grid interpolators is that they were first implemented using the `Regular2DGridInterpolator` from the SciPy library. During the run time performance analysis it was found that the interpolators were up to three times slower than the normal heat exchanger. This was surprising at first, as it was expected that interpolation would be much faster than iteratively solving equations.

After inspecting the implementation of the `Regular2DGridInterpolator`, it was found that it was implemented using an $\mathcal{O}(n^2)$ algorithm to be able to interpolate multiple points at once. This is unnecessarily slow for our purposes, so the $\mathcal{O}(1)$ bilinear interpolation algorithm was implemented manually instead.

Conclusions From the results we can conclude that the interpolators offer speed up compared to the iterative approach, but this speed up was less significant than expected. To further extend the improved performance of the interpolation, the early exit that is possible for the iterative method can be added to the interpolated heat exchangers.

Choosing between the regular and the adaptive interpolator depends on what is preferred. The regular interpolator is faster than the adaptive interpolator, but the adaptive interpolator requires fewer samples to reach the same precision.

In the case of the simulator, the adaptive interpolator is preferred as a high sample size is needed to reach the desired precision. This leads to an unnecessarily high number of samples in ‘easy’ sections of the heat exchanger functions if the regular interpolator were used.

6 Experiments

This chapter describes experiments with the methods described in Chapter 5 and the expected results.

With these experiments, it can be quantified which algorithm performs best in which instance, which settings are best for the algorithms and how much savings can be made for different instances with respect to using heating curve based operations. What measurements are needed to get this information is described in further detail in the first section, 6.1.

Next, Section 6.2 gives a detailed description of the instances used in the experiments to obtain the information we want. In addition to this the meta parameter settings for the algorithms are also described.

Finally, Section 6.3 describes the results expected from the experiments.

6.1. Measurements

To answer research questions **RQ 1** and **RQ 2** we need to know what makes one algorithm better than the other and how optimal the solutions that are found are. In addition to this we also need to know how robust an algorithm is when different kind of DHSs are used and what algorithm settings are best given an instance. Finally, more information might be needed to explain behaviour observed during experiments.

To obtain all this information, the following metrics are measured during the experiments:

- Optimality of the solution found compared to a lower bound
- Number of iterations before convergence, i.e. how often is the simulation run before the algorithm has converged.
- Number of iterations until the first valid solution is found.
- Convergence consistency: Does the algorithm consistently converge to the same solution?
- The solution population over time

These different measurements are further discussed in the remainder of this section.

6.1.1. Optimality

To determine how optimal the solutions found are, the lower bound described in Section 3.7 is used. It is assumed that the amount of energy at the end of the optimisation horizon is at least as much as at the start of the optimisation horizon.

6.1.2. Convergence Speed

As described before, the algorithms are terminated when no improvements have been made for a number of iterations. In the experiments, this is done after 30 iterations without improvement. Algorithms

that converge faster than others might be preferred to the others. In addition to this, it is needed to investigate why some algorithms might converge very slowly and other faster.

6.1.3. First Generation With a Valid Solution

Algorithms that find a valid solution very quickly might get stuck in local optima or converge very slowly compared to algorithms that take longer to find the first valid solution, but find a much better solution.

6.1.4. Convergence Consistency

To measure whether the algorithm converges to the same solution during each run, a measure of similarity is defined. The similarity is defined as the relative Euclidean distance of the solution vectors (Equation 6.1) where element j of solution x_i represents a decision variable, like temperature or valve setting. l, j_{max} and j_{min} are the dimensionality and the upper and lower bounds for the variable values, respectively. The distance for every variable is divided by the difference, because variables can have different scales, e.g. temperature is 2 orders of magnitude larger than valve settings. When the value of the distance measure is low, the solutions are more similar.

To then known how consistent an algorithm is, for every run of an experiment, the best solution of the last generation is taken. This includes invalid solutions. Then, for every solution, the average distance to the other solutions is calculated.

$$d(x_0, x_1) = \sqrt{\frac{1}{l}\sum_{j=0}^{l-1}\left(\frac{x_{0j} - x_{1j}}{j_{\text{max}} - j_{\text{min}}}\right)^2} \tag{6.1}$$

6.2. Setup

This section describes how the experiments are set up. First the instances used in the experiments are described. Then, the setup of the experiments is discussed, which consist of varying the algorithm meta-parameters and the fitness function penalty factors.

6.2.1. Scenarios

Different DHSs have different characteristics and to know how well an algorithm can deal with these different characteristics different scenarios have been created. Depending on the values of these characteristics, different possible ways of operating the DHS more optimally exist. The most important characteristics and their influence on operations are:

- Consumer demand profile
  - How much demand is there? If there is much demand, more heat will have to be produced and the solution space will be smaller as low temperatures will result in demand not being fulfilled.
  - and when? If there is a large amount of demand early in the day and very little during the rest of the day, maintaining a constant supply temperature will be very inefficient.
- Network topology
  - The number of HTSs. As every HTS secondary supply temperature needs to be controlled as well, the number of decision variables increases.
  - The distance from producer to consumer. When the distance between producer and consumer increases, the propagation delay in the network increases as well. In addition to this, the losses in the network increase as water spends more time in the network before it reaches the consumer.
  - Number of producers in the network and their locations. Producers that are closes to consumers lead to fewer losses.
- Production

	GA	SaDE	CMA-ES
Population size	10 – 40	10 – 40	10 - 40
Exploration	0.05 – 0.4	DE/rand/1 DE/best/2	5 – 20
Combination	Fade-over, Plain	0.5	-
Initial solution	Random, Heating curve	Random, Heating curve	Random, Heating curve

Table 6.1: Algorithm meta-parameters

- Production and electricity costs. Dynamic production costs might make it cheaper to produce more energy earlier than less energy later by making use of the network storage effect.

To explore how the algorithms perform, scenarios with different values for these characteristics have been created. The heat exchangers used in the scenarios are described in Table A.1

S.1 Residential DHS

- A residential area with 450 houses with different distances from the production facility. 15 houses are grouped together into one consumer, resulting in 30 consumers in the network simulation
- Houses are reasonably insulated with a base load of 2 kW per home and a two demand peaks of 8.5 kW around 06:00 and and 5 kW around 18:00. These peaks are about 5 and 8 hours wide respectively. This demand profile is randomised by scaling it between 0.8–1.2 and shifting the peak one hour forward or back in time. For more details, please see Appendix A.1.
- The heat production costs and electricity prices are dynamic. See Table A.2 for the prices per hour.
- A topological plot of the grid is shown in Appendix A.1
- This scenario is a DHS with typical demand and production prices. Comparing this instance with the heating curve method can give a good indication of how much can be saved by optimising operations. It is likely the optimisation will make most savings by shifting production to times when production is cheaper than by reducing the amount of heat lost.

S.2 Residential areas with static prices

- Instead of dynamic prices the following prices are used:
  - Heat: 0.029 €/MWh
  - Electricity: 0.09 €/MWh

S.3 Residential areas with high insulation.

- The houses are well insulated with a base load of 1 kW per home and one demand peak of 6 kW around 06:00. This peak is 3 hours wide. The demand profiles are randomised like to before.
- A topological plot of the grid is shown in Appendix A.2. The main difference between these instances is the pipe diameter, which is smaller for well insulated homes.

S.4 Multiple producers

- The network contains two producers which are placed right after each other.
- Demand is multiplied by a factor 2
- A plot with the topology is shown in Appendix A.3.

	GA	SaDE	CMA-ES
Population size	20	20	20
Mutation	0.2	DE/rand/1 DE/best/2	5
Combination	Plain	0.5	-
Initial solution	Random	Random	Random

Table 6.2: Algorithm default meta-parameters

6.2.2. Experiment Settings

The optimisation algorithms all have meta-parameters that can be tuned. The influence these parameters have on the performance of the algorithm is explored by varying one of these parameters while fixing the others.

The possible parameter values used in the algorithms during the experiments are shown in Table 6.1. The default values, used when another variable is varied, are shown in Table 6.2.

Population Size

Population-based algorithms allow the user to specify how large the population should be. In addition to this, although the CMA-ES algorithm does not maintain a population, it does take a number of samples from its distributions. How many samples are taken is determined automatically based on the dimensionality of the problem, but it is possible to manually specify how many samples should be taken.

The population size range is chosen such that the recommended population sizes for CMA-ES fall within the range.

Mutation

The algorithms do mutation to find new search directions that might be interesting.

With GA, the mutation is controlled through the mutation probability, which indicates how likely a solution is to be mutated. The lower bound of the mutation probability range is chosen such that at least there is some mutation, as that is the only way the algorithm can do some exploration. The upper bound has been chosen such that there is not an absurd amount of mutation.

For SaDE, mutation is done by using a strategy to combine existing vectors from the population and combining them using a random value F. This F is drawn from a normal distribution with $\mu = 0.5$ and $\sigma = 0.3$ and clamping it between 0 and 2. Changing this value is not needed, as the algorithm manages the exploration automatically by adapting the strategy selection distribution based on the results achieved so far.

CMA-ES does not have the concept of mutation, but it is possible to specify the initial step size. Changing this step size influences how wide the distribution is that the algorithm samples from for new solutions. A higher value leads to more exploration, while a lower value leads to more exploitation. The step size is varied from 5 °C to 20 °C to explore how this influences the optimisation process. An initial step size of 5 °C seems a reasonable value, as this is the maximum temperature gradient in the instances used in the experiments. The higher step sizes are chosen such that the algorithm can more easily do exploration. The step size of 20 °C seems unnecessarily high, but it might still lead to interesting results.

Combination

Combination combines solutions found so far to find new solutions. As both solutions might contain parts that are good, combining them might result in new solutions that have good parts of both parent solutions.

For the GA different ways to do crossover exist, which have been explained in detail in Section 5.3.3. These different crossover methods and their effect on the quality measures are explored.

Constraint type	Penalty factor
Maximum edge velocity	10^2
Maximum temperature gradient	10^2
Minimum consumer supply temperature	$0.5 \cdot 10^2$
Energy stored in DHN at end of horizon	1
Unfulfilled demand	10^3

Table 6.3: Fitness function penalty factors

In SaDE crossover is done by randomly choosing elements from the mutant or the target vector depending on the *CR* values. The initial *CRm* value is set to be 0.5 and is adapted by the algorithm itself during the optimisation.

Initial Solution
To enable fair comparison in experiments where the performance of the heating curve and an optimised schedule are compared, heating curves are created that cause at least as much energy to be stored in the network as at the start of the horizon.

Convergence
To measure how fast an algorithm converges compared to another, the algorithms are assumed to have converged when there have been 30 subsequent iterations without improvement.

Fitness Function Constraint Violation Penalty Factor Values
As described in Section 3.5.2, the importance of constraints can be defined relatively by adding certain penalty factors for every type of constraint violation.

The penalty factors for every type of constraint violation are shown in Table 6.3. As the energy stored in the network at the end of the horizon is several orders of magnitude larger than any other constraint violation, violations of this kind are not multiplied by a large factor.

6.3. Hypotheses
This section discusses the hypotheses for the different experiments. First, the expected influence of the meta-parameters is discussed in Section 6.3.1. Then, the expected performance of the algorithms relative to each other is discussed in Section 6.3.2. Finally, Section 6.3.3 discusses the savings expected from optimising the DHS operations with respect to the heating curve based operations.

6.3.1. Meta-Parameter Influence
In this section the expected influence of the meta-parameters is discussed. Having an idea of what influence the meta-parameters have allows us to tune the algorithms to make a good balance between runtime and optimality.

Population Size
For all algorithms, increasing the population size will lead to more optimal solutions, but how much will differ for ever algorithm.

- The GA will benefit greatly from a bigger population, as this will increase the number of variables randomly replaced, which increases the probability of finding a good value.
- SaDE will also benefit from a higher population size, as it gives the search more points in the search space to search from. This will allow the search to search through more optima that it might get stuck in, which increases the probability of one of those optima being the global optimum.
- For CMA-ES, the recommended population size is $4+3\lfloor \ln D \rfloor$, however higher values are allowed. It is expected that increasing the population size will have some beneficial effects, but this will effect will diminish quite quickly.

Exploration

As described in the previous section, the exploration parameters are conceptually quite different between all algorithms. As such, no comparisons are made between algorithms given the different meta-parameter values.

- The GA explores the search space by randomly replacing values in solutions. This means that new values cannot enter the population if there is no mutation. Increasing the mutation probability will thus allow the search to do more exploration, which is the only way the algorithm can find new values that might work better.
- The CMA-ES algorithm samples from its distribution to do exploration. The step size controls how far from the mean the algorithm samples. Increasing the initial step size allows the algorithm to make bigger steps at the start of the optimisation. Tuning this parameter manually is not likely to be useful, because the algorithm adapts this step size automatically based on the samples it takes.

Combination

The only algorithm that has configurable combination parameters is the GA.

- The genetic algorithm can be configured with which crossover operation is used. It is expected that fade-over will reach a valid solution in fewer iterations than normal cross-over, but this solution will be less optimal. Fade-over might reduce the amount of exploration done by the algorithm, which means it can get stuck in a local optimum more easily.

Initial Solution

Which initial solution is used is likely to have a large influence on the algorithm performance, as it can bias the algorithms toward a local optimum that is near the initial solution. This might get the algorithms stuck in that local optimum, while better optima might exist.

- Random initialisation will result in best results but will be slower to converge because it allows the algorithms to search through more of the solution space before converging.
- A solution based on the heating curve will be a good starting point, but the optimisation will likely end in a local optimum. The algorithms will improve on the initial solution, as the heating curve is not a local optimum. By lowering the temperatures at certain times, savings can already be made without violating constraints, which makes it easier for the algorithms to move further toward an optimum.

6.3.2. Algorithm Performance

This section discusses the expected performance of the algorithms relative to each other. Exploring this allows the user of the algorithms to make an informed decision on which algorithm is best to use for this problem.

In order of decreasing expected optimality performance:

- SaDE is expected to perform best as it maintains a population of solutions instead of a single solution. This allows the algorithm to explore points that are more widely spread throughout the search space than the single mean point CMA-ES samples around.
- CMA-ES maintains a covariant normal distribution which it samples to search for more promising solutions. As the algorithm adapts this distribution based on the measured fitness values, it makes a more informed search through the search space. However, since the algorithm moves towards the optimum it has sampled around, it might miss other better optima.
- The naive GA will be the worst algorithm as it depends heavily on the mutation operation, which has to be lucky to find a good value for variables. The algorithm does not really do an informed search of the solution space, which means a lot of operations will be wasted that and certain areas of the solution space might not be explored at all.

6.3.3. Optimisation Savings

This section describes how much improvement is expected by optimising the operations of a DHS with respect to a heating curve considering different scenarios.

Based on the different scenarios, the optimisation is expected to make different amounts of savings.

S.1 Residential with dynamic prices. It is expected that the optimisation will lead to significant savings for this scenario, as is able to make use of the network storage effect than the heating curve does. The heating curve forces producers to always output the same temperature during the day, which means it will be more expensive to produce heat when prices are higher. The optimised schedule can avoid this, by lowering the amount of heat produced at the most expensive times.

S.2 Residential with static prices. As the heating curve needs to make the temperatures high enough to fulfil demand at peak times, there are times when the temperatures are unnecessarily high. This means that optimising the operations, will lead to more savings, as less losses will occur in the network at the times between the demand peaks. Since prices are static, the optimisation will likely not make much savings in terms of price, but the total losses will likely be lower than both the heating curve based operations and the case with dynamic prices.

S.3 Highly insulated residential. The insulation of the homes cause the homes to only need heating once every day in addition to heat needed for tap water. This means that keeping the temperature high throughout the day is wasteful. As such, it is expected that even more savings can be made compared to the normal case.

S.4 Residential with multiple producers. There are two producers in this scenario, but the demand has also been increased. This means it is not likely more savings can be made in this scenario compared to **S.1**. However, it is expected that the optimised schedule lowers supply temperatures more than the heating curve does, as the producer supply temperatures can be lowered when there is no demand peak.

7 Results

This chapter describes the results from the experiments and makes some preliminary conclusions. To increase statistical significance every experiment is run at least 5 times and the results are averaged when appropriate.

Before the discussion of the results, Section 7.1 gives a short description of a problem with the fitness function that was encountered while running the experiments and a solution for this problem.

Then, the next three sections of this chapter discuss the most interesting results from the algorithms: GA in 7.2, SaDE in 7.3, and CMA-ES in Section 7.4. The relative performance of the algorithms is discussed in more detail in Section 7.5.

Next, Section 7.6 explores creeping behaviour that has been observed and attempts to find a solution for this behaviour, which should allow the algorithms to converge much more quickly.

To conclude this chapter, Section 7.7 discusses the savings obtained by optimising the DHS operations compared to the heating curve based operations.

7.1. Preliminary

Some preliminary remarks are given in this section.

Section 7.1.1 describes a problem with the fitness function that was found during the experiments. Section 7.1.2 gives an overview of the number of valid solutions found in every experiment.

7.1.1. Constraint Violation Penalty Preferred Over Increasing Costs

While running the experiments, something unexpected happened. For some instances it was found that optimisations would not find valid solutions, while the instances were verified to be feasible by manually creating a heating schedule. Apparently, the optimisations would find solutions that had (a small amount of) constraint violations to save on monetary cost, rather than finding a solution with no constraint violations. The amount of constraint violation had become an order of magnitude smaller than the monetary cost, which meant increasing temperatures would increase costs more than solving the constraints would decrease the penalties.

To solve this problem, in addition to the penalty incurred proportionally with the amount of constraint violation, a constant amount of penalty is added when any amount of constraint violation occurs, essentially creating a piecewise linear step function, as shown in Figure 7.1. The constant offset has been chosen to be an order of magnitude larger than the typical monetary cost for the instances.

7.1.2. Number of Valid Solutions per Experiment

This subsection contains various tables with the number of valid solutions found by the algorithms in every experiment.

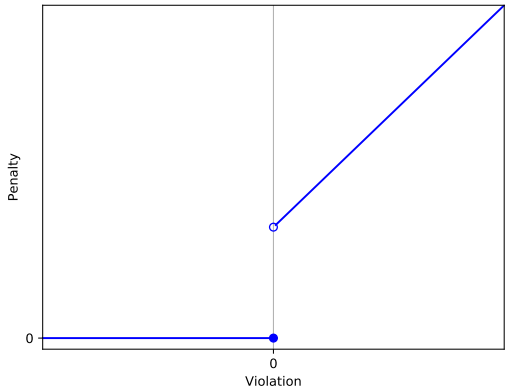


Figure 7.1: Piecewise linear step penalty

Table 7.1 contains the data for varying population sizes.

Population size	10	15	20	25	30	35	40
S.1							
GA	0/5	2/5	2/5	2/5	1/5	3/5	5/5
SaDE	6/6	6/6	5/6	5/6	6/6	3/6	6/6
CMA-ES	5/5	4/5	4/5	5/5	3/5	4/5	4/5
S.2							
GA	2/5	1/5	1/5	1/5	4/5	5/5	4/5
SaDE	5/5	4/5	4/5	4/5	4/5	4/5	3/5
CMA-ES	5/5	5/5	5/5	4/5	5/5	5/5	5/5
S.3							
GA	1/5	3/5	2/5	3/5	4/5	5/5	5/5
SaDE	5/5	4/5	5/5	5/5	4/5	2/5	4/5
CMA-ES	5/5	4/5	5/5	5/5	3/5	5/5	3/5
S.4							
GA	0/5	0/5	0/5	1/5	1/5	2/5	1/5
SaDE	2/5	3/5	4/5	3/5	5/5	1/5	3/5
CMA-ES	0/5	1/5	2/5	4/5	4/5	3/5	3/5

Table 7.1: Number of experiments that resulted in valid solutions with varying population size

For varying variation parameters, the data is shown in Table 7.2.

GA: Mutation probability	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40
S.1	0/5	0/5	3/5	2/5	3/5	2/5	4/5	3/5
S.2	0/5	0/5	3/5	1/5	2/5	4/5	4/5	4/5
S.3	0/5	0/5	2/5	2/5	3/5	3/5	3/5	3/5
S.4	0/5	0/5	0/5	0/5	0/5	1/5	0/5	1/5

CMA-ES: Variation	5 °C	10 °C	15 °C	20 °C
S.1	4/5	5/5	5/5	3/5
S.2	5/5	5/5	5/5	3/5
S.3	5/5	5/5	5/5	4/5
S.4	1/5	3/5	0/5	0/5

Table 7.2: Number of experiments that resulted in valid solutions with varying variation

The influence of the different crossover operators is shown in Table 7.3.

GA: Crossover	Time	Fade-over
S.1	2/5	3/5
S.2	1/5	4/5
S.3	2/5	5/5
S.4	0/5	0/5

Table 7.3: Number of experiments that resulted in valid solutions with varying crossover

With varying initial solutions, the results are shown in Table 7.4.

Initial solution	Random	Heating curve
S.1		
GA	2/5	5/5
SaDE	5/6	5/5
CMA-ES	4/5	5/5
S.2		
GA	1/5	5/5
SaDE	4/5	5/5
CMA-ES	5/5	5/5
S.3		
GA	2/5	5/5
SaDE	5/5	5/5
CMA-ES	5/5	5/5
S.4		
GA	0/5	5/5
SaDE	4/5	5/5
CMA-ES	2/5	5/5

Table 7.4: Number of experiments that resulted in valid solutions with varying initial solutions

7.2. GA

This section discusses the behaviour of the genetic algorithm and the influence the different meta-parameters have on it. Only the most interesting results are discussed here, the full results can be found in Appendix B.

This Section presents to interesting conclusions on the GA. The first makes a case for increasing exploration of the algorithm. The second argues that using an existing initial solution is not always a good idea.

7.2.1. Increased Exploration Improves Performance

There are multiple ways to increase exploration with the genetic algorithm.

Population Size

The first way is by increasing the population size. As shown in Figure 7.2 the performance improves when the population size increases. Because there are more individuals in the population, there are more opportunities for finding new values through mutation.

It is interesting to note that the genetic algorithm is unable to find solutions for **S.4** with even reasonably large population sizes. Even with the largest population size, this the algorithm is often not able to find one. This is likely due to the increased dimensionality.

As can be seen in Figure 7.3a, the number of iterations before the algorithm converges increases as the population size increases. This is due to the fact that the algorithm has more opportunity for exploration, which causes the algorithm to find improving solutions for longer, delaying termination. In

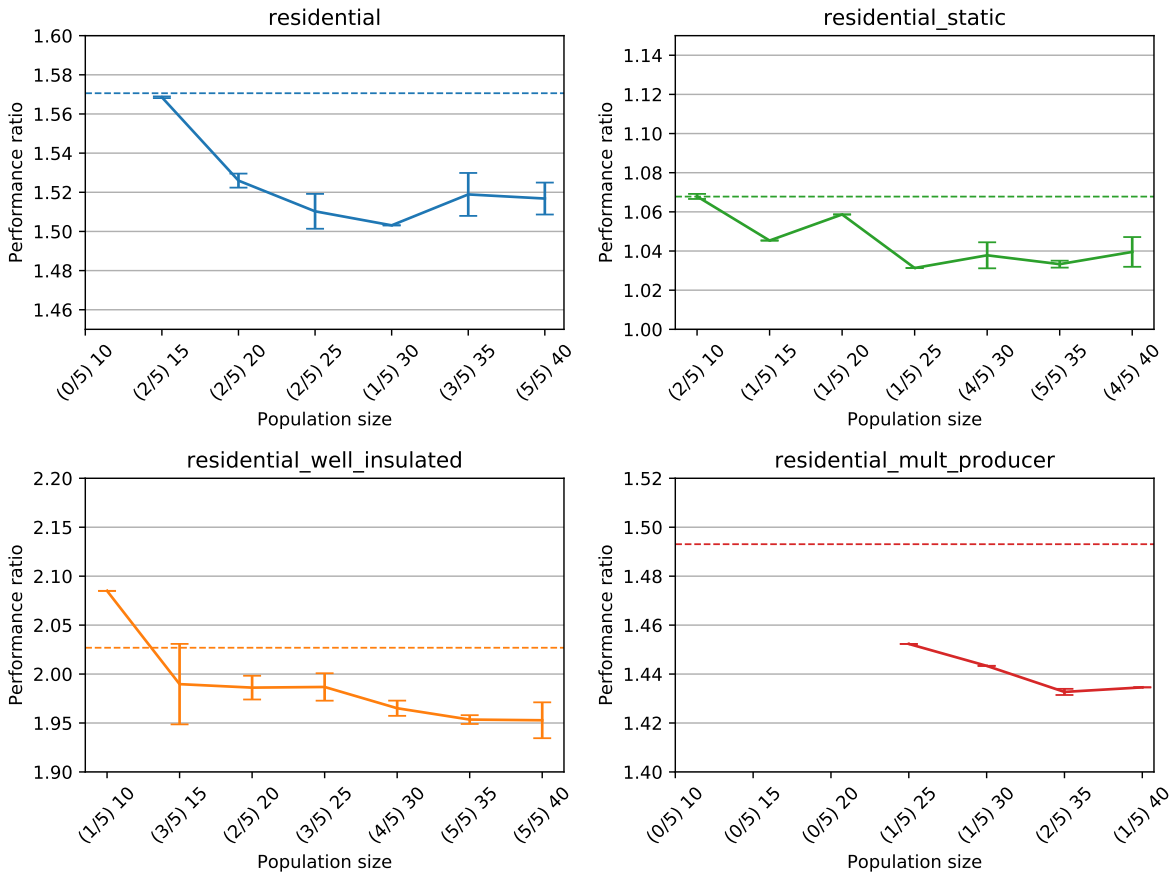


Figure 7.2: Performance ratios of genetic algorithm with varying population size

addition to this, the increased exploration also causes the algorithm to find its first valid solution earlier, as can be seen in Figure 7.3b

Mutation Probability

The second way to increase exploration is by increasing the mutation probability. The effects of this on the performance ratios are shown in Figure 7.4. As was expected, by increasing the mutation probability, the performance of the algorithm increases as well.

