Trabajo Fin de Máster

Optimization in gas networks

Roland Vallery

Máster en Técnicas Estadísticas

Curso 2018-2019

Propuesta de Trabajo Fin de Máster

Título en galego: Optimización baixo incertidumbre en redes de gas

Título en español: Optimización bajo incertidumbre en redes de gas

English title: Optimization in gas networks

Modalidad: Modalidad B

Autor: Roland Vallery, Universidade de Santiago de Compostela

Director: Julio González Díaz, Universidade de Santiago de Compostela

Tutor: Diego Rodríguez Martínez, Itmati

Breve resumen del trabajo:

En este trabajo el alumno llevará a cabo tareas de modelado, implementación y validación de algoritmos de optimización.
Para la realización del trabajo el alumno se integrará en el proyecto de colaboración entre ITMATI y Reganosa, en el contexto de optimización de redes de gas.
Actualmente en dicho proyecto se está llevando a cabo una refactorización completa de los módulos de GANESO (herramienta desarrollada en el marco de la colaboración) y, en particular, de los módulos de optimización.
El alumno deberá familiarizarse con los problemas de optimización que resuelven dichos módulos y colaborar en la validación de los algoritmos implementados. Esto pasará por comparar dichos algoritmos con los resultados obtenidos con solvers "state of the art" gratuitos y comerciales.
Además, el alumno colaborará en la adaptación de dichos algoritmos a nuevos problemas de optimización que han sido identificados desde la empresa.

Recomendaciones: El alumno deberá haber cursado la asignatura de programación lineal y entera y haber cursado o estar matriculado de la asignatura de programación matemática.

Otras observaciones:

Agradecimientos

Quisiera agradecer a los docentes y a los estudiantes del Máster en Técnicas Estadísticas, que hicieron que - aunque los estudios hayan sido muy exigentes para mí - la atmósfera haya sido siempre muy agradable.

Me gustaría agradecer al grupo donde he realizado este trabajo, Diego, Ángel y Damián por su apoyo y en particular a Julio por su competente asistencia, amabilidad, así como por su paciencia.

Por último, quisiera agradecer a Onetsine su soporte y perseverancia.

Danke an meine Familie und Freunde für ihre Unterstützung.

Contents

Resumen

Resumen en español

Las redes de gas deben garantizar el transporte de gas a un coste mínimo desde los proveedores hasta los consumidores y al mismo tiempo tienen que cumplir restricciones físicas. Aunque esta tarea se pueda modelar analíticamente usando modelos matemáticos complejos, es difícil para los solucionadores actuales encontrar soluciones óptimas globales o a veces incluso locales.

Por lo tanto, este trabajo investiga distintas configuraciones de solucionadores de redes de gas para mejorar el proceso de resolución. Las configuraciones propuestas consisten en pasos secuenciales que usan la solución previa como solución inicial del paso siguiente. En cada paso se pueden elegir distintos parámetros de la configuración, por ejemplo el solucionador empleado o los componentes de la red de gas considerados. Para comparar configuraciones diferentes hemos desarrollado una herramienta informática que permite testear rendimientos de las configuraciones complejas para escenarios generados.

Nuestros resultados muestran que empleando pasos secuenciales se puede mejorar significativamente la proporción de escenarios solucionados. Secuencias de pasos incluyendo todos los solucionadores fueron las únicas configuraciones que pudieron solucionar todos los escenarios solucionados por alguna configuración testeada.

English abstract

Gas networks should ensure a cost-minimal supply of gas from providers to consumers while fulfilling various physical constraints. Although this task may be modeled analytically with complex mathematical models, for current solvers it is difficult to find global - and sometimes even local - optima for these models.

Therefore, this master's thesis investigates various gas network solver configurations to enhance the solution process. The solver configurations we propose consist of sequences of steps that use the solution of a previous step as initial solution for the succeeding step. In each step different configuration parameters can be chosen, for example the solver to employ or the gas network model components to take into consideration. To compare different configurations we developed a software framework that enables us to test the performances of complex configurations on solving generated gas network scenarios.

Our results show that using sequences of steps can significantly improve the proportion of solved scenarios. Step sequences comprising all solvers were the only configurations that could solve all scenarios that were solved by any tested configuration.