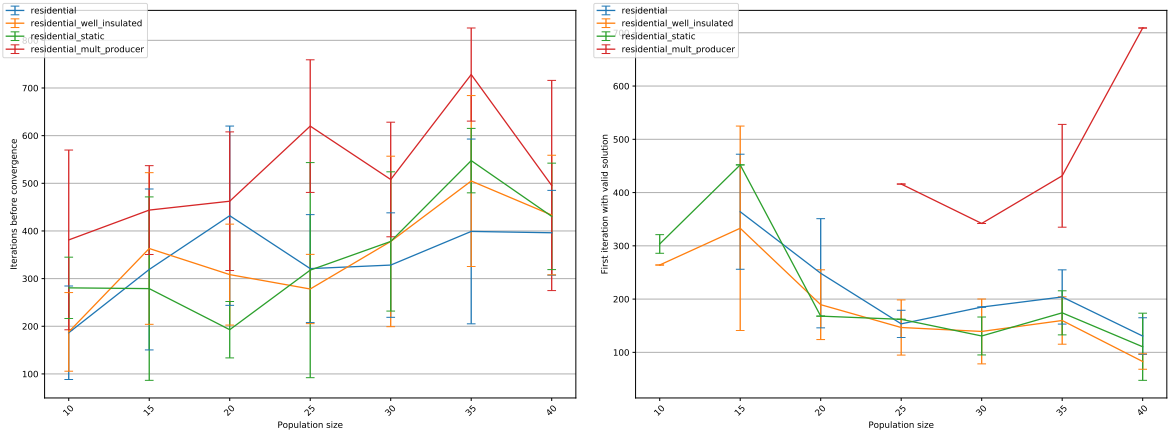
It is interesting that **S.2** is less affected by the mutation probability, which is likely due to the smaller number of local optima in this instance. This instance has fewer local optima due to the static prices, which means the algorithm does not have to make a decision on producing more, earlier, but more cheaply, or producing at a later time with fewer losses. This means that more exploration is no longer necessary. This effect is also seen in the population size results, as the increase in population size no longer improves performance when it is increase above 30.

Crossover Operations

The final option to increase exploration is to use the fade over operation instead of the plain crossover over time. This fade over creates new values as it fixes constraint violations during crossover. Because of the additional source of exploration, the algorithm is able to find new values more easily.

In Figure 7.5a the effects of the two different crossovers is shown on optimality is shown. The fade over results in valid solutions much more often than the plain crossover. This makes sense, as the fade over solves constraint violations explicitly, which creates offspring that are much more likely to be accepted and guides the algorithm towards the feasible solution space.

With the plain crossover the amount of temperature gradient constraint violation could lead to a much higher penalty, which would lead to the offspring likely being rejected. As the fade over solves this



(a) Number of iterations before convergence (b) First iteration with valid solution

Figure 7.3: Iterations before convergence and the first valid solution for the genetic algorithm with varying population size

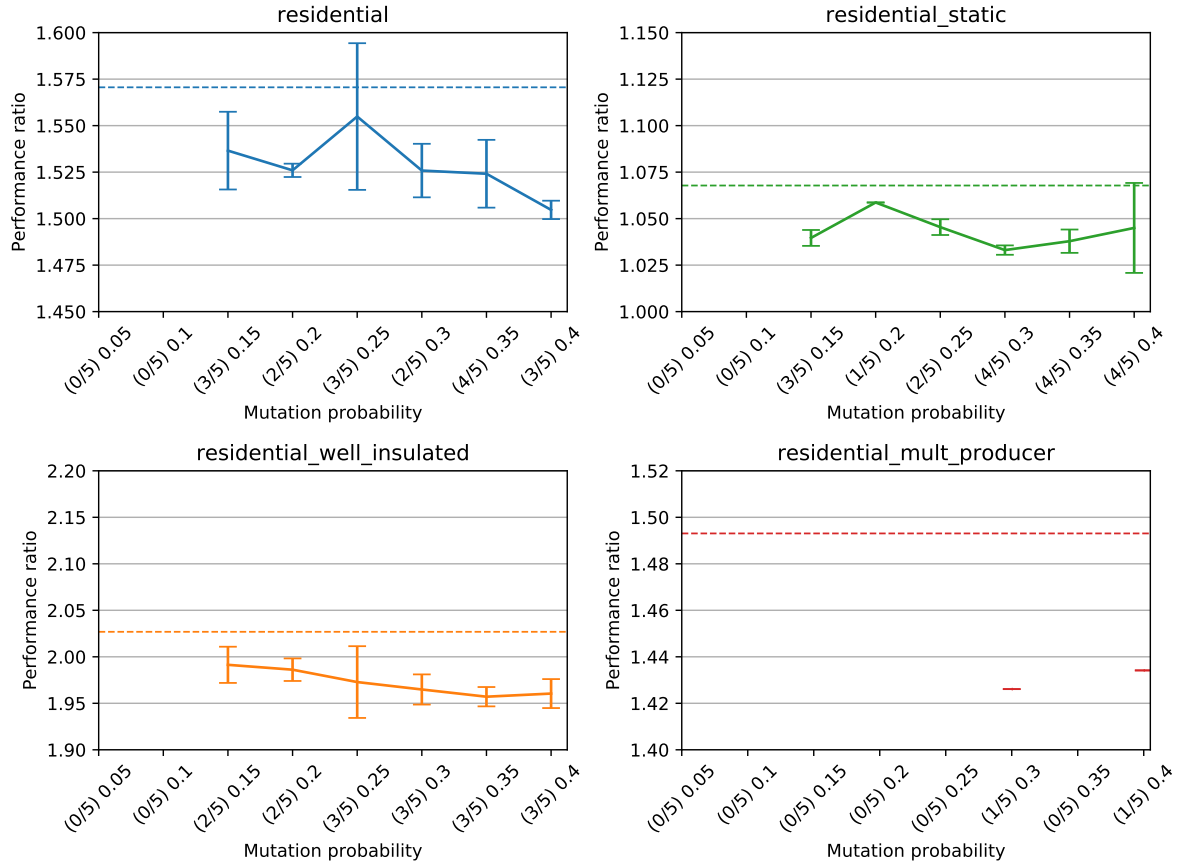
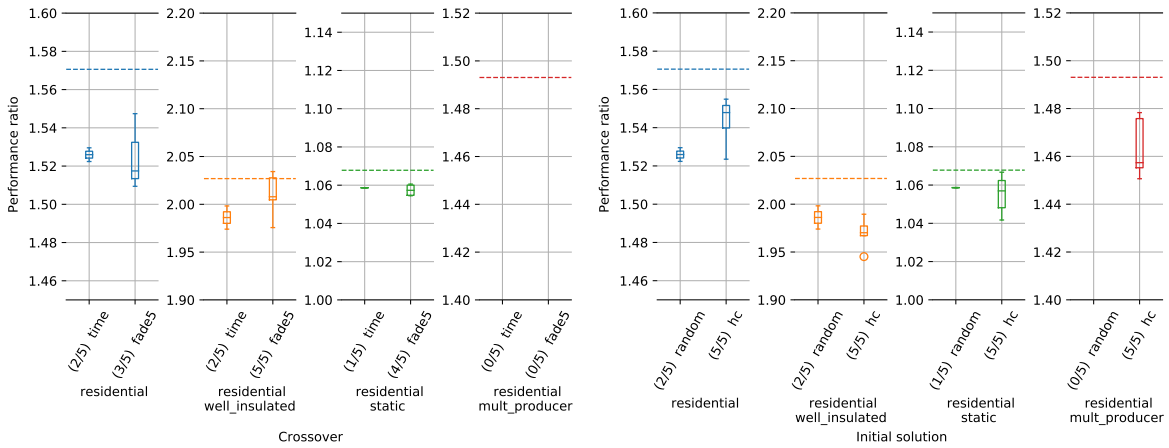


Figure 7.4: Performance ratios of genetic algorithm with varying mutation probability



(a) Varying crossover operators

(b) Varying initial solution

Figure 7.5: Performance ratios for genetic algorithm

constraint explicitly, and removes possible constraint violations of times before and after the crossover point, it finds valid solutions much more easily.

It is also interesting to note that even with this new crossover operator, the algorithm is still unable to find solutions for **S.4**. By inspecting the solutions manually, it becomes clear that the algorithm is unable to find valid valve settings that allow the producers to fulfil demand. The valve settings are not balanced enough, which causes too much mass flow to be assigned to one producer.

7.2.2. Initial Solution Causes Getting Stuck in Local Optimum

The final meta-parameter is the initial solution. The influence of this meta-parameter is shown in Figure 7.5b. As expected, starting with a valid solution allows the algorithm to always find a valid solution. As was also expected, the algorithm does get stuck in a local optimum sometimes, although the increase in the number of valid solutions found, might will be worth it.

7.3. SaDE

In this Section the most interesting results about the SaDE algorithm are discussed. The full results can be found in Appendix B.

7.3.1. Large Populations Cause Bad Performance

In Figure 7.6 the influence of the population size on the optimality of the results is shown. The population size seems to have an influence on the results, especially for **S.1**.

When the population size is increased too much, the quality of the solutions goes down and the number of valid solutions that are found decreases as well. This indicates that the increase in exploration causes the algorithm to move to the global optimum too slowly, getting stuck in local optima before the algorithm is terminated.

This is supported by the fact that the algorithm terminates earlier with an increasing population size, as shown in Figure 7.7a. The algorithm does not find improving solutions because it explores too much of the search space, causing the algorithm to have difficulty converging towards one solution. The lack of improving solutions, causes the algorithm to terminate. This is also supported by the fact that the algorithm finds it first valid solution later, when the population size is increased, as shown in Figure 7.7b.

7.3.2. Initial Solution Improves Performance

As was expected, starting with an already valid solution causes the algorithm to always find a valid solution. It was also expected that the algorithm might end up in a local optimum, but this does not seem to be the case. As shown in Figure 7.8a, the results are comparable or better when the heating

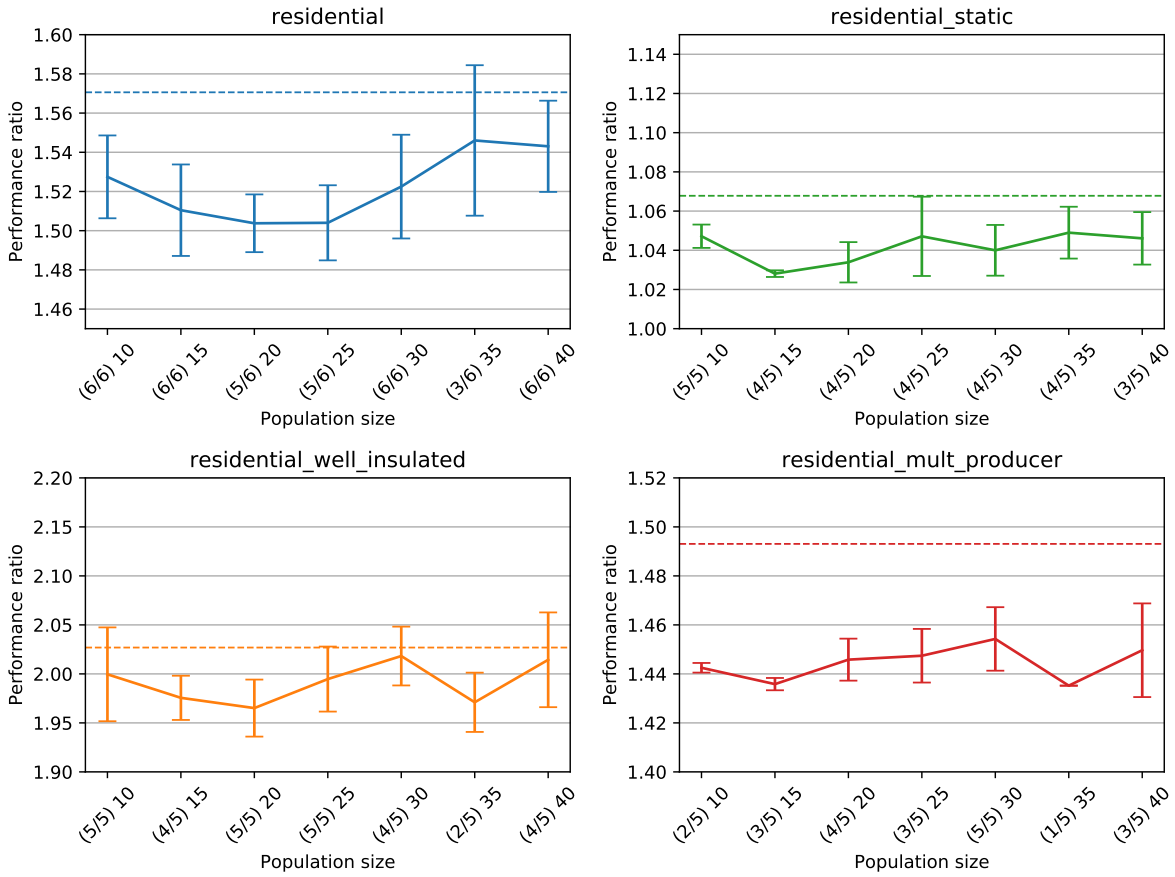
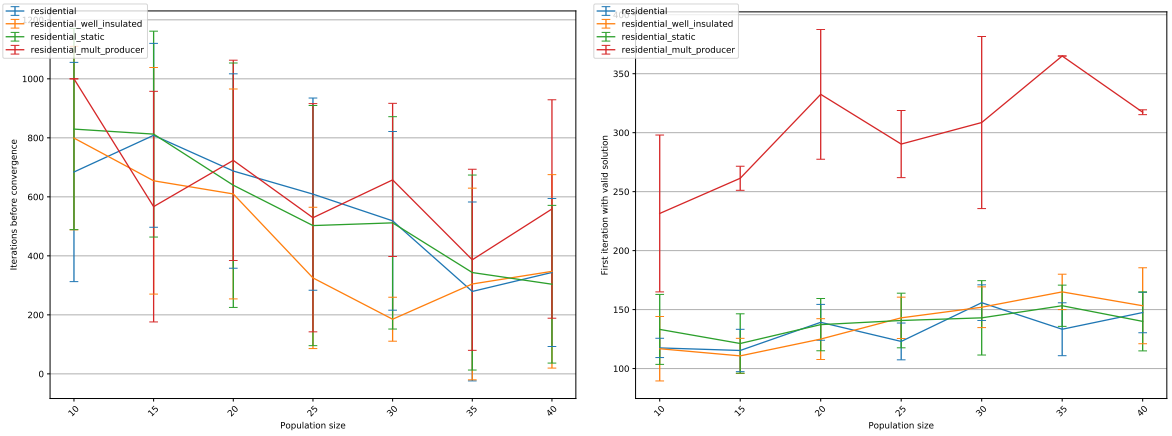


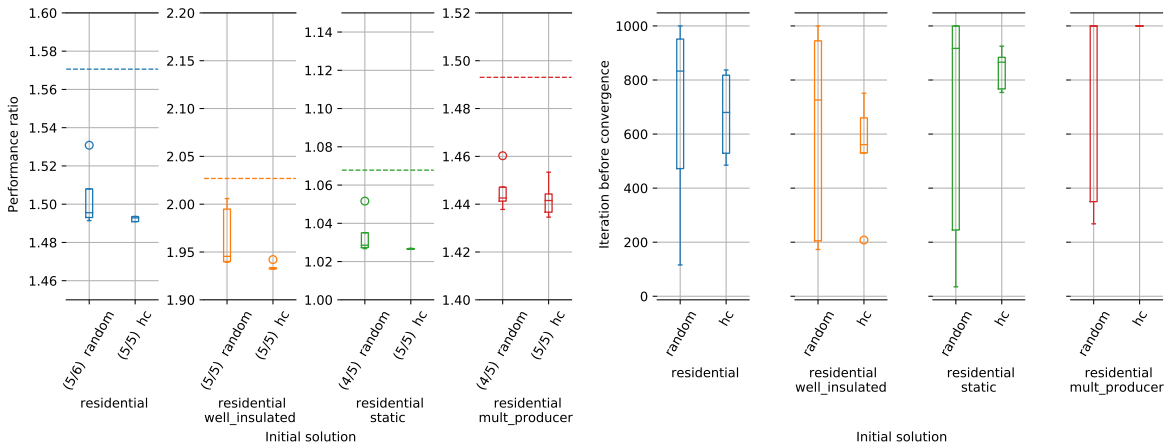
Figure 7.6: Performance ratios of SaDE with varying population size



(a) Number of iterations before convergence

(b) Iteration with the first valid solution

Figure 7.7: Oversized SaDE population causes too slow convergence towards optimum with varying population size



(a) Performance ratios

(b) Iterations before convergence

Figure 7.8: SaDE performance with varying initial solutions

curve solution is used as the initial solution.

What is unexpected is that even with an existing solution as starting point, the algorithm still converges quite slowly. This is shown in Figure 7.8b. For **S.4**, the algorithm is consistently terminated due to the algorithm having run for the maximum amount of iterations (1000). This suggests that the algorithm keeps making small improvements, which delays termination. This behaviour is further explored in Section 7.6.

7.4. CMA-ES

In this section the performance of the CMA-ES algorithm with respect to the various meta-parameter values is discussed. Only the most interesting results are discussed. The full results can be found in Appendix B. In the first subsection the influence of the sample size is investigated. The next subsection investigates the influence of the initial step size. Finally, the influence of the initial solution is discussed.

7.4.1. Sample Size

In Table 7.1 the number of valid solutions found by the algorithm for every instance is shown.

Increasing Sample Size Too Much Causes Too Much Exploration

As can be seen in Figures 7.9 and 7.10a, the influence of the different sample sizes depends strongly on the instance.

For **S.1**, the increase in sample size causes the algorithm to perform worse both in terms of consistency and optimality. This is likely caused by the increase in exploration, which causes the algorithm to spend too much time on local optima, causing it to get stuck. This makes it harder to move toward the global optimum. This conclusion seems to be supported by the decrease in consistency, illustrated in Figure 7.10a.

With static prices, the algorithm appears to benefit from an increased sample size and eventually seems to converge to only one solution. In this case, the algorithm is not distracted by local optima to explore, which were created by the choices between producing more, earlier, but cheaper, or less, later, but more expensive.

This is likely due to the decrease in the number of local optima, which means the algorithm can less easily get stuck.

The performance of the well insulated instance is fairly stable when the sample size is increased, until the sample is increased to 40, when the algorithm's performance becomes much worse. It is interesting to note that the performance and consistency seem best when the sample size is 35. This is surprising, as the algorithm performance decreases with an increasing sample size for scenario **S.1**.

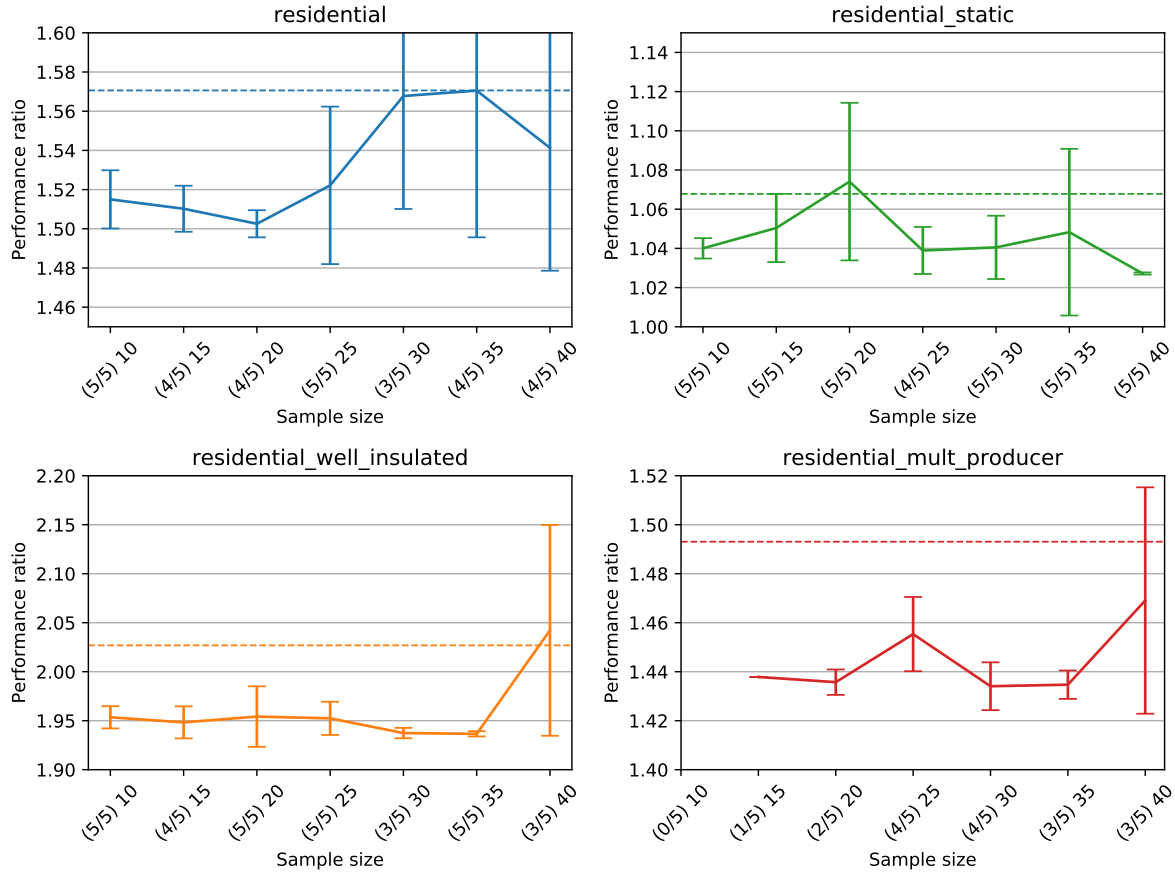
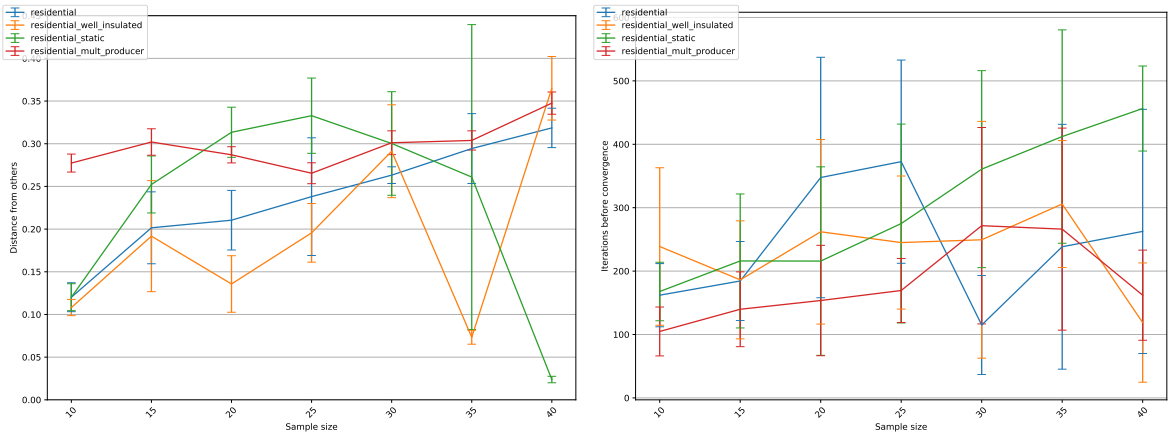


Figure 7.9: CMA-ES performance ratios with respect to the sample size. The number between the braces in the labels indicate the number of valid solutions found



(a) CMA-ES distances to other solutions. Both valid an invalid solutions are considered in the data. Lower results imply the algorithm becomes more consistent, as the distances between the solutions decrease.

(b) Number of iterations before convergence

Figure 7.10: CMA-ES consistency and number of iterations before convergence with respect to the population size

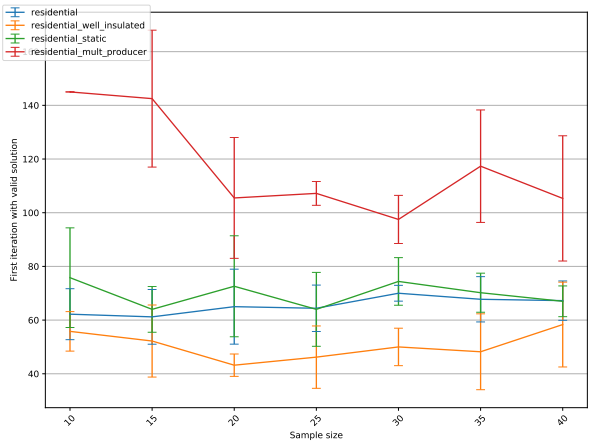


Figure 7.11: CMA-ES first generation with valid solution

This sudden deviation from the trend needs to be investigated further, as it seems likely to be caused by an insufficient number of samples. From Figure 7.10b it does indeed appear that the algorithm gets stuck in local optima, as the algorithm terminates earlier, which means the algorithm is unable to make improvements which would postpone termination.

For the instance with multiple producers, the algorithm seems to need a sample size of at least 25, as the algorithm is otherwise unable to find solutions consistently. This makes sense as the number of decision variables for this instance is three times larger than for the other instances, due to the algorithm needing to find temperatures for 2 producers and valve settings for 1 valve.

First Valid Solution Insensitive to Population Size

Interestingly, the CMA-ES algorithm seems to be fairly insensitive to the population for the instances with one producer when considering the first iteration in which a valid solution appears. This is shown in Figure 7.11

This is likely due to the algorithm being able to build a fairly accurate distribution with few samples. In addition to this, the penalties for constraint violations grow steeply when violations are increased. This means the algorithm will be steered towards valid solutions quite aggressively.

For the instance with multiple producers, the algorithm finds valid solutions earlier with larger population sizes, although this only seems to happen for sample sizes smaller than 20. Interestingly, this is exactly the population size that is recommended for CMA-ES for instances of this dimensionality. The recommended sample size, as described in Section 5.5, is $4 + \lfloor 3 \ln 72 \rfloor = 16$. When sample sizes smaller than this are used, finding the first valid solution takes considerably longer.

7.4.2. Initial Step Size

In Figure 7.12 the performance ratios of the solutions found by the algorithm with a varying initial step size are shown. As can be seen, the initial step size influences the performance of the algorithm considerably. Not only in the optimality of the solutions found, but also in the number of solution that are found. When the step size is increased too much, the algorithm finds a valid solution much less often.

It is hard to make conclusions on what is “the optimal” initial step size as this seems to vary from instance to instance and the algorithm is not as sensitive for every instance.

7.4.3. Initial Solution

In Figure 7.13 the performance ratios of the solutions found by the algorithm with varying initial solutions are shown. As expected, the algorithm is always able to find a solution when a valid initial solution is used. This makes sense, as moving to an invalid solution would result in high penalties.

However, it was also expected that the algorithm would get stuck in local optima when an existing

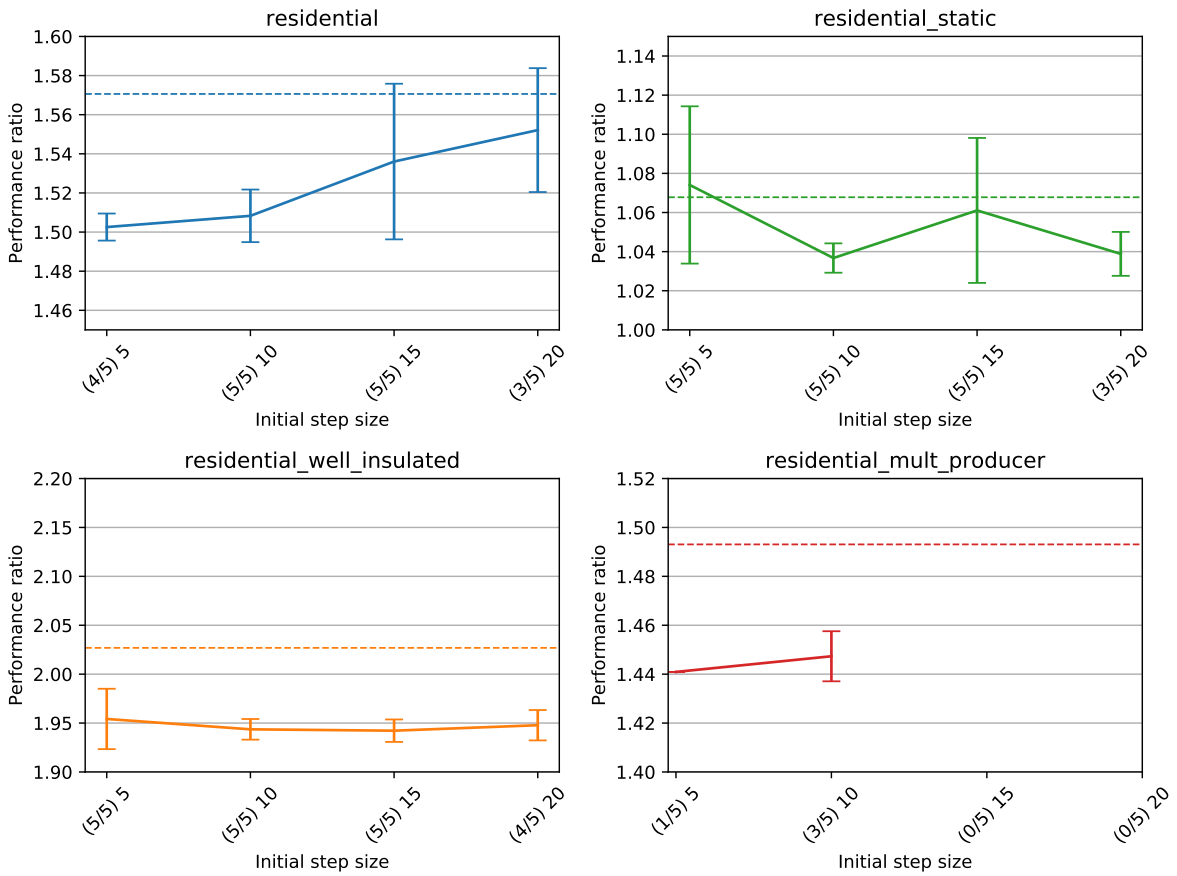


Figure 7.12: CMA-ES performance ratios with respect to varying initial step size

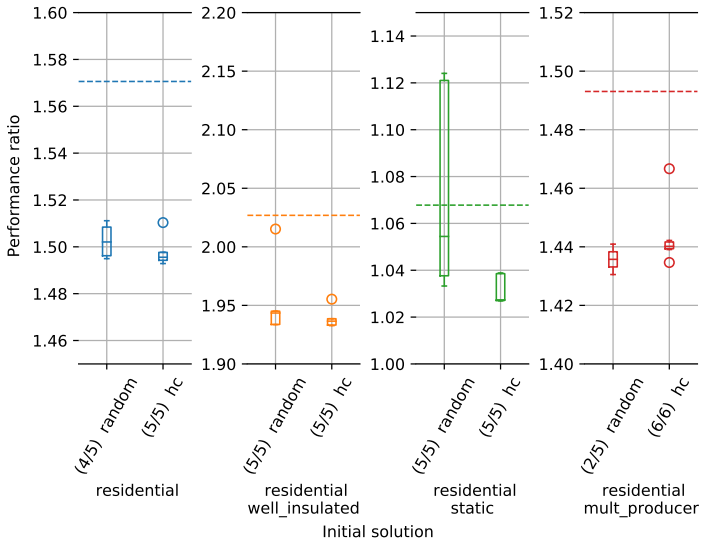
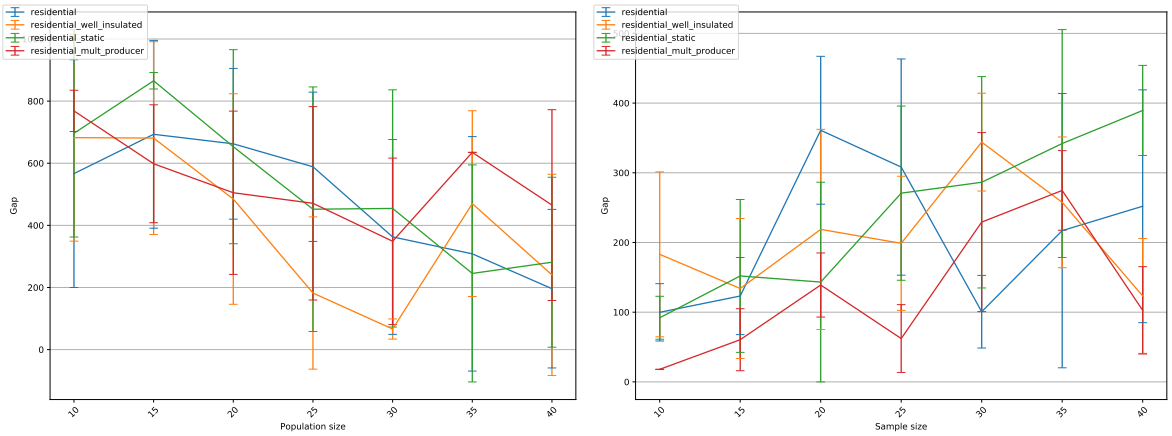


Figure 7.13: CMA-ES performance ratios with varying initial solutions



(a) SaDE (b) CMA-ES

Figure 7.14: Number of iterations until termination after the first valid solution has been found

solution was used, which does not always seem to be the case. The results are usually comparable or better than the solutions found with random initialisation.

7.5. Relative Performance

From the results in the previous sections it becomes apparent that the genetic algorithm does not perform as bad as was expected, although the solutions it finds are far from optimal.

SaDE performs quite well in terms of optimality as it is able to outperform all other algorithms. However, this does come at the expense of convergence speed. As shown in Figure 7.7a, SaDE does not always converge and thus is terminated after the maximum number of iterations is reached. This does not mean that the algorithm jumps around the search space, but that it simply keeps making improving moves. This was also further discussed in Section 7.6.

The CMA-ES algorithm was able to find quite optimal solutions, which are usually close to the SaDE solutions. Although the solutions are a little less optimal, the algorithm converges much more quickly. This might be preferable in an MPC setting, as the optimisation can be run more often.

The first question, **RQ 1**, posed in the introduction questioned which metaheuristic approaches would work well for the DHS problem. From the results in this chapter it becomes clear that metaheuristics designed for combinatorial optimisation problems do not work very well. Because of the limited ability to explore new values, the algorithm often gets stuck in local optima or is completely unable to find valid solutions. The real-valued metaheuristic approaches are much better suited and are able to get very close to the global optimum of the problem. This unfortunately does come at the price of slow convergence, which has to be investigated further before they can be used in an online setting.

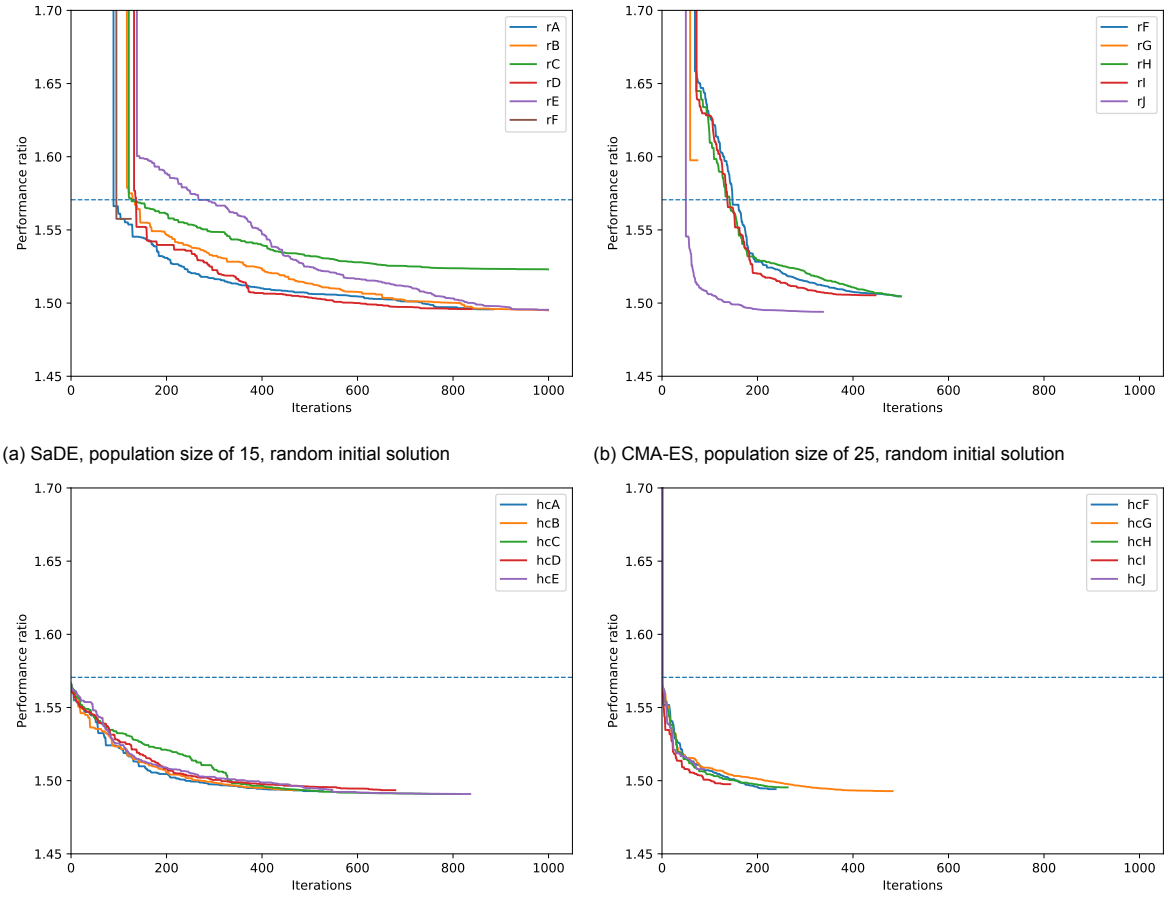
7.6. Creeping Behaviour

Although SaDE and CMA-ES both find quite optimal solutions, they are both slow to converge. In Figure 7.14 the number of iterations the algorithm still runs after the first valid solution has been found is shown. As can be seen, the algorithm still run for quite a while before termination.

In Figure 7.15 the fitnesses of the best individuals in the populations are shown. For CMA-ES the results for the population size experiments with a population size of 25 are taken. For SaDE the results for a population of 15 are taken.

As can be seen in these figures, the algorithms are able to solve the constraints quite quickly, but then take many iterations to optimise the solution further. This is most unfortunate with SaDE as it is able to beat CMA-ES in terms of optimality, but it converges much more slowly.

When an existing initial solution is used, the initial starting point is feasible, thus no constraints have to be solved. This does not seem to speed up the algorithms by much, though, as convergence is still



(a) SaDE, population size of 15, random initial solution

(b) CMA-ES, population size of 25, random initial solution

(c) SaDE, heating curve initial solution

(d) CMA-ES, heating curve initial solution

Figure 7.15: Algorithm performance ratio over time for **S.1**. The dotted blue line indicates the heating curve performance. Traces with 'r' or 'hc' as a prefix are randomly initiated or with a heating curve solution, respectively

quite slow.

To understand why this happens, the evolution of solution hcF is shown in Figure 7.16 for CMA-ES. As can be seen, the first solution is adjusted quite a lot with respect to the initial solution. From there, the differences between the solutions become smaller, but the shape of the differences remain similar. This raises the question of why it takes so long for the optimisation to make all those small steps instead of just making one big step.

One possible explanation is that the optimisation is constrained by the temperature gradient constraint. This would make it difficult for the optimisation to make big steps in the right direction, as bigger steps mean that the step direction needs to be precise to prevent ending up in at an infeasible solution. To verify this hypothesis, the experiment is rerun without the temperature gradient constraint. This will likely result in an infeasible solution, but it might offer some insight into the algorithm's convergence.

The convergence speed of the optimisation with the temperature gradient constraint disabled is shown in Figure 7.17. From this it becomes apparent that the CMA-ES algorithm does indeed converge much more quickly when the constraint is disabled. The SaDE algorithm still converges quite slowly.

In Figure 7.18 the evolution of tempF is shown. It does indeed appear that the algorithm is able to make bigger steps when the constraint is disabled.

It might be interesting to create a new optimisation procedure that runs in two phases. First, the algorithm optimises without the temperature gradient solution. When this has converged, the constraint is activated again and the optimisation is run to fix the constraint violation.

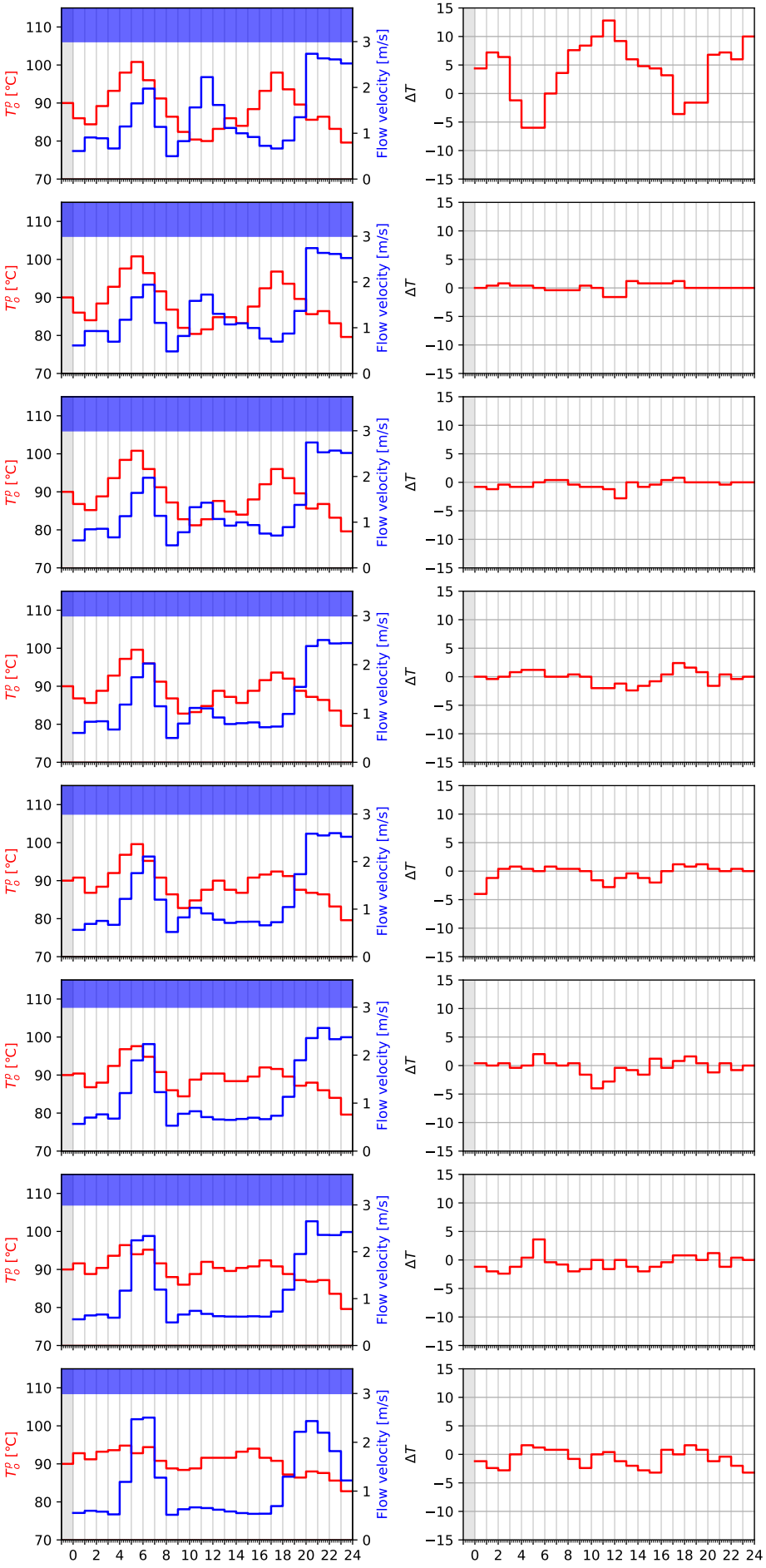
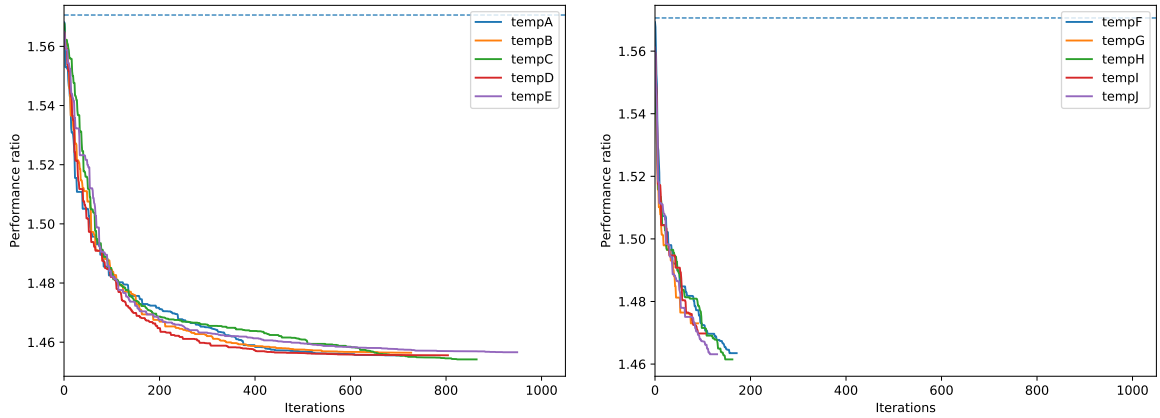


Figure 7.16: The progress of finding the hcF solution over time. The last row represents the final solution and the first row the 12.5% solution. The rows in between are equally spaced over the iterations. The left column is the solution and the corresponding mass flows. The right column represents the difference between the current solution and the previous one.



(a) SaDE (b) CMA-ES

Figure 7.17: Performance ratio over time for S.1 with the heating curve initial solution and no maximum temperature gradient constraint. The 'temp' prefix indicates the lack of temperature gradient constraint

A different possible solution for this might be to encode the problem differently. The encoding used in the algorithms in this research simply encode the absolute temperature settings for every producer. It might however work better to encode the solutions as relative temperature changes. This would remove the explicit temperature gradient constraint, which might make it easier to move towards the optimum. However, it might also make this problem harder to optimise as variables become much more dependent on each other. A small change at the start of the optimisation horizon will change the entire solution instead of just a small part.

7.7. Optimisation Savings

In this final section the savings achieved by the optimisation are described. It assumed that the DHS is currently operated using a heating curve. The optimisations are done assuming the amount of energy in the DHS at the end of the optimisation is at least as much as at the start of the horizon.

The best solutions found in Sections 7.3 and 7.4 are used as the minimal optimised costs.