List of Figures

List of Tables

1 Introduction

Maintaining gas networks imposes a difficult task on the distributor and operator of the network. While being bound by physical gas laws, at the same time the operator strives to optimize its costs. Figure 1 gives an overview of the setup of the gas network in Europe. In this setup we can distinguish various components: supply points, consumption points, compressor stations to regulate the pressure in the network, valves, pipelines, junctions and in detail many more. These components and natural gas laws lead to certain restrictions the operator has to bear in mind, in particular on the physical quantities involved (esp. pressure and flow). For example, naturally at every node the sums of incoming and outgoing gas flows have to coincide.

So, in a gas network we can distinguish between optimization goals and constraints. Typical optimization goals are for example the minimization of gas consumed in the network or the minimization of the number of compressor stations. Constraints result primarily from the laws of physics, e.g. the gas laws, and the interaction of the components (Á. M. G. Rueda 2017). Optimization goals as well as constraints can to a certain degree be modeled as a mathematical optimization problem. Furthermore, modern software packages allow to employ sophisticated solvers to find optimal solutions for these problems. Nevertheless, the resulting models may turn out to be too difficult for some or all solvers to minimize.

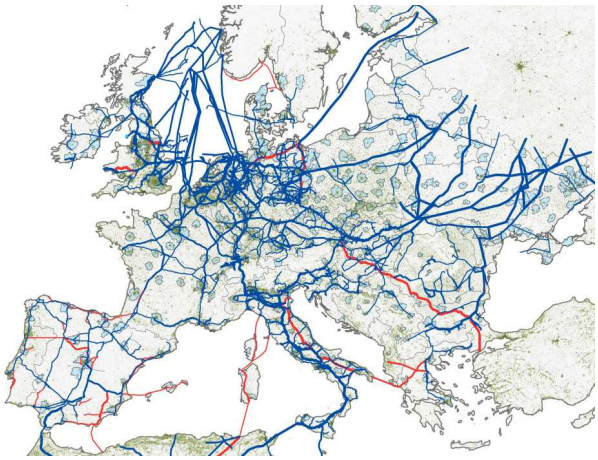

Figure 1: European gas network (Image: ETH Zurich) with planned pipelines in red, green spots represent dense population

Moreover, a general problem of multimodal mathematical optimization, which we also face in gas network models, lies in the fact that the solutions returned by most solving algorithms depend heavily on an initial solution (Fourer, Gay, and Kernighan 1989), which can be specified by the user or is chosen by the algorithm. Often, it is challenging to provide an initial solution that leads to a "good" result of the algorithm[1].

Therefore, this thesis investigates the performance of different solvers and validates approaches to support the solvers in finding promising solutions by providing them with further hints in multiple steps, exploiting the characteristics of the gas network setup. These hints consist in simplifications and extensions of the gas network model which serve to give clues to the solvers. The clues might indicate with which initial configurations to start searching for optimal solutions or the clues might consist in formulating the model in a way that makes it easier for the solvers to optimize, for example.

This master's thesis is structured as follows: in the chapter Mathematical optimization of basic gas networks we first explain gas networks in general and summarize existing modeling and solving techniques for gas networks. We continue with an example of the mathematical modeling of gas networks and end the chapter with the formulation of our research problem, which consists in identifying parameters that have an effect on the solution process and in optimizing the selection of these parameters. The chapter Gas network solver configuration strategies elucidates the techniques we apply to reach our research goal, that is, we identify tuning parameters and propose to simulate various networks and sequential configurations. In the course of carrying out this research thesis we developed a software framework that allows us to launch test batteries against different gas network scenarios, which we sketch in chapter 4. Along with the software framework, in chapter 4 we also clarify core concepts of this thesis and analyze the relationships between these concepts. Chapter 5 describes the experimental setup of the test batteries along with the results of the test batteries.

[1] In the section Research goal we will discuss by what criteria we judge the quality of a solution in this master's thesis, respectively, what we consider to be a "good" result.

Concluding, we give recommendations on configurations for different objectives , describe the shortcomings of this thesis and outline future work that would make sense to precede this thesis.

The section Definitions in the appendix lists definitions of reappearing key terms of this thesis. Also in the appendix, we give an introduction to mathematical programming.

2 Mathematical optimization of basic gas networks

2.1 Mathematical modelling of gas networks

In this master's thesis we apply mathematical programming techniques (see section introduction to mathematical programming) to the modeling of gas networks. Therefore, we devote this chapter to explaining existing approaches and open questions regarding the modeling of gas networks. Mathematically, gas networks are typically represented in the form of connected graphs as is illustrated by the European gas network presented in figure 1. In these graphs edges mainly define pipelines or compressor stations through which the gas flows. Nodes usually represent consumption or supply points.