The heating curve has been generated using the method described in Section 5.1.2.

Every subsection will discuss the savings achieved for a scenario as they are described in Section 6.2.1. In Table 7.5 the savings for every scenario are shown.

7.7.1. Residential

The residential scenario S.1 requires a heating curve of 92 °C to fulfil all demand and ensure that the same amount of energy is present in the network at the end of the horizon.

Comparing the costs between the heating curve solution and the optimised solution, the optimisation leads to a 5% cost reduction. The optimised heating schedule is shown in Figure 7.19 and the corresponding cost analysis in Figure 7.20.

As shown in Table 7.5, a reduction of 2.99 % in heat production was reached, but a reduction of 5.07 % in costs was reached. This is possible to due the optimisation rescheduling heat production to produce more during cheaper times and less when it is expensive.

When looking at the schedule effects on the DHS it becomes clear that it clearly attempts to prevent producing heat at times when it is most expensive. From 02:00 until 06:00 the optimised schedule produces more heat than the heating curve schedule does, but from 06:00 until 11:00, the optimised schedule produces less heat than the heating curve schedule. This makes sense, as the first time window contains the cheapest times to produce heat and the second are among the most expensive. This same phenomenon occurs between 13:00 until 18:00 and 18:00 until 23:00.

From Table 7.5 it also becomes clear that the optimisation has increased mass flows, resulting in higher

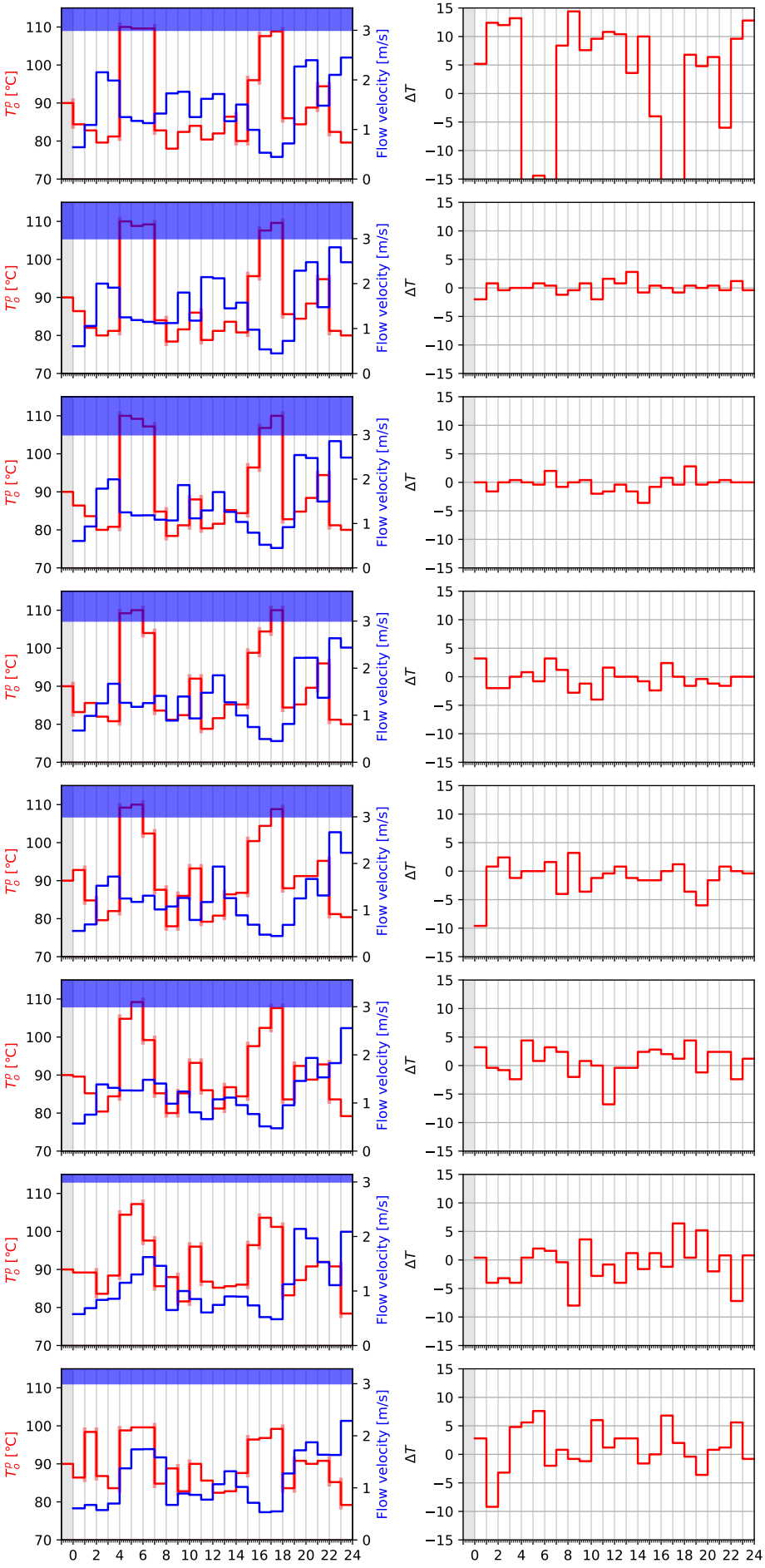


Figure 7.18: Algorithm performance ratio over time with deactivated temperature gradient constraint. Note that these solutions are not valid.

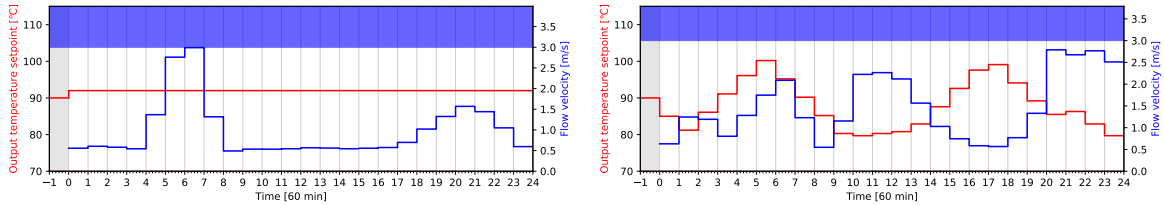
	Heating curve	Optimised	Difference	% Difference
S.1 Residential				
Stored in network at end	20.83 MW h	8.22 MW h	−12.61 MW h	−60.55 %
Losses	176.51 MW h	172.94 MW h	−3.57 MW h	−2.02 %
Produced heat	541.38 MW h	525.19 MW h	−16.19 MW h	−2.99 %
Pump electricity usage	8.01 kW h	17.14 kW h	9.13 kW h	113.98 %
Heat cost	€ 7225.04	€ 6858.02	€ -367.02	−5.07 %
Pump electricity cost	€ 0.23	€ 0.55	€ 0.32	139.13 %
Total costs	€ 7225.27	€ 6858.57	€ -366.70	−5.07 %
S.2 Residential, static prices				
Stored in network at end	20.83 MW h	8.06 MW h	−12.77 MW h	−61.26 %
Losses	176.51 MW h	168.21 MW h	−8.30 MW h	−4.70 %
Produced heat	541.38 MW h	520.30 MW h	−21.08 MW h	−3.89 %
Pump electricity usage	8.01 kW h	38.87 kW h	30.86 kW h	385.27 %
Heat cost	€ 12933.36	€ 12429.83	€ -503.53	−3.89 %
Pump electricty cost	€ 0.72	€ 3.50	€ 2.78	386.11 %
Total costs	€ 12934.08	€ 12433.33	€ -500.75	−3.87 %
S.3 Residential, well insulated				
Stored in network at end	6.30 MW h	0.28 MW h	−6.02 MW h	−95.55 %
Losses	154.72 MW h	153.82 MW h	−0.90 MW h	−0.58 %
Produced heat	309.41 MW h	302.50 MW h	−6.91 MW h	−2.23 %
Pump electricity usage	4.18 kW h	2.00 kW h	−2.18 kW h	−52.15 %
Heat cost	€ 4061.67	€ 3871.55	€ -190.12	−4.68 %
Pump electricity cost	€ 0.12	€ 0.06	€ -0.06	−50 %
Total costs	€ 4061.79	€ 3871.61	€ -190.18	−4.68 %
S.4 Residential, multiple producers				
Stored in network at end	26.10 MW h	5.35 MW h	−20.75 MW h	−79.50 %
Losses	177.07 MW h	169.71 MW h	−7.35 MW h	−4.15 %
Produced heat	891.24 MW h	863.13 MW h	−28.11 MW h	−3.15 %
Pump electricity usage	2.23 kW h	12.62 kW h	−10.39 kW h	465.91 %
Heat cost	€ 11945.40	€ 11360.44	€ -584.61	−4.89 %
Pump electricity cost	€ 0.06	€ 0.41	€ 0.35	583.33 %
Total costs	€ 11945.46	€ 11360.85	€ -584.61	−4.89 %

Table 7.5: Optimised heating schedule savings per scenario with respect to heating curve heating schedule

electricity usage in the network.

Compared to the heating curve based schedule, the optimised schedule increases supply temperatures at peak demand times. This is surprising, as pumping costs are several orders of magnitude smaller than heat production costs, thus it should be preferred to have mass flow as high as possible. From the cost and network analysis it becomes clear that the optimisation raises temperatures in the entire network right before a demand peak, because these peaks almost coincide with price peaks. Then, before the price peaks, the optimisation lowers the temperatures as much as possible. Because the return temperature is relatively high and the supply temperature is lowered as much as possible, ΔT becomes quite small. This apparently reduces costs more effectively than maintaining a high mass flow through the network.

For this instance, the optimisation was able to reduce losses by 2.02 % and overall cost by 5.07 % by increasing mass flows, and scheduling heat production at cheaper times.



(a) Heating curve based heating schedule

(b) Optimised heating schedule

Figure 7.19: Residential scenario S.1 heating schedules

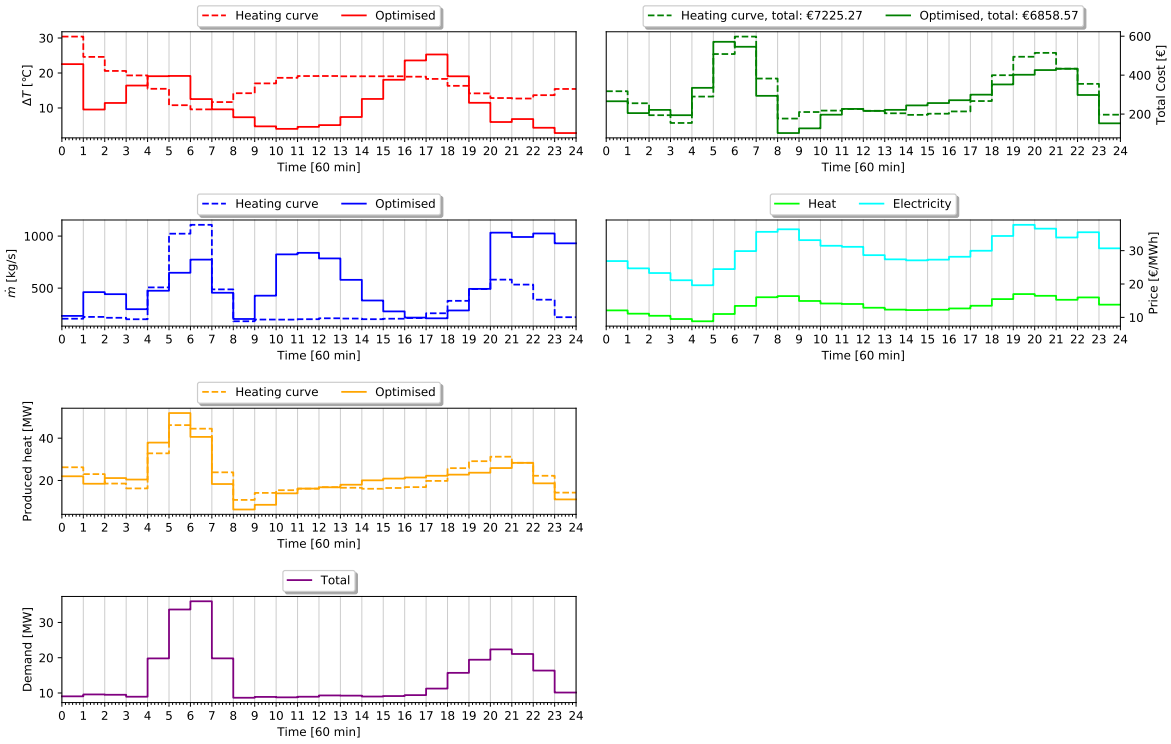


Figure 7.20: Residential scenario S.1 heating schedule cost examination from the perspective of the producer

7.7.2. Residential, Static Pricing

As this instance is the same as scenario S.1, but with static prices, the heating curve is exactly the same as for that scenario, i.e. a constant temperature of 92 °C.

The optimised schedule for this instance can be seen in Figure 7.21 and reduces operating costs by

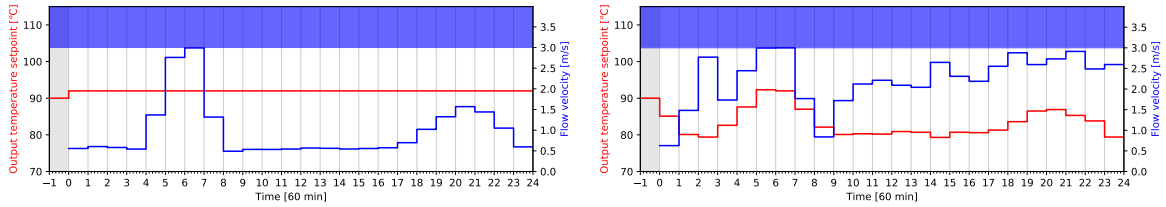
3.87 %.

This schedule does indeed attempt to maximise mass flows through the network and lower the supply temperature as much as possible to reduce losses. This is also apparent from the amount of energy used to pump the water through the DHN, as this is increased by 386.11 %.

As can be seen in Table 7.5, the losses are reduced by 4.70 %. This is more than the loss reduction for **S.1**. This is due to the fact that in that instance the optimisation was able to make use of the network storage effect, as it had dynamic prices. It was possible to decide to accept more losses to save on higher production at expensive times.

For this instance, the optimised schedule indeed lowers temperatures as much as possible to maximise mass flows through the network. After examining the solution further, it became clear that lowering the temperatures even further is not possible, as this would violate the minimum consumer supply temperature constraint for the customer that is farthest from the producer.

For this instance, the optimisation was able to reduce losses by 4.70 % and costs by 3.87 % by increasing mass flows through the network.



(a) Heating curve based heating schedule

(b) Optimised heating schedule

Figure 7.21: Residential scenario **S.2** heating schedules

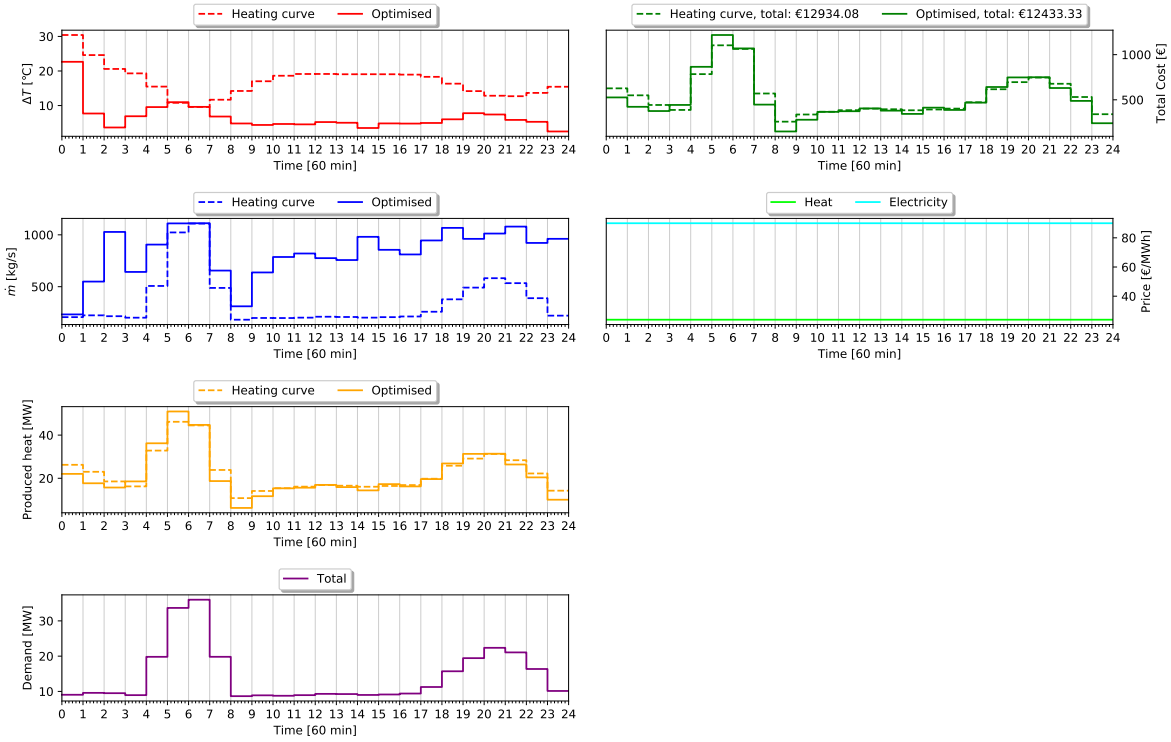


Figure 7.22: Residential scenario **S.2** heating schedule cost examination from the perspective of the producer

7.7.3. Residential, Well Insulated

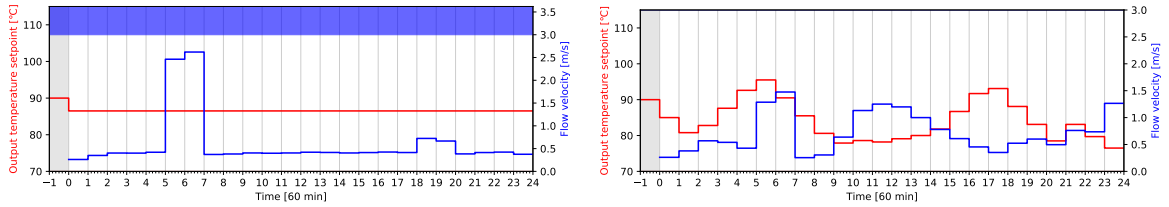
The heating curve for the well insulated residential instance consists of a constant temperature of 86.5 °C.

The optimised heating schedule has about the same shape as the optimised heating schedule for S.1, which is to be expected as the prices and the topology of the network are identical. What is surprising is the amount of loss reduction made. It was expected this would be much higher than the 0.58 % that was achieved. After further investigating why this was the case, it became clear that the minimum consumer supply temperature constraint prevents the temperature from being lowered further. This makes it impossible to further reduce losses without breaking any constraints.

It is interesting to note that the mass flows are apparently lower in the optimised schedule compared to the heating curve schedule. As the pumping costs are 4 orders of magnitude smaller than the heating costs, this does not result in significant savings on the overall costs.

As the prices in this schedule are also dynamic, the optimisation attempts to prevent producing heat at peak times just like it does for S.1.

The optimisation is able to reduce losses by 0.58 % and costs by 4.68 % for this instance by



(a) Heating curve based heating schedule

(b) Optimised heating schedule

Figure 7.23: Residential scenario S.3 heating schedules

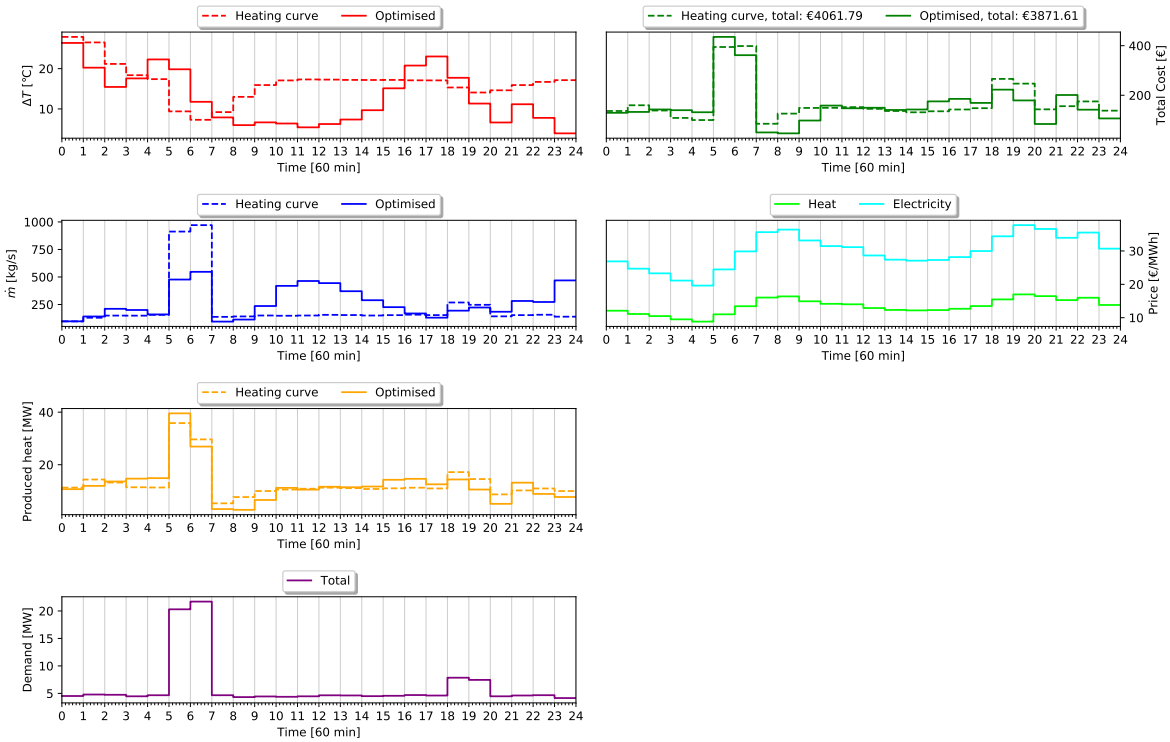


Figure 7.24: Residential scenario S.3 heating schedule cost examination from the perspective of the producer

7.7.4. Residential, Multiple Producers

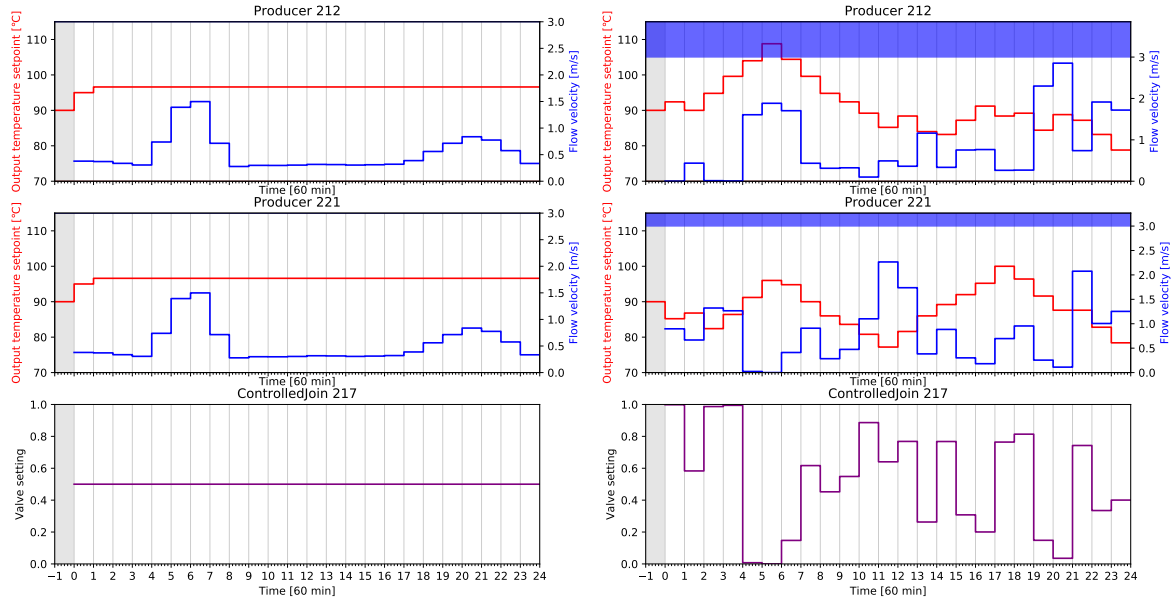
The heating curve for multiple producers is generated by settings the valve settings equal to 0.5 for all time slots and then using the same procedure as used for a single producer. In Figure 7.25 the heating curve and optimised schedule is shown.

As can be seen, the valve setting is very unstable. In this case this does not make sense as both producers are very close to each other and they have the same production costs. It would have made more sense to do the same as the heating curve, assigning the same amount of mass flow to both producers, as this would allow lower temperatures in the network, leading to lower losses.

But, when considering the results of the optimisation, it is surprising that the optimisation was able to reduce losses by 4.15 % as this was expected to be around the same as for **S.1**. The total amount of produced heat has been reduced by 3.15 %, while the total heating cost has been reduced by 4.68 %. This means the optimisation has been able to make use of the network storage effect to produce more at cheaper time, while saving on production costs at expensive times.

From this, it can also be concluded that it is possible to reduce the total losses further, although that might result in more expensive operations. In Figure 7.26, the cost analysis for every producer is shown. It is interesting that the optimisation has assigned much more heat production to one producer than the other. Why this happens should be investigated more, but it is likely this is due to the algorithms getting stuck in a local optimum.

The optimisation is able to reduce operating costs by 4.68 % by making use of the network storage effect and reducing losses.



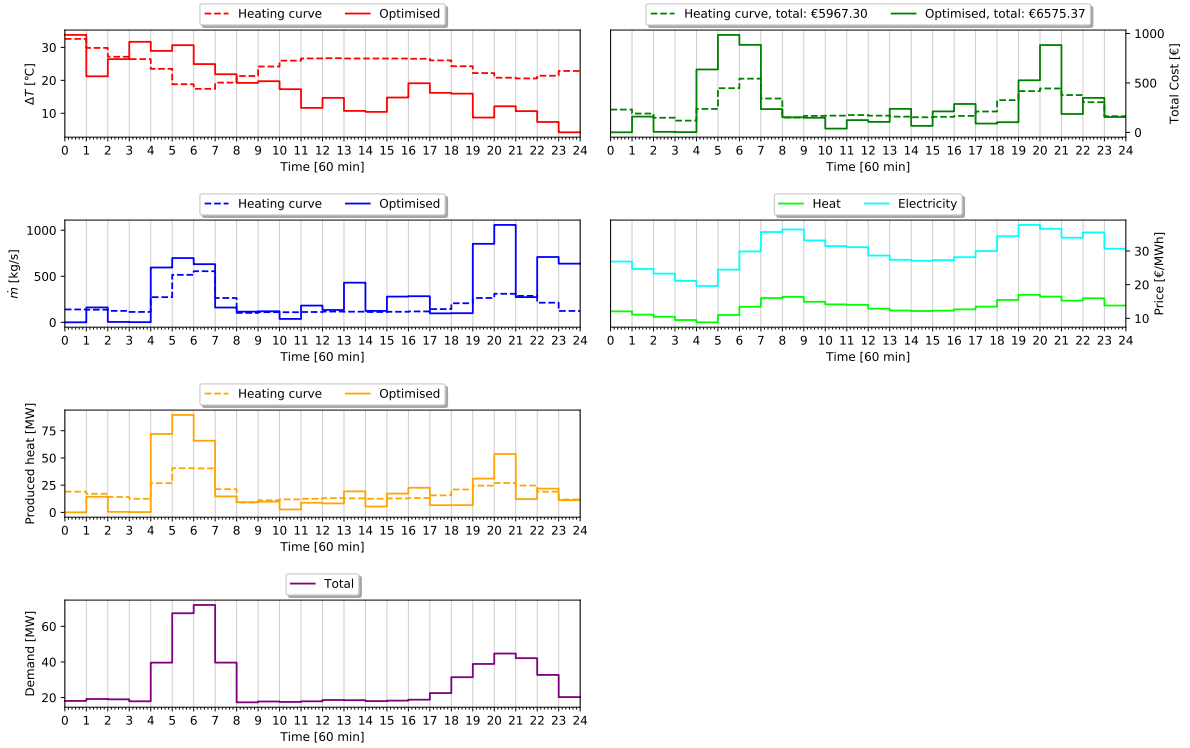
(a) Heating curve based heating schedule

(b) Optimised heating schedule

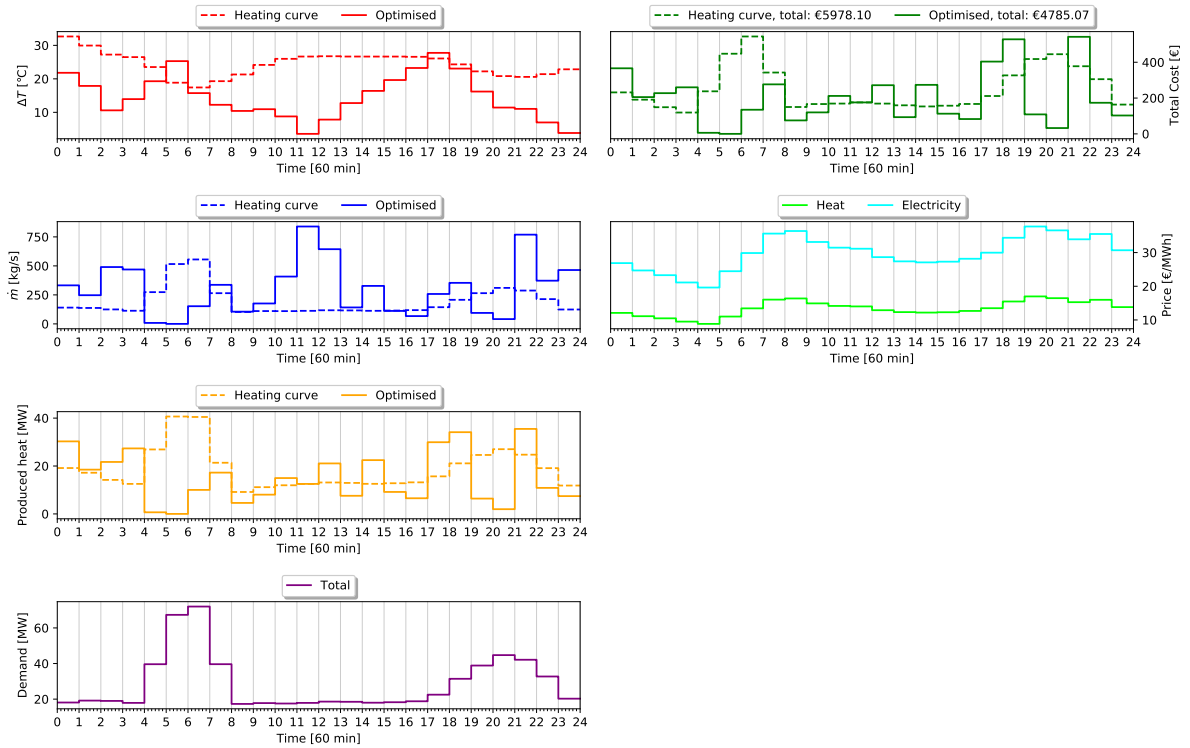
Figure 7.25: Residential, multiple producers scenario **S.4** heating schedules

7.7.5. Conclusions

The question **RQ 2** in the introduction questioned how much can be saved by optimising DHS operations. From the results in this section, it becomes clear that the optimisations are able to make effective use of the network storage effect to reduce costs and that they are able to reduce losses. Depending on the scenario, the optimisations are able to reduce operational costs by 3 % to 5 %. Since operational costs are often quite large, even this small improvement can lead to significant cost reductions.



(a) Cost factors producer 212



(b) Cost factors producer 221

Figure 7.26: Residential, multiple producers scenario **S.4** heating schedule cost examination from the perspective of the producers

8

Discussion

This final chapter discusses the research and its conclusions. In Section 8.1 conclusions are drawn from all the results in this report, answering the research questions posed in Section 1. In Section 8.2 several possible future approaches and DHS types are discussed.

8.1. Conclusions

To conclude this research, the main conclusions are given, answering the research questions posed in Chapter 1.

8.1.1. Best Metaheuristic Optimisation Approaches

The first research question posed in the introduction is **RQ 1**: *What metaheuristic optimisation approaches work well for solving the DHS optimisation problem?*. From the results in Section 7.5 it becomes clear that the approaches specifically built for real-valued problems perform best in terms of optimality of the solutions found.

The genetic algorithm is based on concepts behind combinatorial optimisation problems and does not work very well, although it is able to come close to the other algorithms for the simplest instances. From the results of using the fade over operation instead of crossover, it becomes clear that incorporating some domain knowledge can lead to better performance.

The SaDE and CMA-ES algorithms both are very capable of finding good solutions although both algorithms do have trade-offs. SaDE is able to find very optimal solutions, coming within 2.9 % of the lower bound for certain instances, but this algorithm does converge quite slowly. CMA-ES on the other hand converges more quickly, but still not as fast as expected. In terms of optimality, CMA-ES comes very close to the performance of SaDE.

Prerequisites

Some things are needed before metaheuristics can be to optimise the DHS problem. From **RQ 1.1**: *What is needed to make the use of metaheuristics feasible?*

First, as described in Section 3.3.2, an artificial constraint is needed that requires a certain amount of energy at the end of the optimisation horizon. This is not necessarily exclusive to metaheuristic optimisation techniques, but without this constrain optimisations will make savings at the end of the optimisation horizon that are not useful and might even harm actual cost savings. In addition to this, the constraint allows the determination of a lower bound on the operational costs.

Second, access to a simulator (or equivalent) is needed to determine what the effects of making certain decision are. This includes information on the actual monetary cost of a heating schedule, but also physical information like mass flow limits that are exceeded by a schedule or consumer supply temperatures that are lower than allowed.

8.1.2. Operational Cost Reduction

The second research question **RQ 2** in the introduction: *How much improvement does an optimised heating schedule offer with respect to the existing operational methods?* From the results in Section 7.7 it becomes clear that depending on the instance, the optimisation of the operations can lead to a cost reduction of 3 % to 5 %.

Due to the large costs of operating DHSs, even this small percentage leads to significant cost reduction. These cost reductions are thanks to the optimisation's ability to make use of the network storage effect, moving heat production to cheaper times and to reduce overall heat production, by reducing heat losses in the network.

Lower Bound on Operational Costs

One of the larger contributions of this work is the method to determine a lower bound on the operational costs. A lower bound for the problem helps to get a feeling for whether an algorithm has reached the optimal solution and makes it easier to compare the quality of solutions. This section answers **RQ 2.1**: *Is a lower bound on the production costs available (in literature) or is it possible to determine a useful one?*

The first approach to the question was to look for a lower bound in literature. However, after searching for this, none have been found. Thus, a novel method has been created to determine this lower bound, as described in 3.7.

From the results in Chapter 7 it has become clear that the method can get very close to the real lower bound for some instances. For **S.2**, the optimised values were within 3 % of the lower bound. This means the lower bound is very tight for instances with static prices. For dynamically priced instances, the lower bound is less tight, which is to be expected, but it is still within a reasonable range.

8.1.3. Context

The final research question compares the new approaches with existing optimisation approaches found in literature. **RQ 3**: *How do the approaches compare to the existing mathematical optimisation approaches found in literature?*

The main difference between the approaches used in this work and the works discussed in Chapter 4 is the optimisation methods used. Most of the existing work uses mathematical optimisation in combination with a simulator to optimise the operations. This work uses metaheuristics in combination with a simulator. This new approach should be better able to deal with the non-convexity of the problem, as mathematical optimisation is often not able to deal with this.

The existing work that does use metaheuristics based optimisation, have simplified the problem strongly by ignoring key characteristics like the network storage effect and the non-linear losses. For smaller networks this should not be a large problem, but when the networks grow to span several kilometres, these simplifications will start to add up. The results achieved in this work are comparable to existing work in terms of savings [10, 21].

8.2. Future Work

This research focussed mainly on whether it is feasible to use metaheuristics to optimise district heating operations. As has been concluded, it is feasible to do this, but there are some limiting factors and/or possibly better alternatives. In addition to this, some ideas on what is needed to apply this algorithm in practice are presented.

8.2.1. Fitness Function

The fitness function is a vital component of the algorithms as it is the only indication the algorithms have of how good the solutions they propose are.

Simulation Replacement

One of the largest limiting factors is the simulator that backs the fitness function. Even with several dozen of consumers, the simulation slows down to the point where it becomes too slow to use the

optimisation in an MPC setting. To solve this problem, simulator performance should be increased, or an alternative for online simulation should be found.

A possible alternative for online simulation would be to replace the simulation with a machine learning model that can provide the information now provided by the simulation like maximum edge velocity violation and production costs.

In [11] a neural network model was used to replace an expensive traffic simulation in the fitness function of a genetic algorithm. This genetic algorithm was used to optimise traffic lights in the city of Warsaw to minimise total waiting time. A similar method could be used to replace a DHS simulation.

A long-short term memory (LSTM) neural network seems the most promising model as these models maintain an internal state and accept a series of values as inputs. The internal state allows the model to remember how much heat is stored in which locations in the network and the input series in this case would be the valve and temperature settings.

Optimising the Fitness Function

As described in Section 3.5, the fitness function is constructed by determining the amount of constraint violation and multiplying this with penalty factors. These factors have been chosen manually, but it might be possible to automate this. Doing this might improve the convergence speed of the algorithms discussed, as the fitness landscape shape determines how quickly the algorithms can move through it.

8.2.2. Alternative Optimisation Approaches

In addition to metaheuristics and mathematical optimisation, several other optimisation techniques might be promising to explore for this problem. Two possible alternatives are surrogate modelling and reinforcement learning.

Surrogate Modelling

With surrogate modelling, a model of the fitness function is built online and this model is then used to find an optimal solution. The algorithm takes an initial measurement of the fitness function and fits a function to. The algorithm then iteratively takes new measurements of the fitness function and fits a function to all measurement points. Every new measurement is taken at the minimum of the current fitted function. Finding this minimum can be problematic if the function is non-convex, but this might be circumvented by using simulated annealing.

This approach could work, but the high dimensionality of the DHS problem might be problematic as surrogate modelling can typically not handle more than a few dozen variables. Recent research has devised a method that would expand this to several thousand variables. [22]

Reinforcement Learning

Reinforcement learning builds a model of the system under optimisation and creates a policy that can then be used to operate the system. A policy is a mapping of states to actions that indicates what action should be taken given the current system state to optimise the reward.

An interesting application of reinforcement learning is learning a computer program to play Atari games [24]. The reinforcement learning model was created using a neural network that was trained with a variant of Q-learning. Q-learning is a method that can build a Q function that indicates how much reward can be gained by performing action a while currently being in state s. The input of the neural network are the raw pixel values that are displayed on screen and the output was a value function that can estimate the future rewards of performing an action.

The system was able to outperform existing methods in all 6 six games and even outperform human experts in three of the games.

In the case of the DHS problem the system state consists of the heat stored in the network and the expected consumer demand. The actions could be defined as increasing the production temperatures of producers and operating the pumps in the network. A possible problem with approach could be the high dimensionality in the number of actions. For the Atari games there are 18 possible actions, but the number of actions for a DHS are much higher, especially when the number of producers and valves increases.

8.2.3. Extensions

The methods used in this researched were mostly quite basic algorithms that were not specialised on the DHS problem. In addition to this, certain assumptions have been made, which might have been made too broadly to cover problems that might occur had they been made too narrowly.

Binary Decision Variables for Buffer Storage

The current problem instances have no buffers in the networks. Buffers are components in the network that can store heat in large water vessels with very little heat loss. For buffers, it can be decided to either fill them or empty them, which is a binary decision variable. The heat stored in a buffer ahead of time allows operators to use fewer peak boilers during peak demand times and thus reduce costs.

However, the real-valued algorithms have not been designed to deal with binary decision variables.

To deal with these kinds of problems it might be interesting to combine a binary genetic algorithm with the real-valued algorithms. The real-valued algorithms would become a sub procedure of the binary genetic algorithm, where the genetic algorithm first decided whether a buffer should be filled or emptied and can then use this decision to optimise the real-valued decision variables. This kind of procedure was already attempted [29] where the binary decision variables were used to decide which production plants to use.

A different kind of buffer is a aquifer where heat is stored in the ground water. Operation of aquifers is on a longer time span than what is done in this research, but it might still present an interesting research direction.

Optimisation Horizon Length

This first assumption is the optimisation horizon length. Depending on the network length and the minimum flow velocities, a certain optimisation horizon is needed to ensure the algorithm makes good decisions close to the present. In this research, the required horizon length was estimated by manually trying several settings and estimating whether the horizon is long enough.

However, the minimum horizon length needed varies depending on the state of the network. If a lot of heat is stored in the network, consumers will need less mass flow to fulfil their demand which means the horizon should be longer as decisions will have a longer effect on the network.

This horizon length could possibly be optimised to reduce the problem dimensionality, although it should not really have a high priority.

8.2.4. Low Temperature District Heating Systems

The DHSs in this research were all high temperature, which means the supply temperatures never go below 70 °C. Other kinds of DHSs also exist in which the supply temperatures are much lower, sometimes as low as 15 °C. [19] The operation of these kinds of systems is much different, as consumers often need additional systems to boost the power received from the DHS to the required temperatures.

Because high temperatures lead to much more losses, requires much more energy and often requires fossil fuels, low temperatures DHSs are more likely to be the system of the future. Research on optimising the operation of these kinds of systems is a more promising idea.

8.2.5. Prerequisites for Deploying the Optimisation

Before the optimisation in the work can be deployed in practice, several missing pieces should be built.

The first and most important component is to build a model of the DHS under optimisation. This consists of a topological model of the grid and other properties like heat resistances and consumer heat exchangers.

The second thing that is needed is some way to predict consumer heat demand and production prices. These predictions are what the optimisation uses to make decisions on when to produce heat and how much should be produced. Demand prediction could be done by using one of the techniques discussed in Section 4.2. Production prices can be predicted using other techniques commonly found in literature, like time series forecasting.

Finally, some way to deploy the solutions to grid operators is needed. The solution found by the optimisation is a set of settings for valves, but it is possible that this is not how grid operators manage the mass flows in the network. Some grid operators only use the valves to completely close one route in the network and use pressure settings to do fine-tuned routing. This would require the valve settings to be translated to pressure settings for pumps in the network, which can be done using pressure calculations.

8.3. Closing

Optimising district heating operations using metaheuristics is feasible and does work, but more research is needed. The biggest challenge is the convergence speed of the algorithms, which is relatively low compared to conventional optimisation techniques. The algorithms used in this research were quite basic and little domain knowledge was used in the algorithms. It is likely that adding more of this domain knowledge to the algorithms will speed up convergence significantly, as this allows algorithms to make more informed decisions.

Despite this, the algorithms in this research were able to make significant savings on operational cost with respect to the current method of operating district heating systems. In particular, the algorithms are able to exploit the network storage effect and do indeed reduce losses to reduce operating costs.

A

Experiment Scenarios

The heat exchanger parameters used in the scenarios are shown in Table A.1.

	S.1, S.2, S.3	S.4
k	55000	55000
q	0.8	0.8
A [m^2]	0.15	0.2
T_{si} [°C]	45	45
T_{so} [°C]	45	45
$\dot{m}_{p_{\max}}$ [kg/s]	30	3000

Table A.1: Heat exchanger parameters used in the scenarios

A.1. Residential (Statically Priced)

The production prices are shown in Table A.2. These prices are based on the average electricity prices in the EU from 2019-06-26 to 2019-07-04. The heat production costs are equal to 0.45 times the electricity cost, assuming a combined heating and power (CHP). This means, when heat production prices are high, the electricity price is high, which would mean generating electricity is more profitable than generating heat.

The average demand is shown in Table A.2 and in Figure A.1.

Time	Costs [€/MWh] Heat	Electricity	Demand [kW] Residential
00:00	12.0938	26.8750	30.1707
01:00	11.1105	24.6900	31.9669
02:00	10.4805	23.2900	31.6624
03:00	9.5062	21.1250	29.8166
04:00	8.8313	19.6250	66.0272
05:00	11.0092	24.4650	112.2481
06:00	13.4392	29.8650	120.0151
07:00	16.0357	35.6350	66.0511
08:00	16.3800	36.4000	28.8446
09:00	14.9152	33.1450	29.6311
10:00	14.1570	31.4600	29.2258
11:00	14.0152	31.1450	29.8077
12:00	12.8902	28.6450	31.0145
13:00	12.3255	27.3900	30.8380
14:00	12.1905	27.0900	29.9946
15:00	12.2850	27.3000	30.4345
16:00	12.6743	28.1650	31.3260
17:00	13.4865	29.9700	37.5071
18:00	15.4778	34.3950	52.3718
19:00	16.9830	37.7400	64.7837
20:00	16.4655	36.5900	74.5452
21:00	15.2730	33.9400	70.2064
22:00	15.9818	35.5150	54.5279
23:00	13.8105	30.6900	33.7633

Table A.2: Dynamic heat production cost, electricity prices, and residential demand

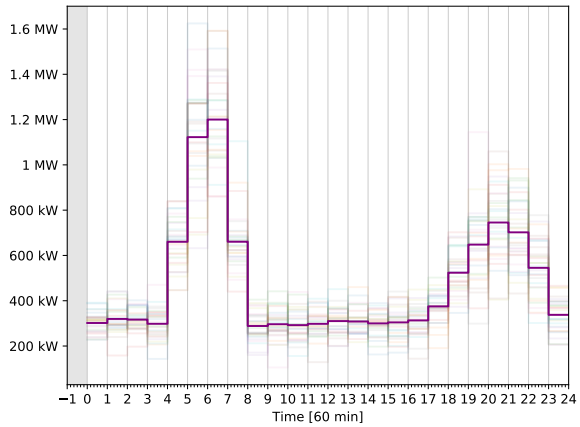


Figure A.1: Average residential demand

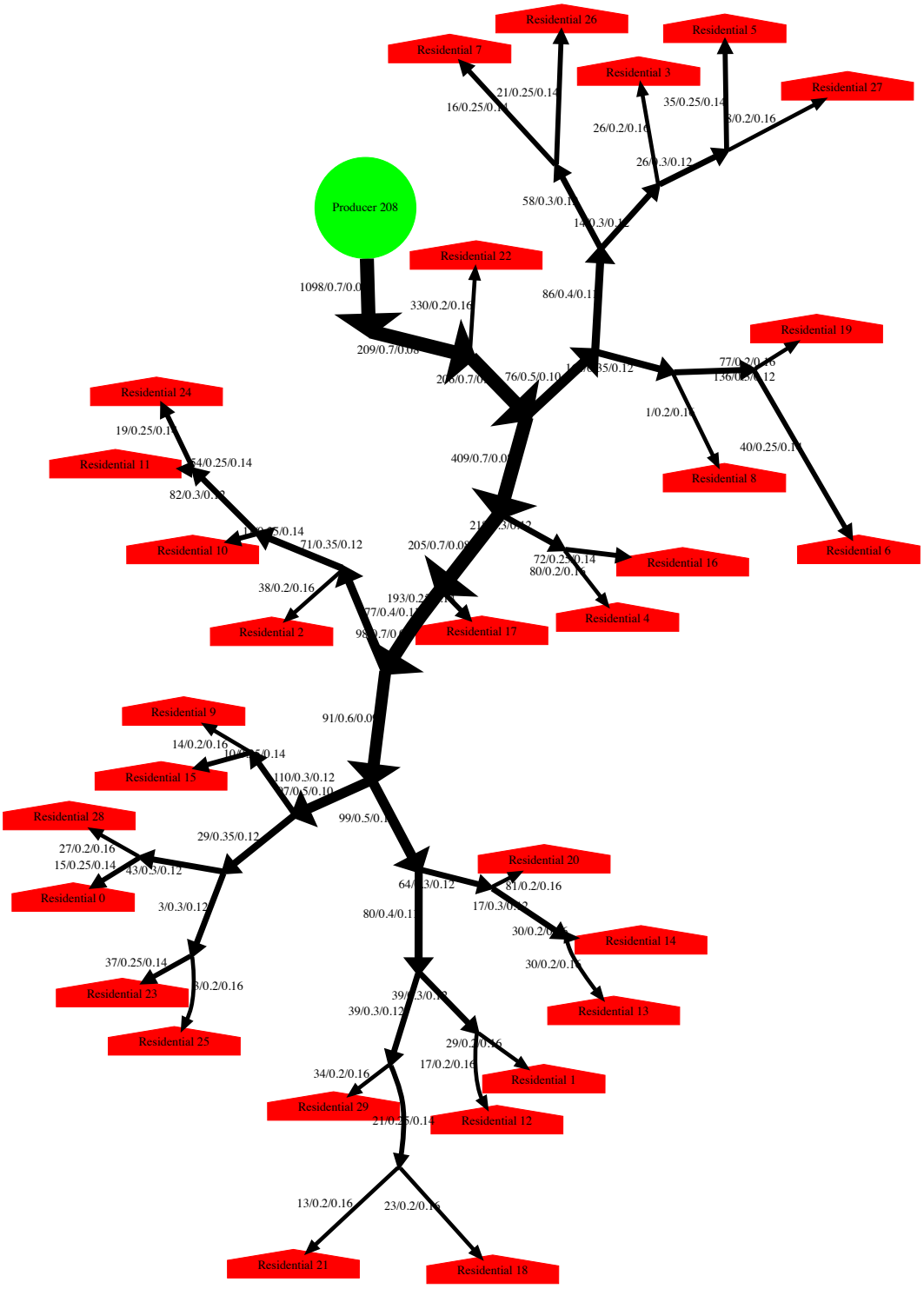


Figure A.2: Residential scenario topology, supply side only. Return side is symmetric. Edge labels: length/diameter in meters