When modeling a gas network we are typically confronted with various optimization goals and different physical restrictions that have to be taken into account (Á. M. G. Rueda 2017). The starting position for the optimization approaches we undertake in this thesis is as follows: we assume that the structure of the network - comprising the physical components like pipelines, compressors and supply and consumption points - is given and cannot be changed. Furthermore, the structure of the network, which we also call scenario, as well as the real gas demand establish boundaries on physical magnitudes like pressure or mass flow at different points in the network. What can be changed, however, is for example the amount of gas we introduce into the network at each supply point, as long as the specified boundaries are fulfilled. That is to say, we do not aim to improve the network structure. Instead, we strive to optimize the gas flow configuration for a given network structure. Furthermore, we assume the gas network to be in steady state. Above that, we restrict the resulting optimization problem to gas network systems that minimize the gas consumption with the help of compressor stations and under the constraint of preserving gas quality.

In the following section we will give a complete mathematical specification of the gas networks we consider in this thesis. We call such a specification a model. A more thorough presentation of the mathematical modeling of gas networks can be found in (Á. M. G. Rueda 2017).

2.2 Gas network specification

The basic elements of our gas network specification, i.e. of our model, are variables and parameters. Variables stand for all possible values of the quantities we are interested in and can be changed by solvers, which we employ to solve models. Parameters, on the other hand, are fixed and cannot be changed by the solvers. We distinguish between two different types of parameters. The first kind of parameters are given by the network scenario and its resulting physical constraints. The second kind of parameters, which we call alpha parameters, are used to calibrate terms in the model, in particular in the objective function.

Tables 1, 2 and 3 show the mathematical symbols we use throughout this thesis for variables, parameters and sets. The values of parameters with no default value will be assigned by the scenario.

Table 1: Variable notations, definitions and bounds

Variable	Definition	Lower bound	Upper bound
$\Delta_{q_{(j,k)}}$	difference of square pressures in compressor between nodes j and k	0	
c_{ini_k}	mass flow rate exchanged with the exterior at node k	$c_{\text{ini}_{\min_k}}$	$c_{\text{ini}_{\max_k}}$
cv_k	calorific value at node k	$cv_{\min_k}$	$cv_{\max_k}$

Table 1: Variable notations, definitions and bounds *(continued)*

Variable	Definition	Lower bound	Upper bound
$g_{(j,k)}$	self-consumption of a compressor	0	
p_k^2	squared pressure at node k	$\frac{p^2_{\min_k}}{s_p}$	$\frac{p^2_{\max_k}}{s_p} + \frac{p^2_{\text{extramax}_k}}{s_p}$
$p_{\text{avg}_{(j,k)}}$	square root of the average of the square pressures of the pipe	0	
p_{extra_k}	extra pressure at node k	0	$p_{\text{extra}_{\max_k}}$
$q^-_{(j,k)}$	mass flow in the direction from node k to node j	0	$\max(\lvert q_{\min_k}\rvert, \lvert q_{\max_k}\rvert)$
$q^+_{(j,k)}$	mass flow in the direction from node j to node k	0	$\max(\lvert q_{\min_k}\rvert, \lvert q_{\max_k}\rvert)$
$q_{(j,k)}$	mass flow from node j to node k, the sign determines the direction	$q_{\min_k}$	$q_{\max_k}$
$Z_{(j,k)}$	compressibility factor of the pipe between the nodes j and k		

Table 2: Parameter notations and definitions

Parameter	Definition	Default value
α_{absqual}	penalty associated to the gas quality	0.0001
α_{difpres}	penalty associated to the pressure difference in compressors	1e-13
α_{gc}	penalty associated to gas consumption in compressors	1
α_{pextra}	penalty associated to the surplus pressure	1
$\epsilon^c_{(j,k)}$	mechanic efficiency of the compressor	
$\eta_{(j,k)^c}$	optimal isentropic efficiency of the compressor.	
γ	ratio of specific heats	1.3031118413795963
$\lambda_{(j,k)}$	friction factor that describes the losses due to the viscious friction in a pipe	$\frac{1}{ef^2}\frac{0.009427}{\sqrt[3]{D_{(j,k)}}}$