A.2. Residential, Well Insulated

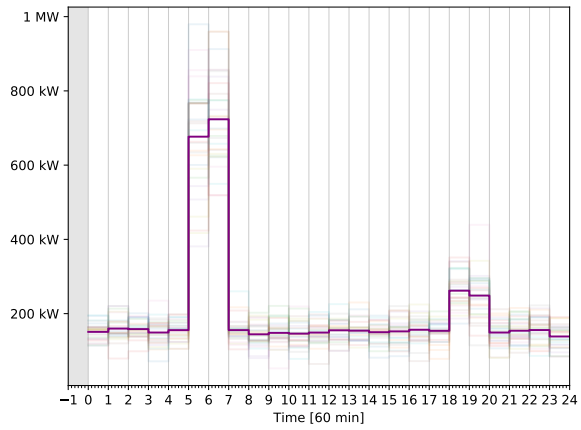


Figure A.3: Average residential well insulated demand

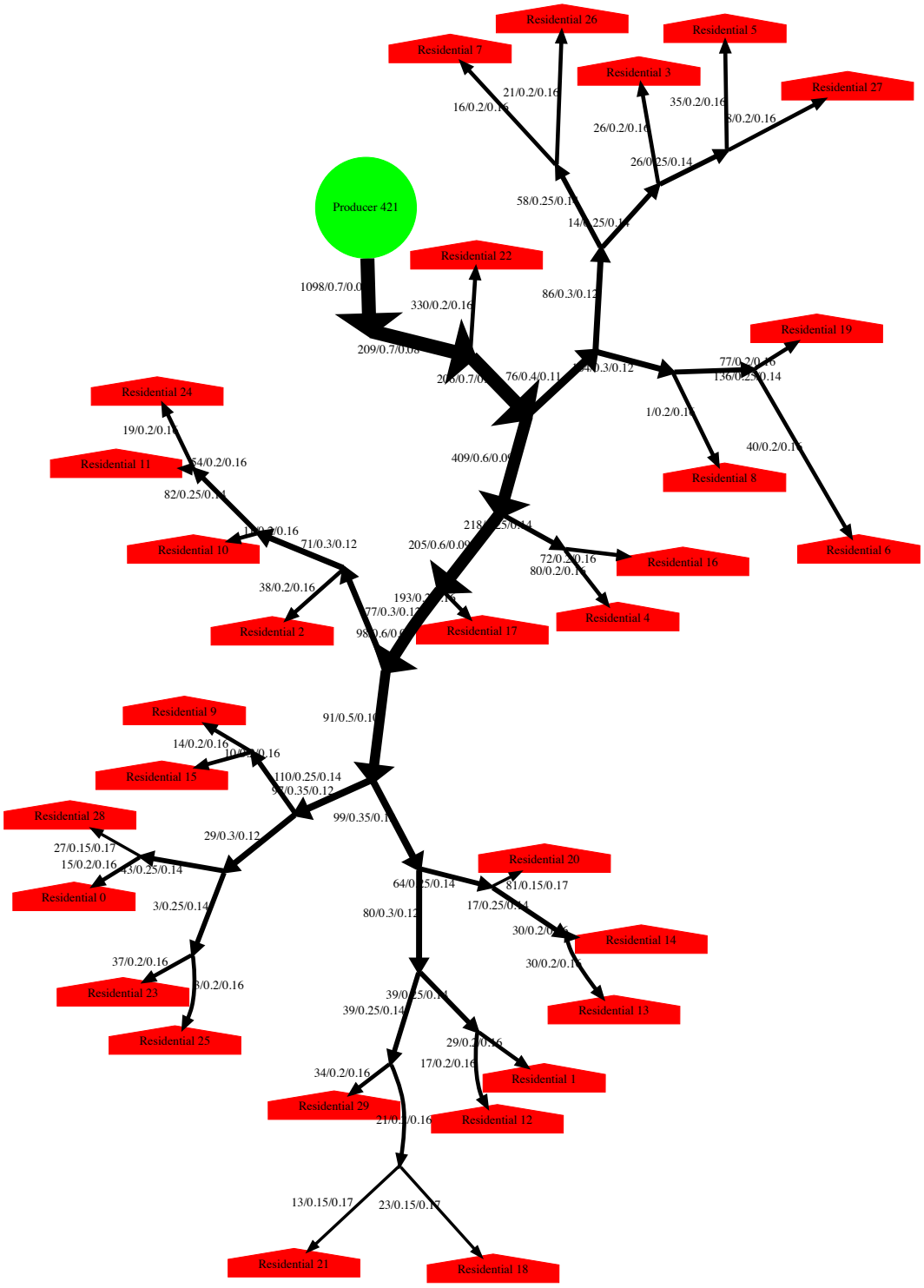


Figure A.4: Well insulated residential scenario topology, supply side only. Return side is symmetric. Edge labels: length/diameter in meters

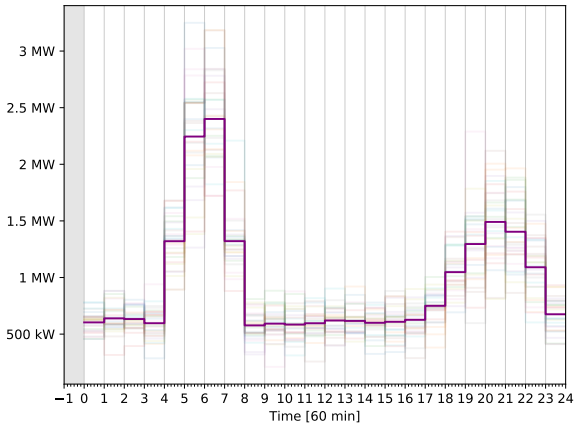


Figure A.5: Average residential multiple producers demand

A.3. Residential, Multiple Producers

The heat exchangers used in this scenario are different from the heat exchangers in **S.1**, as the demand is considerably higher. These parameters can be found in Table A.1.

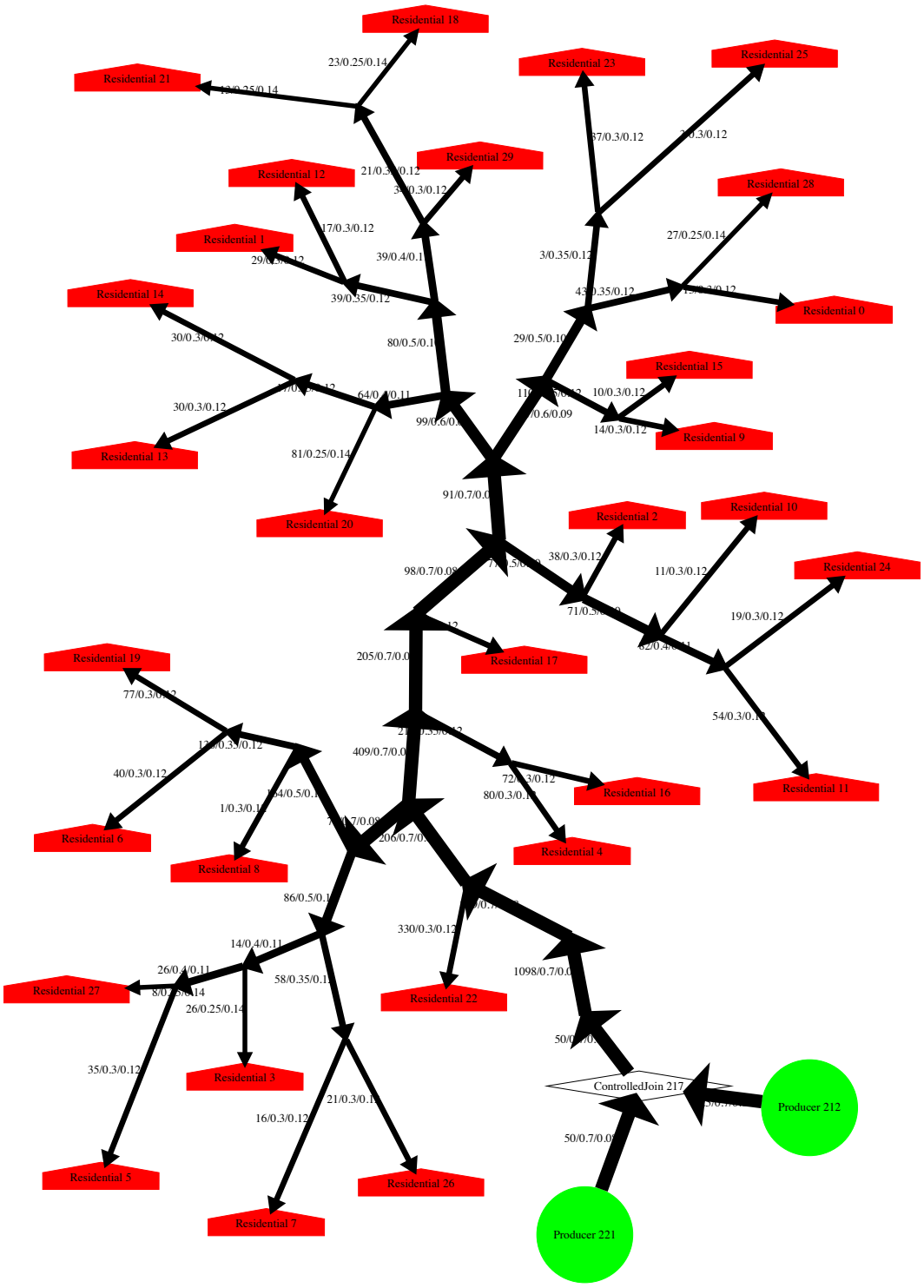


Figure A.6: Residential scenario topology with multiple producers, supply side only. Return side is symmetric. Edge labels: length/diameter/thermal resistance in m/meter/K m/W

B

Full Results

This appendix contains figures that represent all the results. The results are organised by meta-parameter and then by the measured properties.

B.1. Population Size

This section contains the results for the varying population sizes.

B.1.1. Optimality

In Figure B.1 the performance ratio of the algorithms with respect to the lower bound is shown.

B.1.2. Convergence Speed

Figure B.2 shows the number of iterations needed before the algorithms are converged.

B.1.3. First Iteration With a Valid Solution

Figure B.3 shows the number of iterations before the first valid solution has entered the population.

B.1.4. Consistency

In Figure B.4 the distance from the other final solutions for every final solution is shown.

B.2. Variation

As described in Section 6.3, varying the exploration meta-parameters of an algorithm will likely influence how optimal the solutions found are. In this section the influence of varying this parameter is explored. Table 7.2 shows the number of valid solutions found by the algorithms with the given variation parameters.

B.2.1. Optimality

The optimality of the solutions reached by the algorithms are shown in Figures B.5 and B.6.

B.2.2. Convergence Speed

The number of iterations required before termination are shown in Figures B.7 and B.8

B.2.3. Generations Until Valid

The number of iterations until the algorithm finds its first valid solution, is shown in Figures B.9 and B.10.

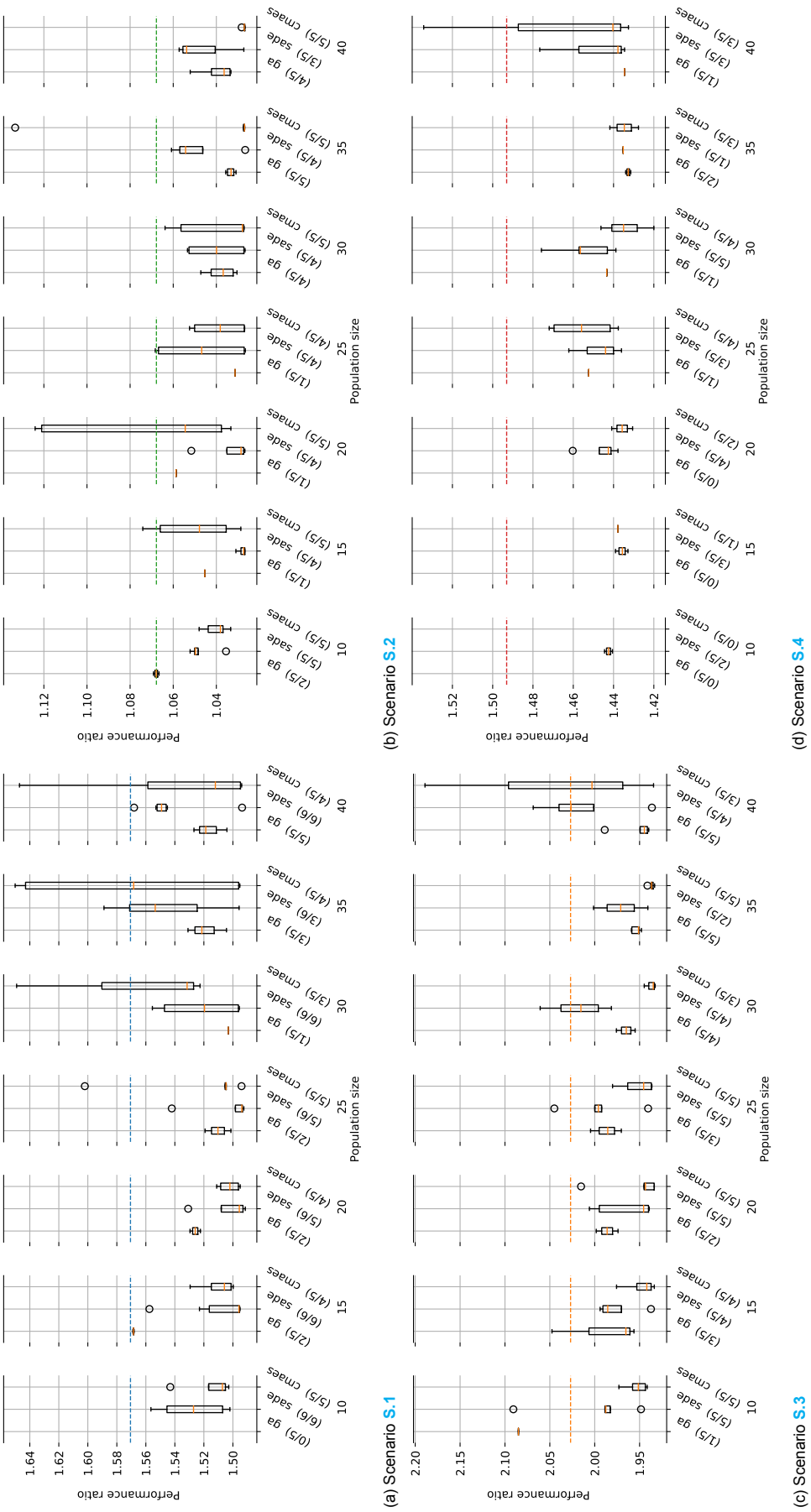
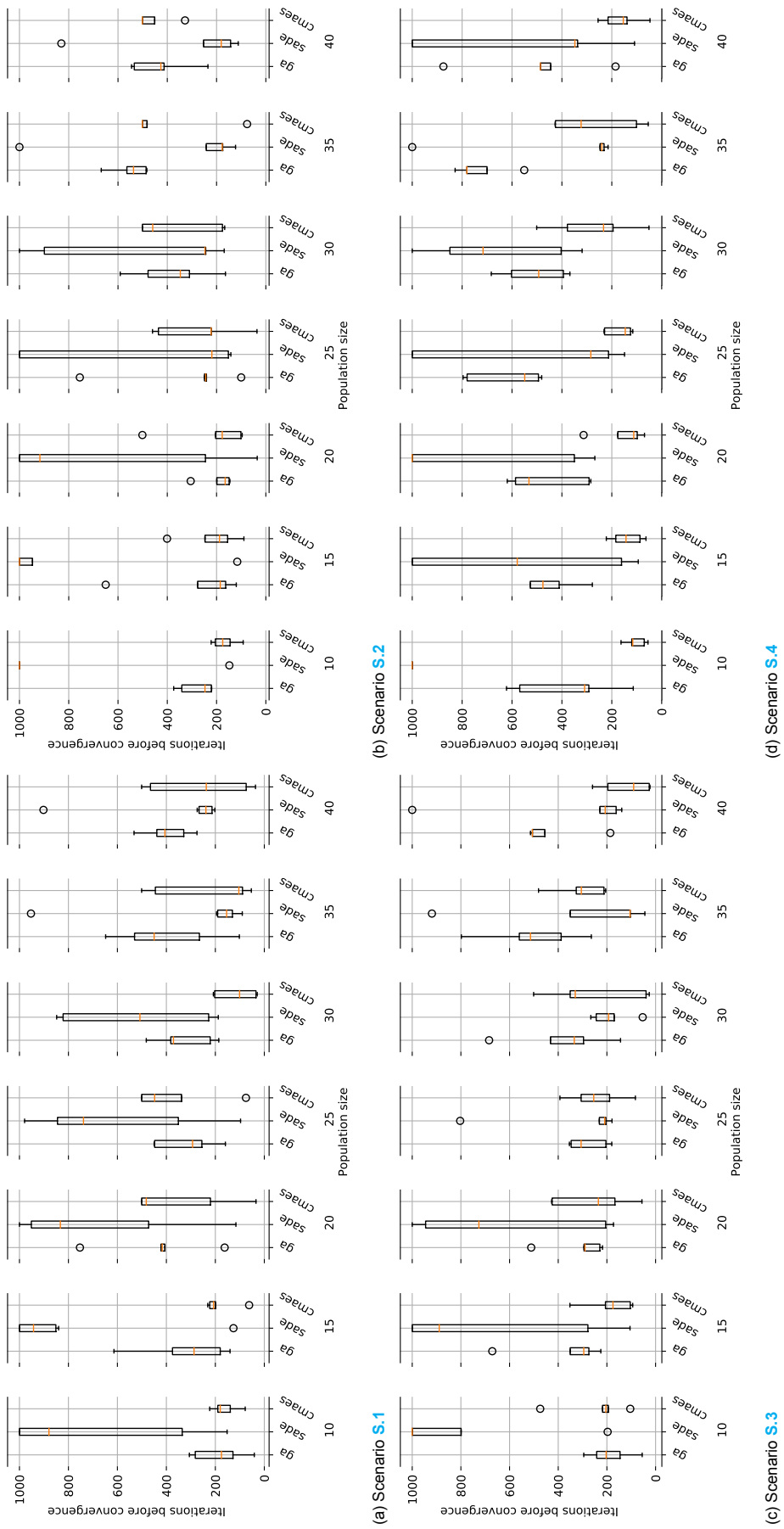


Figure B.1: Performance ratios with increasing population sizes. The number of valid solutions is shown between the braces. The dashed lines are the performance ratios of the heating curve based operations.



(a) Scenario S.1 (b) Scenario S.2 (c) Scenario S.3 (d) Scenario S.4

Figure B.2: Number of iterations before convergence

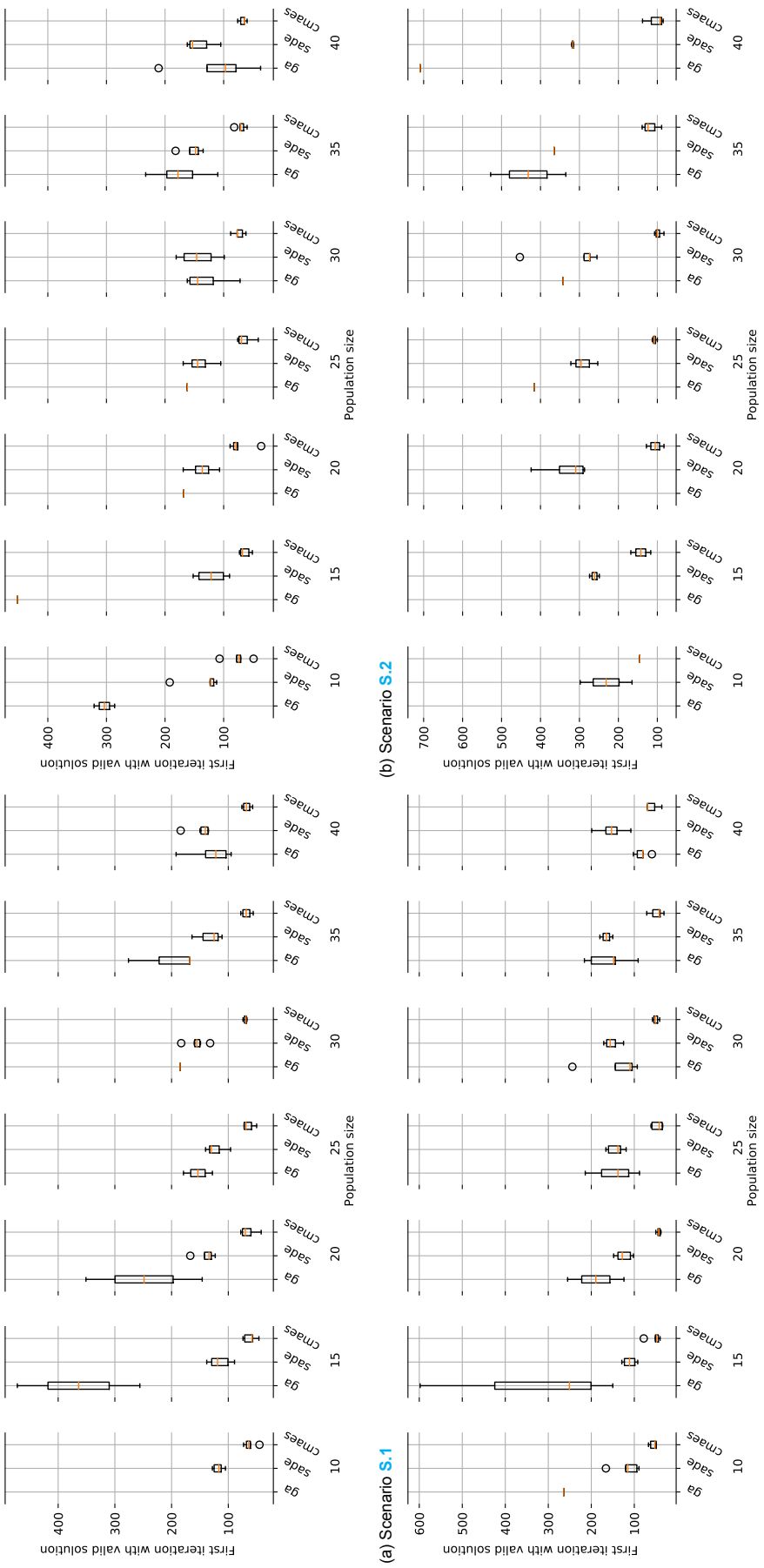


Figure B.3: Number of iterations before a valid solution is found. The number of experiments that found a valid solution is shown in the braces