Table 2: Parameter notations and definitions *(continued)*

Parameter	Definition	Default value
θ	average of the gas temperature in the pipes	288.15000000000026 K
θ_c	critical temperature	198.03994571477782 K
$\xi^c_{(j,k)}$	efficiency of the gas turbine which drives the compressor	
$c_{\text{ini}_{\min_k}}$	upper bound of the mass flow rate exchanged with the exterior at node k	
$c_{\text{ini}_{\max_k}}$	lower bound of the mass flow rate exchanged with the exterior at node k	
$cv_{\max_k}$	upper bound of the calorific value at node k	
$cv_{\min_k}$	lower bound of the calorific value at node k	
$D_{(j,k)}$	diameter of the pipe between the nodes j and k	
ef	Weymouth's efficiency coefficient	0.95
g	acceleration of the gravity	9.81 $\frac{m}{s^2}$
h_j	height at the the beginning of the pipe	
h_k	height at the the end of the pipe	
$L_{(j,k)}$	length of the pipe between the nodes j and k	
$p_{\text{extra}_{\max_k}}$	maximum extra pressure at node k	1
$p_{\max_k}$	maximum pressure at node k	
$p_{\min_k}$	minimum pressure at node k	
p_c	critical pressure	4.605692623049976 MPa
$q_{\max_k}$	maximum mass flow at edge k	
$q_{\min_k}$	minimum mass flow at edge k	
R	gas constant	487.6288819794937 $\frac{kJ}{kmol\,K}$

Table 2: Parameter notations and definitions *(continued)*

Parameter	Definition	Default value
s_{Δ_q}	pressure difference in compressors scaling factor	1e-10
s_{cv}	calorific value scaling factor	1
s_g	gas consumption in compressors scaling factor	1
s_l	pressure loss scaling factor	1
$s_{p_{\text{avg}}}$	mean pressure scaling factor	1
s_Z	compressibility scaling factor	$\sqrt{1e-10}$
s_p	pressure scaling factor	1e-10
s_q	flow scaling factor	1e-10

Table 3: Set notations and definitions

Set	Definition
E	set of edges
EC	set of compressor stations
EP	set of edges with potential pressure loss, in particular compressor stations
V	set of nodes

The gas networks we strive to optimize shall fulfill the following constraints (1) to (10). We multiply both sides of the equations for each constraint with a scaling factor to keep all magnitudes within "compatible ranges" (Bazaraa, Sherali, and Shetty 2006, 29).

Flow conservation:

$$s_q \sum_{(k,l)\in E} q_{(k,l)} = s_q \sum_{(j,k)\in E} \left(q_{(j,k)} + c_{\text{ini}_k}\right) \ \forall k \in V \tag{1}$$

That is, the sums of incoming and outgoing flows at every node k have to coincide except for an allowed amount of flow c_{ini} that is exchanged with the exterior.

Pressure upper bound:

$$s_l s_p p_k^2 \leq s_l \left(s_p p_{\max_k} + s_p p_{\text{extra}_k}\right) \ \forall k \in V \tag{2}$$

The variable p_{extra_k} is intended to help the solvers in finding feasible solutions by first allowing surplus pressure in the constraints for the node k and later on penalizing the surplus pressure in the objective function (see equation (11)).

Pressure loss:

$$s_l\left(s_p p_j^2 - s_p p_k^2\right) = s_l\Big(\frac{16\lambda_{(j,k)} L_{(j,k)} Z_{(j,k)} R\theta |q_{(j,k)}| q_{(j,k)}}{\pi^2 D_{(j,k)}^5} + \frac{2\,g\,p_{\text{avg}_{(j,k)}}^2}{R\theta\left(1^{-20} + Z_{(j,k)}\right)}(h_k - h_j)\Big)\ \forall (j,k) \in EP \tag{3}$$

Determination of the compressibility factor Z using the AGA8 model for edges with potential pressure loss:

$$s_Z Z_{(j,k)} = s_Z\left(1 + \frac{0.257\,p_{\text{avg}_{(j,k)}}}{p_c} - \frac{0.533\,p_{\text{avg}_{(j,k)}}\theta_c}{p_c\theta}\right)\ \forall (j,k) \in EP \tag{4}$$

Determination of the compressibility factor Z using the AGA8 model for compressor stations:

$$s_Z Z_{(j,k)} = s_Z\left(1 + \frac{0.257\sqrt{s_p p_j^2}}{p_c} - \frac{0.533\sqrt{s_p p_j^2}\theta_c}{(p_c\theta)}\right)\ \forall (j,k) \in EC \tag{5}$$

We can make use of **compressors** to increase the pressure at the cost of consumed gas (see equation (11) for the objective function). Analogously, we can implement valves that decrease the pressure.

The compressibility factor Z tells us how the gas behaves in comparison with an ideal gas.

Mean pressure:

$$s_{p_{\text{avg}}}\sqrt{s_p} p_{\text{avg}_{(j,k)}} = s_{p_{\text{avg}}}\,\frac{2}{3}\left(\sqrt{s_p p_j^2} + \sqrt{s_p p_k^2} - \frac{\sqrt{s_p p_j^2}\sqrt{s_p p_k^2})}{\sqrt{s_p p_j^2} + \sqrt{s_p p_k^2}}\right)\ \forall (j,k) \in EP \tag{6}$$

Difference of square pressures in compressor:

$$s_{\Delta_q}\Delta_{q_{(j,k)}} = s_{\Delta_q}(s_p p_k^2 - s_p p_j^2)\ \forall (j,k) \in EC \tag{7}$$

The difference of pressures between nodes will be penalized in the objective function (see equation (11)).

Gas consumption:

$$s_g g_{(j,k)} = s_g\left(\frac{1}{\epsilon^c_{(j,k)}\xi^c_{(j,k)}\eta_{(j,k)^c} LCV}\,\frac{\gamma}{\gamma - 1} Z_{(j,k)} R\theta\big((\frac{s_p p_k^2}{s_p p_j^2})^{\frac{\gamma-1}{2\gamma}} - 1\big)|q_{(j,k)}|\right)\ \forall (j,k) \in EC \tag{8}$$

Gas quality:

$$s_{cv}\ cv_k\left(\sum_{(k,l)\in E} {q^-}_{(k,l)} + \sum_{(j,k)\in E} {q^+}_{(j,k)}\right) = s_{cv}\left(\sum_{(k,l)\in E} {q^-}_{(k,l)} cv_l + \sum_{(j,k)\in E} {q^+}_{(j,k)} cv_j\right)\ \forall k \in V \tag{9}$$

The calorific value cv_k is a specific property of a substance, in our case of a gas mixture. The gas quality constraint enforces that the calorific value at each node remains within given bounds. As equation (9) shows, we estimate the calorific value of the gas mixture at a node by the average of the calorific values of the components of the mixture weighted by their mass, which we determine by the mass flow at the node. As the following constraint (10) postulates, for each edge the mass flows in both directions add up to the total gas flow at the edge.

Directed absolute value decomposition of the total mass flow:

$$s_q(q_{(j,k)}) = s_q(q^+_{(j,k)} - q^-_{(j,k)}) \ \forall (j,k) \in E \tag{10}$$

In addition to the constraints regarding the gas quality listed in equations (9) and (10), we add a term to the objective function (11) that penalizes the sum of q^+ and q^-, so that at least one of the two variables should adopt the smallest value possible, which is 0.

The objective function, which we aim to minimize, is given in equation (11):

Objective function:

$$f = \alpha_{\text{pextra}} \sum_{k \in N} p_{\text{extra}_k} + \alpha_{gc} \sum_{(j,k) \in EC} g_{(j,k)} + \alpha_{\text{difpres}} \sum_{(j,k) \in EC} \Delta_{q_{(j,k)}} + \alpha_{\text{absqual}} \sum_{(j,k) \in E} \left(q^+_{(j,k)} + q^-_{(j,k)} \right) \tag{11}$$

As we already mentioned respectively for equations (2), (7) and (10), the first and last two summands of the objective function can be interpreted as mere constraints written as penalizations. The remaining second summand represents the actual goal considered in this thesis: minimizing the amount of gas consumed in compressor stations (cp. eq. (8)).

In summary, the problem of optimizing gas networks as specified in this section can be written as a nonlinear program as defined in the section introduction to mathematical programming.

The gas network model described in this section, including its implementation in Python and AMPL, had already been created at the beginning of writing this master's thesis, with the exception of gas quality components (see equations (9) and (10) as well as the respective term in the objective function). All elements of the model are integral parts of the GANESO software, which has been developed in cooperation between the University of Santiago de Compostela and the Technological Institute for Industrial Mathematics (ITMATI) for the energy company REGANOSA (Bermúdez et al. (2015)). REGANOSA is a Galician company "engaged in the transmission and regasification of natural gas" (REGANOSA (2018)). Currently, gas quality restrictions are being integrated into the GANESO software. In this context, the objective of this thesis was to test the performances of different approaches to solve the optimization problem resulting from the addition of gas quality constraints to the existing model.