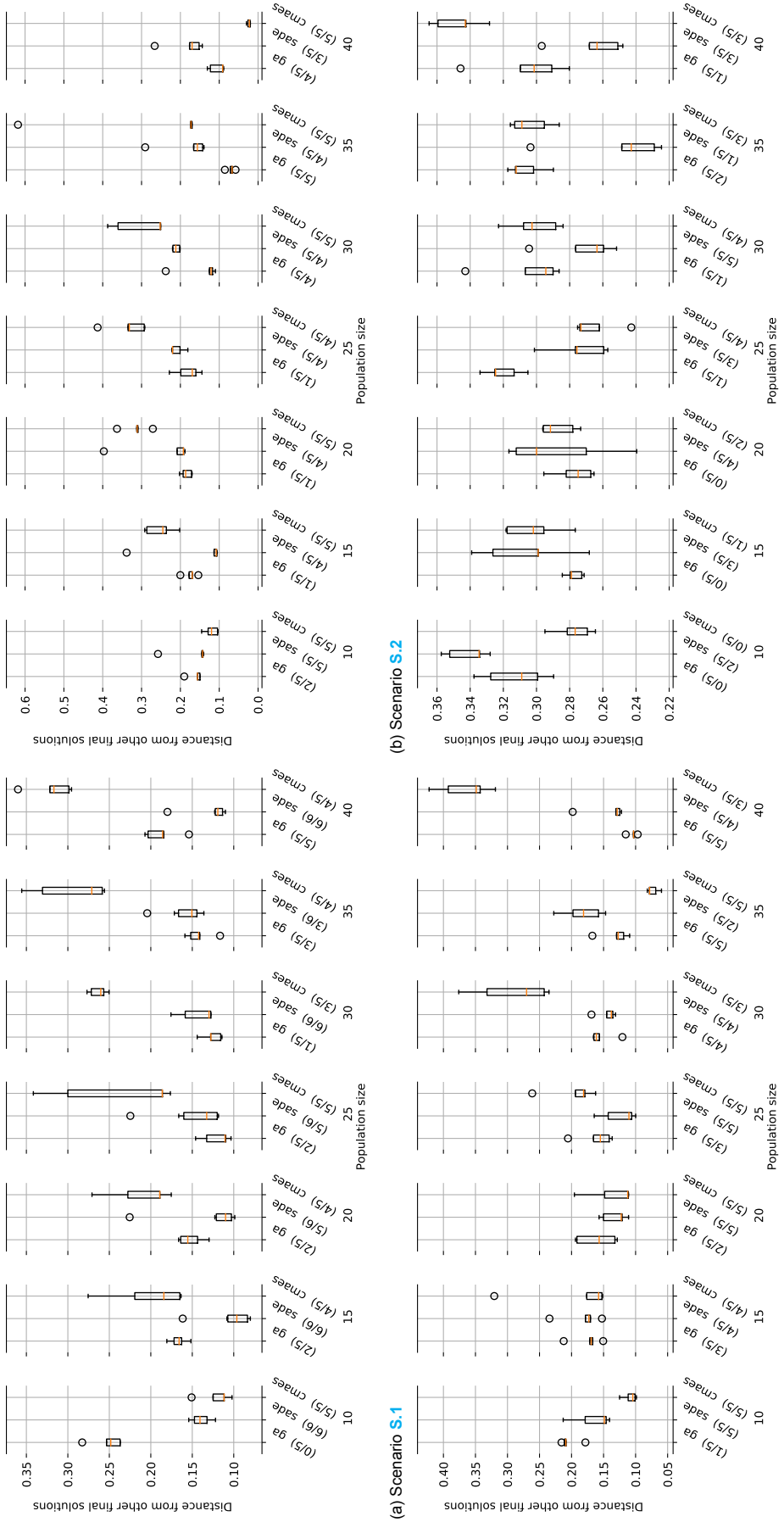
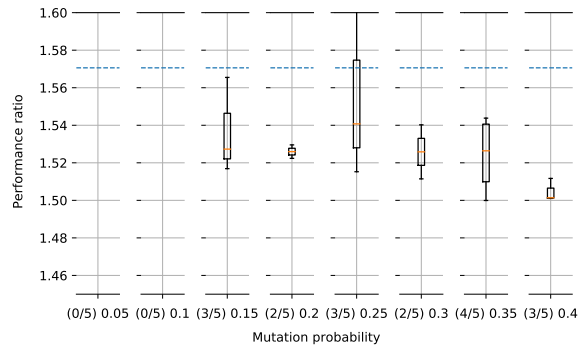
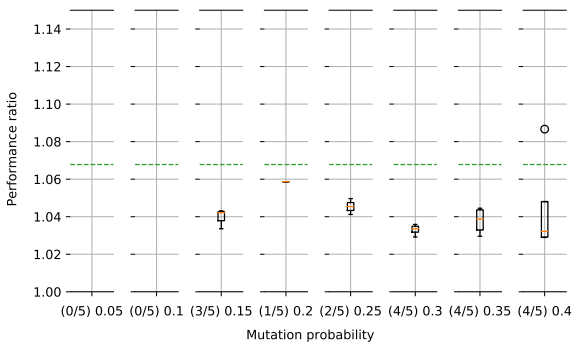


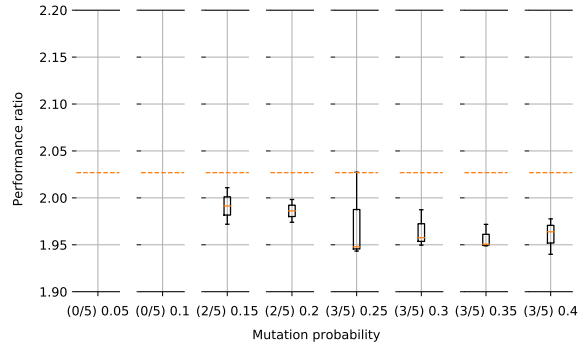
Figure B.4: Final solution distances from other final solutions in the same experiment with varying population size



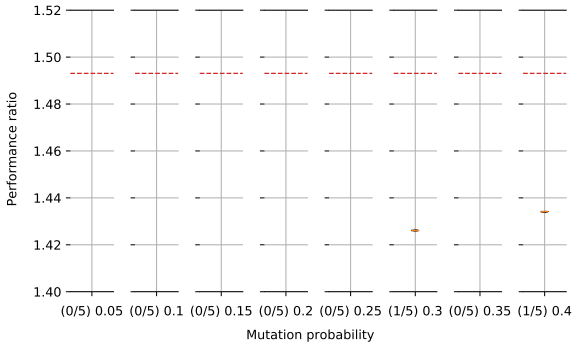
(a) Scenario S.1



(b) Scenario S.2

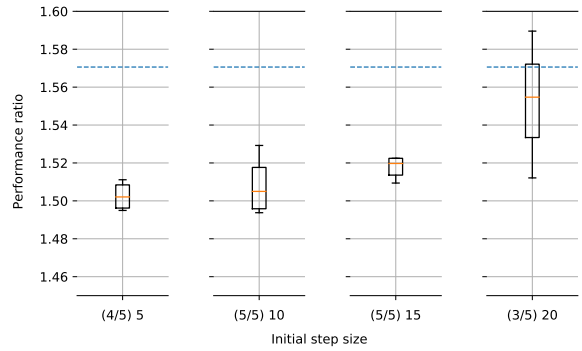


(c) Scenario S.3

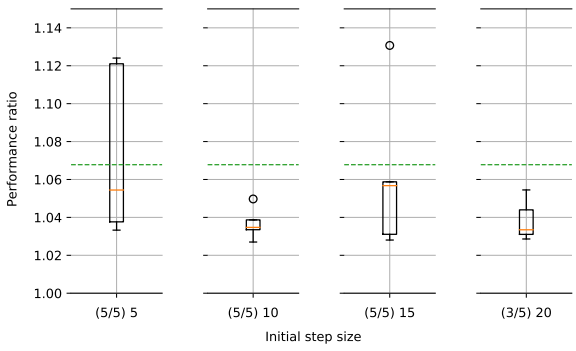


(d) Scenario S.4

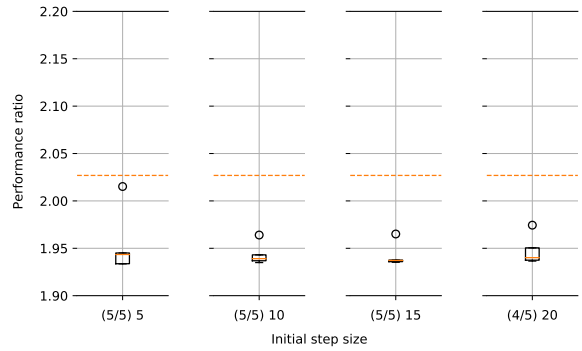
Figure B.5: Performance ratios genetic algorithm with varying mutation probability



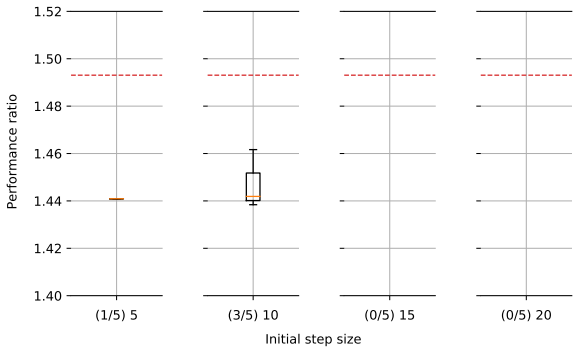
(a) Scenario S.1



(b) Scenario S.2

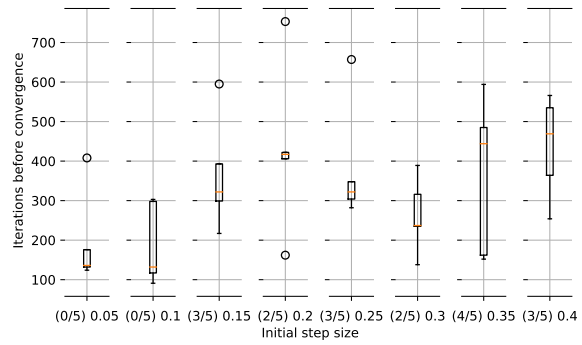


(c) Scenario S.3

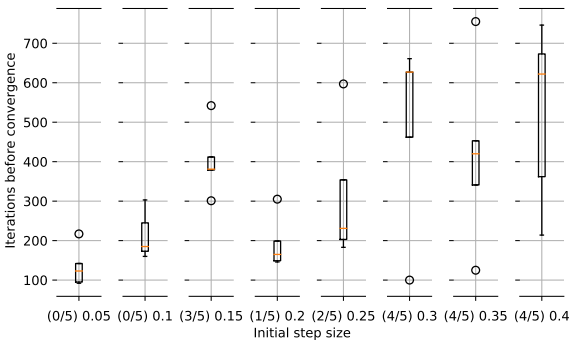


(d) Scenario S.4

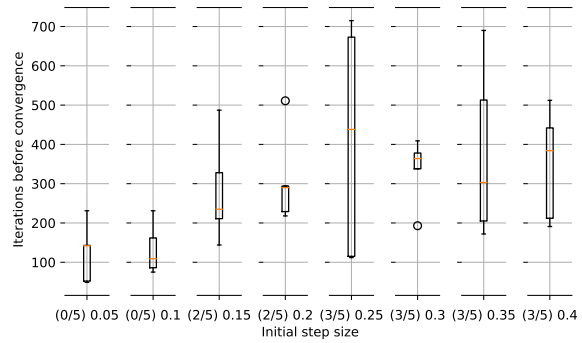
Figure B.6: Performance ratios CMA-ES with varying variation



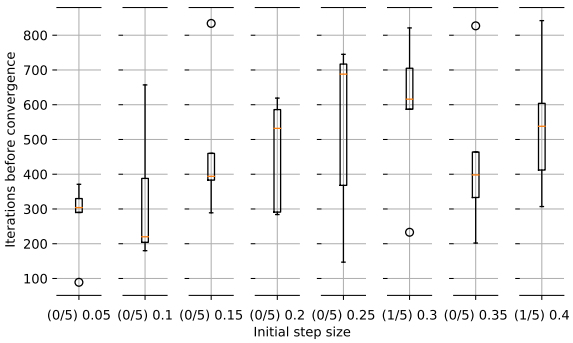
(a) Scenario S.1



(b) Scenario S.2

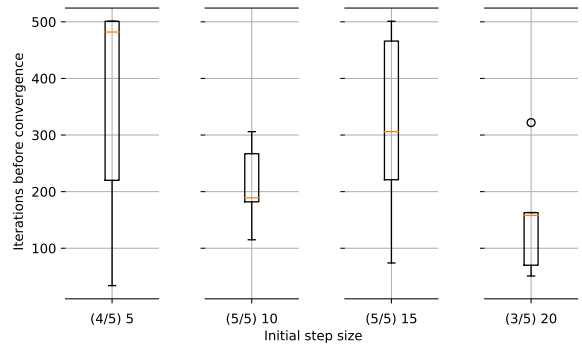


(c) Scenario S.3

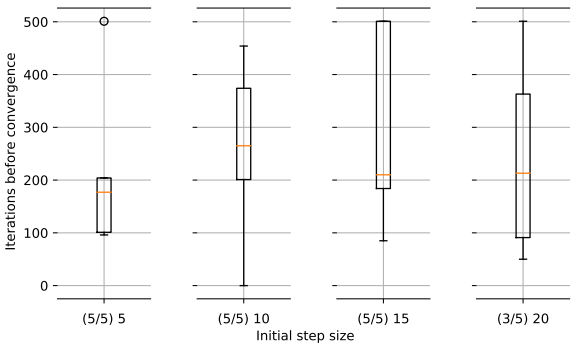


(d) Scenario S.4

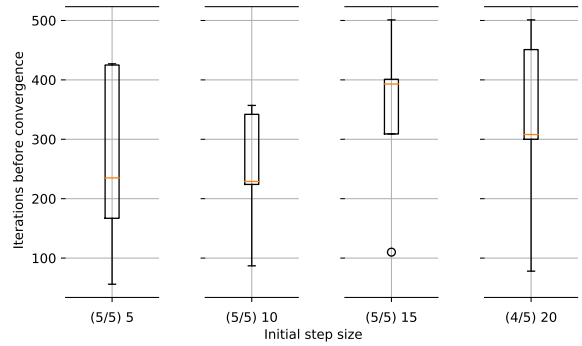
Figure B.7: Number of iterations before convergence genetic algorithm with varying mutation probabilities



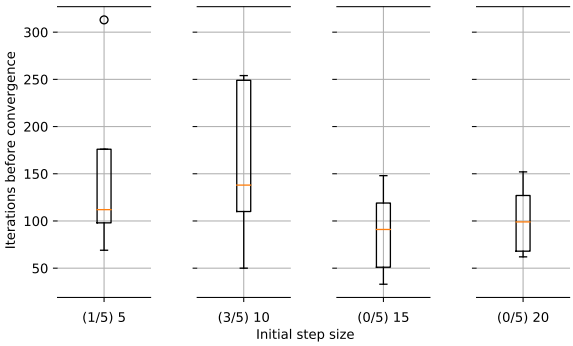
(a) Scenario S.1



(b) Scenario S.2

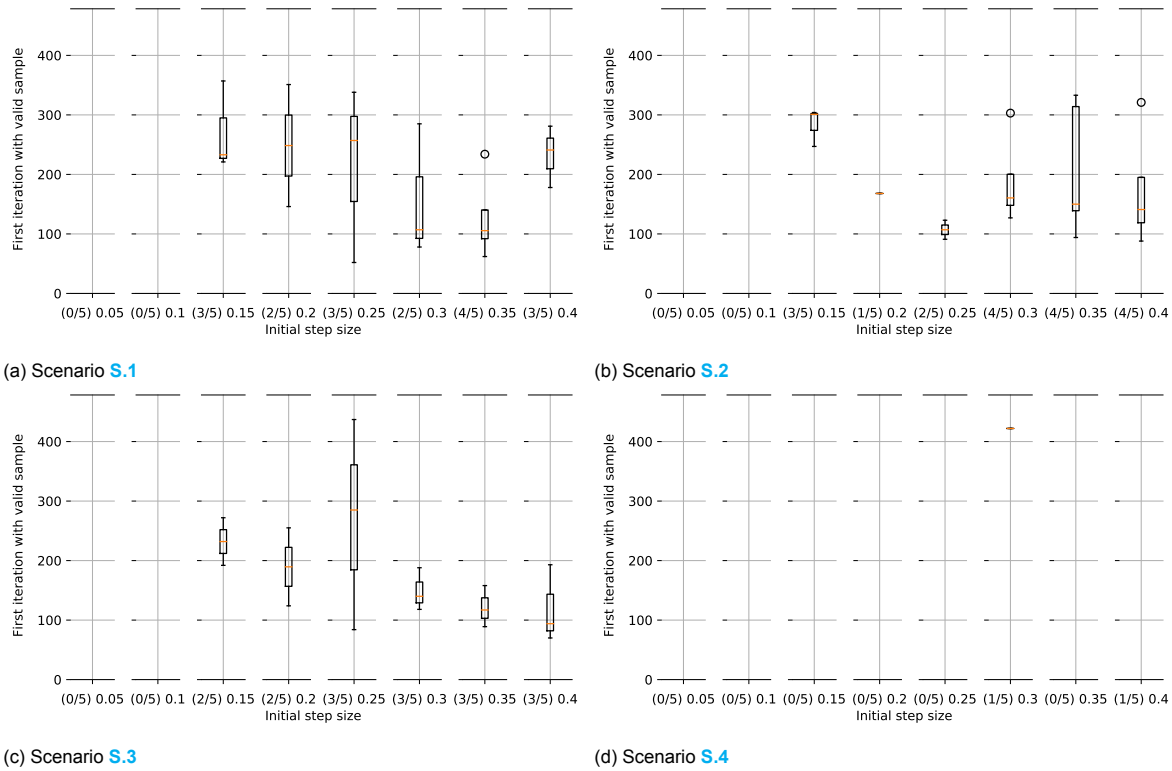


(c) Scenario S.3



(d) Scenario S.4

Figure B.8: Number of iterations before convergence CMA-ES with varying initial step sizes



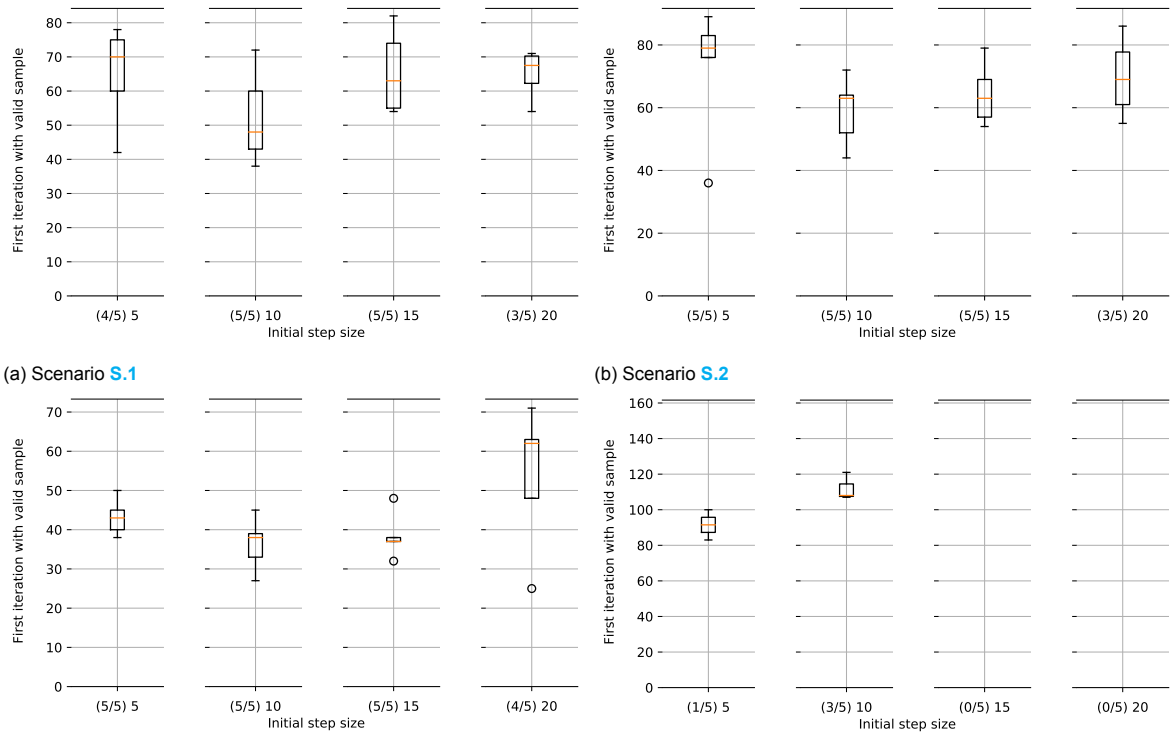
(a) Scenario S.1

(b) Scenario S.2

(c) Scenario S.3

(d) Scenario S.4

Figure B.9: First generation with a valid solution GA with varying mutation probability



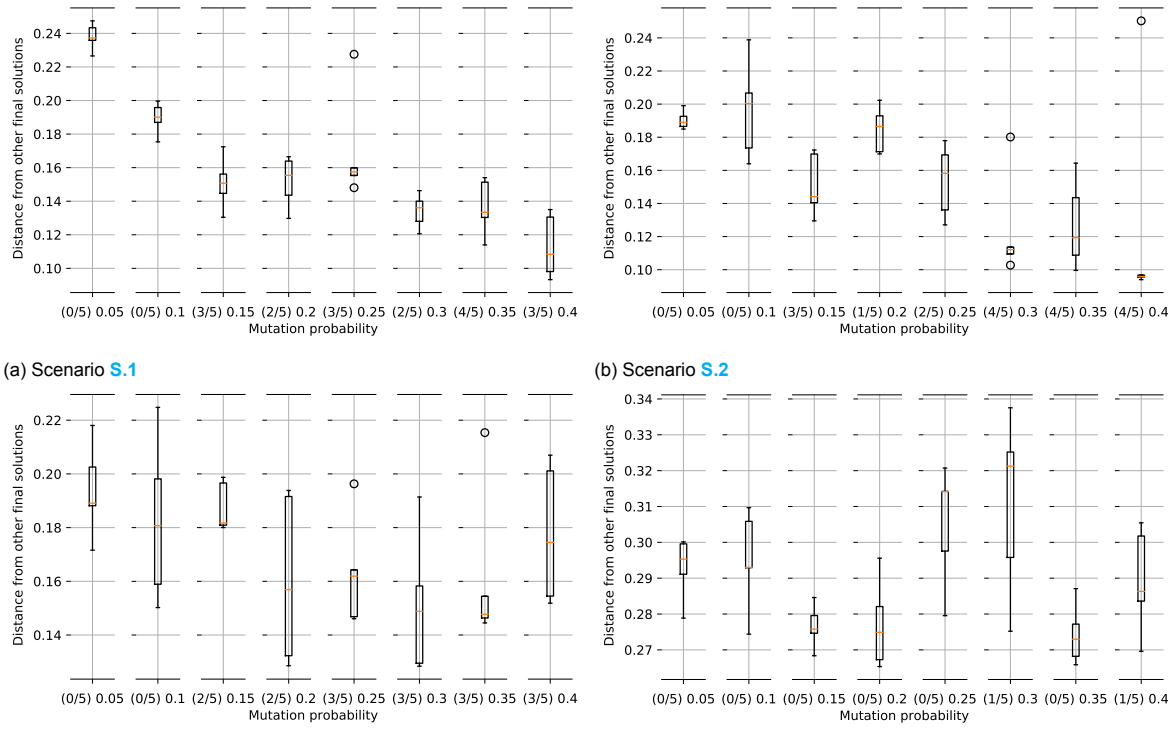
(a) Scenario S.1

(b) Scenario S.2

(c) Scenario S.3

(d) Scenario S.4

Figure B.10: First valid solution sample CMA-ES with varying initial step size



(a) Scenario S.1

(b) Scenario S.2

(c) Scenario S.3

(d) Scenario S.4

Figure B.11: Final solution distances from other final solutions in the same experiment for the genetic algorithm with varying mutation probability

B.2.4. Consistency

In Figures B.11 and B.12 the distance from the other final solutions for every valid final solution is shown for the GA and CMA-ES.

B.3. Combination

The only algorithm with configurable combination parameters is the GA. Table 7.3 shows the number of valid solutions found for every scenario given a crossover operation.

B.3.1. Optimality

In Figure B.13 the effects of the two different crossovers is shown on optimality is shown.

B.3.2. Convergence Speed

In Figure B.14 the number of iterations before convergence is shown for the two different crossover operations.

B.3.3. Generations Until Valid

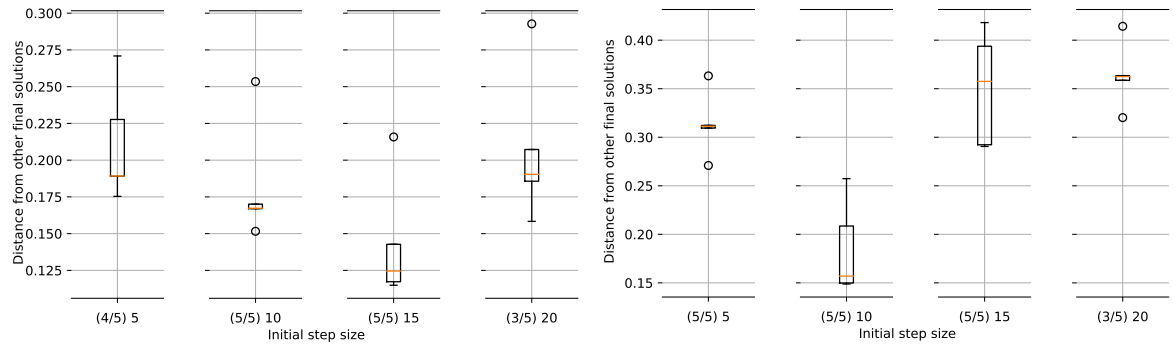
The first generation with a valid solution is shown in Figure B.15 for every crossover operation.

B.3.4. Consistency

In Figure B.16 the consistency of the GA with both crossover operations is shown.

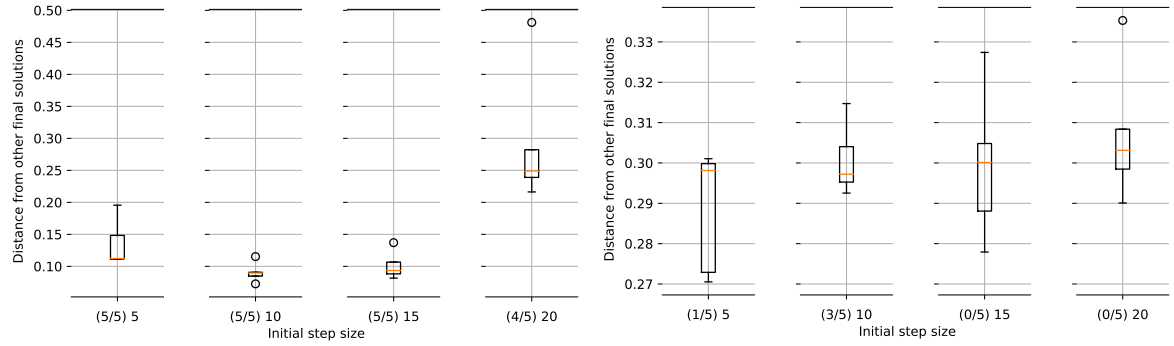
B.4. Initial Solution

In these experiments, instead of the algorithms starting with random initial solution, the algorithms use the existing heating curve solution as a starting point. This is a more realistic scenario when the optimisation is used in practice, as existing solutions will be used that have already optimised the operations with the information available at the time of the previous optimisation.



(a) Scenario S.1

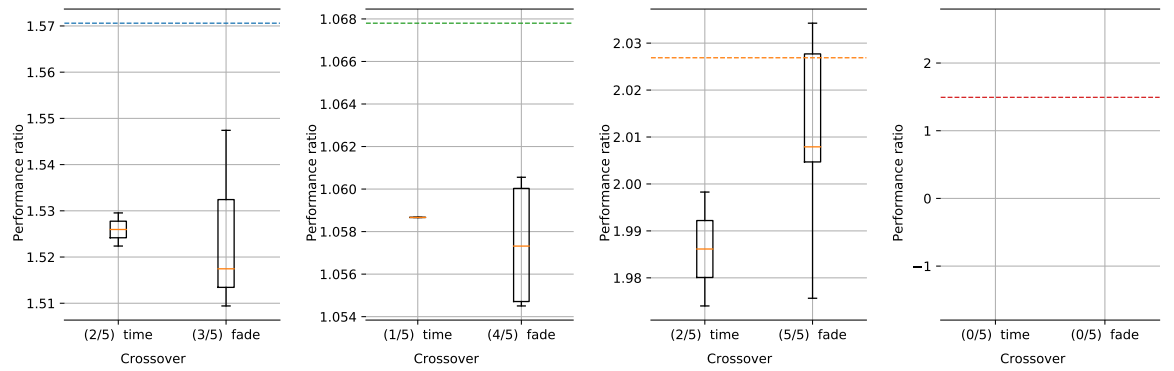
(b) Scenario S.2



(c) Scenario S.3

(d) Scenario S.4

Figure B.12: Final solution distances from other final solutions in the same experiment for CMA-ES with varying initial step size



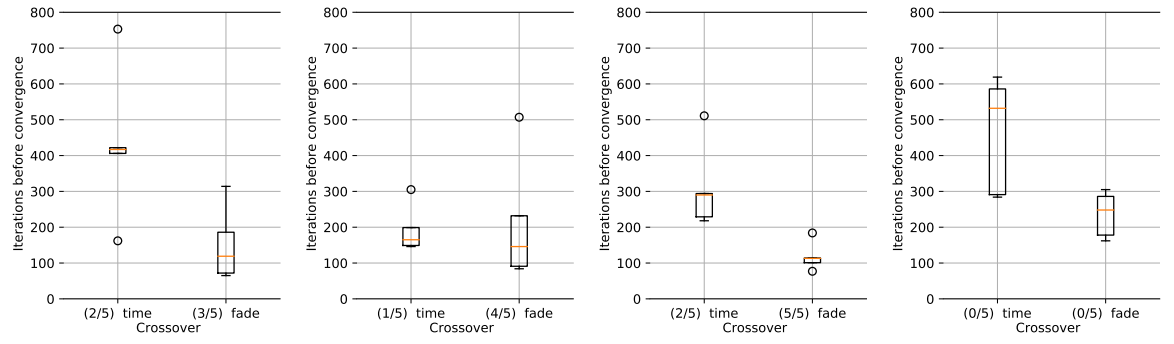
(a) Scenario S.1

(b) Scenario S.2

(c) Scenario S.3

(d) Scenario S.4

Figure B.13: Performance ratio with varying crossover



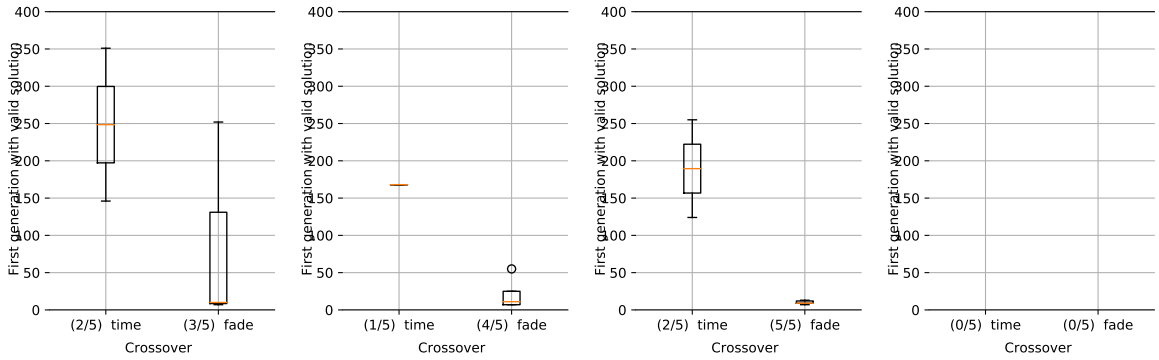
(a) Scenario S.1

(b) Scenario S.2

(c) Scenario S.3

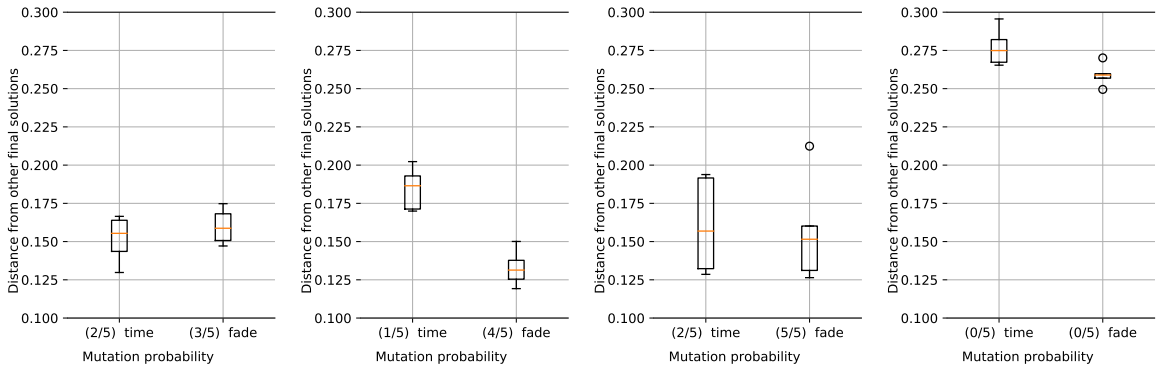
(d) Scenario S.4

Figure B.14: Number of iterations before convergence with varying crossover



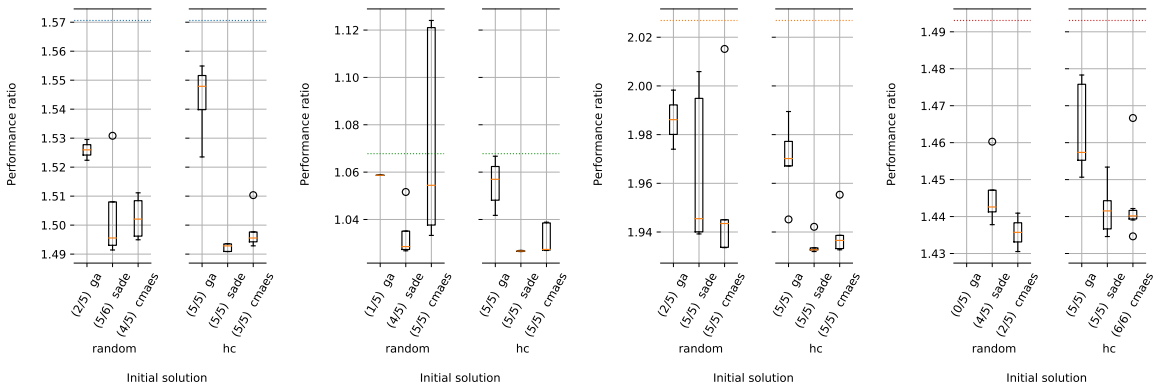
(a) Scenario S.1 (b) Scenario S.2 (c) Scenario S.3 (d) Scenario S.4

Figure B.15: First generation with a valid solution with varying crossover. See Table 7.3 for the number of valid solution found for every crossover



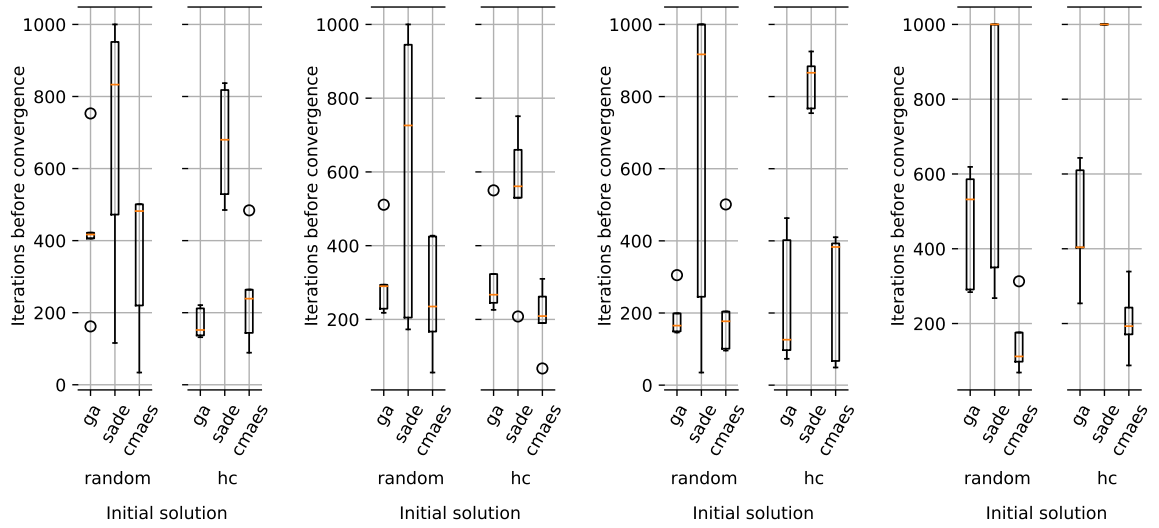
(a) Scenario S.1 (b) Scenario S.2 (c) Scenario S.3 (d) Scenario S.4

Figure B.16: Final solution distances from other final solutions in the same experiment varying crossover



(a) Scenario S.1 (b) Scenario S.2 (c) Scenario S.3 (d) Scenario S.4

Figure B.17: Performance ratios with varying initial solutions. The blue dashed line is the performance ratio of the heating curve solution



(a) Scenario S.1 (b) Scenario S.3 (c) Scenario S.2 (d) Scenario S.4

Figure B.18: Iterations before convergence with varying initial solutions

B.4.1. Optimality

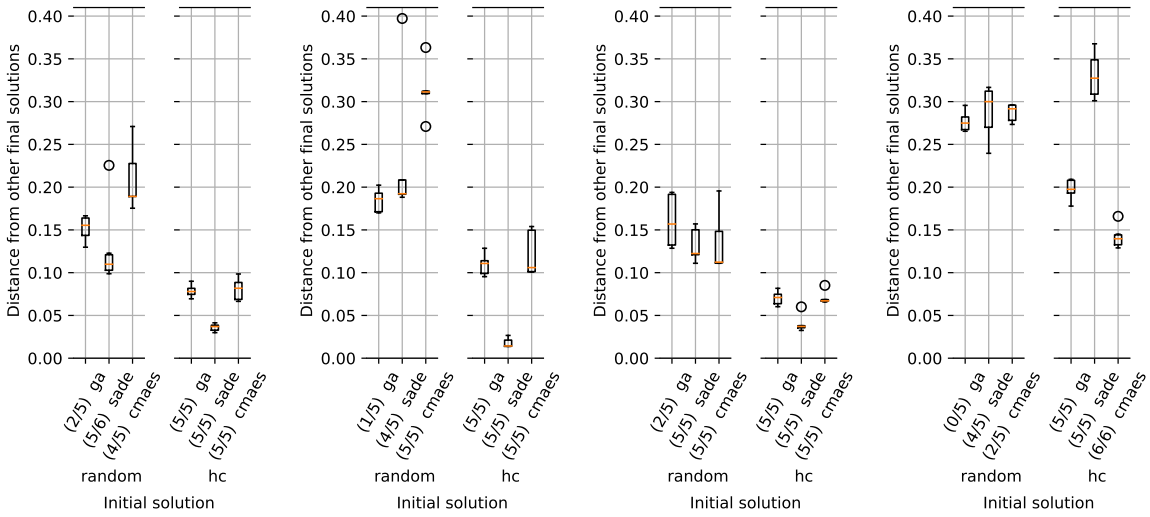
In Figure B.17 the performance ratios for the algorithms with varying initial solutions is shown. The blue dashed line is the performance ratio of the heating curve solution.

B.4.2. Convergence Speed

In Figure B.18 the number of iterations before convergence is shown.

B.4.3. Consistency

In Figure B.19 the consistency of the algorithms is shown.



(a) Scenario S.1 (b) Scenario S.2 (c) Scenario S.3 (d) Scenario S.4

Figure B.19: Final solution distances from other final solutions in the same experiment varying initial solution

Bibliography

[1] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, 1996. ISSN 00983500. doi: 10.1145/235815.235821.

[2] R. Bavière and M. Vallée. Optimal Temperature Control of Large Scale District Heating Networks. In *Energy Procedia*, 2018. doi: 10.1016/j.egypro.2018.08.170.

[3] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[4] Atli Benonysson, Benny Bøhm, and Hans F. Ravn. Operational Optimization in a District Heating System. *Energy conversion and management*, 36(5):297–314, 1995.

[5] Johannes Dorfner and Thomas Hamacher. Large-scale district heating network optimization. *IEEE Transactions on Smart Grid*, 5(4):1884–1891, 2014. ISSN 19493053. doi: 10.1109/TSG.2013.2295856.

[6] Erik Dotzauer. Simple model for prediction of loads in district - heating systems. *Applied Energy*, 73(3-4):277–284, 2002. ISSN 03062619. doi: 10.1016/S0306-2619(02)00078-8.

[7] Tingting Fang and Risto Lahdelma. Genetic optimization of multi-plant heat production in district heating networks. *Applied Energy*, 159:610–619, dec 2015.

[8] Samira Fazlollahi and François Maréchal. Multi-objective, multi-period optimization of biomass conversion technologies using evolutionary algorithms and mixed integer linear programming (MILP). *Applied Thermal Engineering*, 50(2):1504–1513, 2013. ISSN 13594311. doi: 10.1016/j.applthermaleng.2011.11.035. URL http://dx.doi.org/10.1016/j.applthermaleng.2011.11.035.

[9] Loïc Giraud, Roland Bavière, Cédric Paulus, Mathieu Vallée, and Jean-François Robin. Dynamic Modelling , Experimental Validation and Simulation of a Virtual District Heating Network. *ECOS2015, 28th International Conference on Efficiency, Cost, Optimization, Simulation and Environmental Impact of Energy Systems*, (June), 2015.

[10] Loïc Giraud, Massinissa Merabet, Roland Baviere, and Mathieu Vallée. Optimal Control of District Heating Systems using Dynamic Simulation and Mixed Integer Linear Programming. In *12th International Modelica Conference*, pages 141–150, jul 2017. doi: 10.3384/ecp17132141. URL http://www.ep.liu.se/ecp/article.asp?issue=132{%}26article=14.

[11] Pawel Gora and Marek Bardoński. Training neural networks to approximate traffic simulation outcomes. *5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems, MT-ITS 2017 - Proceedings*, pages 889–894, 2017. doi: 10.1109/MTITS.2017.8005639.

[12] S. Grosswindhager, M. Kozek, Andreas Voigt, and Lukas Haffner. Fuzzy predictive control of district heating network. *International Journal of Modelling, Identification and Control*, 19(2):161–170, 2013. ISSN 1746-6172. doi: 10.1504/ijmic.2013.054320.

[13] Guus Haas de, Frans Haas de, and Oel Clarine van. Cijfers voortgang uitfasering open- verbrandingstoestellen. Technical Report November, 2016. URL https://www.rijksoverheid.nl/binaries/rijksoverheid/documenten/rapporten/2016/11/22/cijfers-voortgang-uitfasering-open-verbrandingstoestellen/cijfers-voortgang-uitfasering-open-verbrandingstoestellen.pdf.

[14] Nikolaus Hansen. The CMA Evolution Strategy: A Tutorial. 2016. URL http://arxiv.org/abs/1604.00772.

[15] Nikolaus Hansen, Youhei Akimoto, and Petr Baudis. CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634, February 2019. URL https://doi.org/10.5281/zenodo.2559634.

[16] Y. Hori, A. Yamada, M. Shimoda, M. Bannai, and K. Ito. Development of optimal planning method using a genetic algorithm for district heating and cooling plants with heat storage tanks. *Kagaku Kogaku Ronbunshu*, 22(4):700–701, 1996. ISSN 0386216X. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-24044478859{&}partnerID=40{&}md5=4c8bd01e8c6a9eb6e3e0fa2103d6abdc.

[17] Enso Ikonen, Istvan Selek, Jeno Kovacs, Markus Neuvonen, Zador Szabo, Jozsef Bene, and Jani Peurasaari. Short term optimization of district heating network supply temperatures. *ENERGYCON 2014 - IEEE International Energy Conference*, pages 996–1003, 2014. doi: 10.1109/ENERGYCON.2014.6850547.

[18] Enso Ikonen, Istvan Selek, Jeno Kovacs, and Markus Neuvonen. Examination of operational optimization at Kemi district heating network. *Thermal Science*, 20(2):667–678, 2016. ISSN 03549836. doi: 10.2298/TSCI131119039I.

[19] Sabine Jansen. 'Cool heat grids' for sustainable urban energy systems, 2019. URL https://www.tudelft.nl/evenementen/2019/tu-delft/01-january/urban-energy-lecture-by-sabine-jansen-tu-delft/.

[20] Jesse Klaver, Lodewijk Asscher, Sandra Beckerman, Rob Jetten, Carla Dik-Faber, Dilan Yesilgöz-Zegerius, Agnes Mulder, and Simon Geleijnse. Klimaatwet, 2018. URL https://www.eerstekamer.nl/wetsvoorstel/34534_initiatiefvoorstel_klaver. [Accessed August 11, 2019].

[21] Leo Laakkonen. *Predictive Supply Temperature Optimization of District Heating Networks*. PhD thesis, Tampere University of Technology, 2016.

[22] C Lataniotis, S Marelli, and B Sudret. Extending classical surrogate modelling to ultrahigh dimensional problems through supervised dimensionality reduction: a data-driven approach. *arXiv e-prints*, page arXiv:1812.06309, dec 2018.

[23] Xiang-li Li, Lin Duanmu, and Hai-wen Shu. Optimal design of district heating and cooling pipe network of seawater-source heat pump. *Energy and Buildings*, 42(1):100–104, 2010. ISSN 03787788. doi: 10.1016/j.enbuild.2009.07.016.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. pages 1–9, 2013. URL http://arxiv.org/abs/1312.5602.

[25] Henrik Aalborg Nielsen and Henrik Madsen. *Predicting the heat consumption in district heating systems using meteorological forecasts*. PhD thesis, Technical University of Denmark, 2000. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.177.4659{&}rep=rep1{&}type=pdf.

[26] Kenneth Price and Rainer Storn. Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, (11):341–359, 1997.

[27] Provincie Zuid-Holland. Visie Aardwarmte Zuid-Holland. Technical report. URL http://www.zuid-holland.nl/publish/pages/23180/visieaardwarmtezuid-holland.pdf.

[28] A.K. Qin and P.N. Suganthan. Self-adaptive Differential Evolution Algorithm for Numerical Optimization. (May 2014):1785–1791, 2005. doi: 10.1109/cec.2005.1554904.

[29] Masatoshi Sakawa, Kosuke Kato, and Satoshi Ushiro. Operational planning of district heating and cooling plants through genetic algorithms for mixed 0-1 linear programming. *European Journal of Operational Research*, 137(3):677–687, 2002. ISSN 03772217. doi: 10.1016/S0377-2217(01)00095-9.

[30] B.V. Sazanov and O.O. Mil'man. Selection of Cycles and Parameters for District Heating Gas Turbine Plants. *Teploenergetika*, (12):76–79, 1969. ISSN 00403636 (ISSN). URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-0014660749{&}partnerID=40{&}md5=7d0fc6acb3c90d14060ef15edf1bbc5f.

[31] I Selek, J G Bene, and E Ikonen. Utilizing permutational symmetries in dynamic programming - with an application to the optimal control of water distribution systems under water demand uncertainties. *International Journal of Innovative Computing, Information and Control*, 9(8):3091–3113, 2013. ISSN 13494198 (ISSN). URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84880127865{&}partnerID=40{&}md5=2d56ff2b5509cb51c777108e8bbccc3c.

[32] Istvan Selek and Jozsef G. Bene. Optimal control of mass / energy distribution networks under uncertainties. *18th Nordic Process Control Workshop*, 2012.

[33] Wikimedia Commons, user: Sentewolf. Concept of directional optimization in cma-es algorithm.png, 2008. URL https://commons.wikimedia.org/wiki/File:Concept_of_directional_optimization_in_CMA-ES_algorithm.png. [Online; Released into public domain; accessed August 5, 2019].

[34] Jing Zeng, Jie Han, and Guoqiang Zhang. Diameter optimization of district heating and cooling piping network based on hourly load. *Applied Thermal Engineering*, 107:750–757, 2016. ISSN 13594311. doi: 10.1016/j.applthermaleng.2016.07.037. URL http://dx.doi.org/10.1016/j.applthermaleng.2016.07.037.