2.3 Example gas network scenario

In the following we visualize and elucidate the structure of the gas networks investigated in this thesis by gradually adding components to a very simple example network. Accordingly, we begin with the basic gas network shown in figure 2, which only comprises the requirements of flow conservation (see equation (1)) at nodes (including a maximum flow difference c_{ini}) and lower and upper bounds for the squared pressure p^2 at nodes and the gas flow q at edges. The concrete requirements are drawn in red color. Restrictions concerning nodes are drawn next to the nodes, restrictions concerning edges are drawn next to the edges. For example, we model the squared pressure p^2 in the network as an attribute of the nodes. Also, we omit units of physical quantities. Parameters that we do not explicitly mention are set to their default values (see table 2).

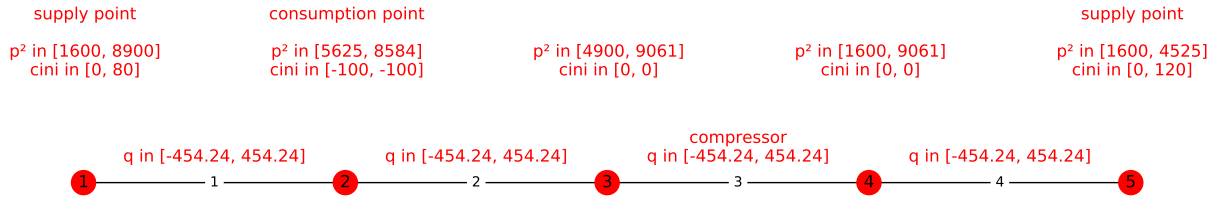

Figure 2: Basic gas network model restrictions

Nodes 1 and 5 of the basic example network represent supply points, meaning places where gas can be introduced into the network, but limited to a surplus (c_{ini}) of 80 at node 1 and 120 at node 5. Node 2 takes on the role of a consumption point, requiring a flow of 100 which gets consumed at the node.

We do not draw arrows representing the direction of the gas flow as the direction of the flow depends on the result of the optimization. In these examples, the flow is assumed to be positive if the gas flows from the node with the lesser number to the node with the greater number and negative in the opposite case.

In this basic example gas network we regard compressors as usual edges and in particular we do not take the consumption of gas at compressors (equation (8)) into account. Neither do we consider gas quality restrictions (equation (9)) or optimization goals (formula (11)) .

Thus, in the basic network the main requirement is that the flow from the left and the flow from the right at consumption node 2 have to add up to 100. Figure 3 gives us an arbitrary valid solution in blue color with a flow of 21.27 coming from the left and a flow of 78.73 coming from the right.

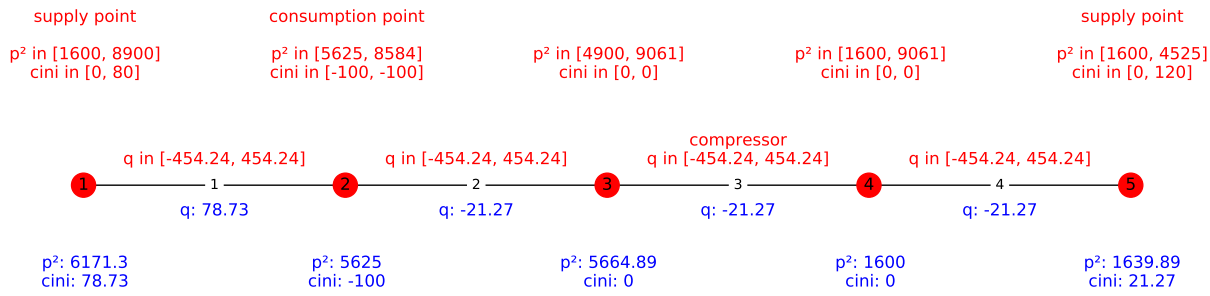

Figure 3: Basic gas network model restrictions and solution

Now, if we take into account that compressor stations consume gas which causes costs that we want to minimize, we have to add these costs to the objective function as laid out previously (see formula (11)) .

It becomes evident that under these premises we should insert as much gas as possible from supply point 1 into the network, thereby avoiding gas flow through the only compressor station in the network between nodes 3 and 4. However, as the maximum flow surplus in node 1 is 80, we are obliged to insert a flow of at least 20 from node 5 as well. More precisely, introducing an amount of exactly 20 units of gas from node 5 and 80 units from node 1 will lead to a global minimum of the objective function as we thereby minimize the amount of gas consumed at the compressor station. Figure 4 shows the gas network model including gas consumption at compressors and the resulting optimal solution.

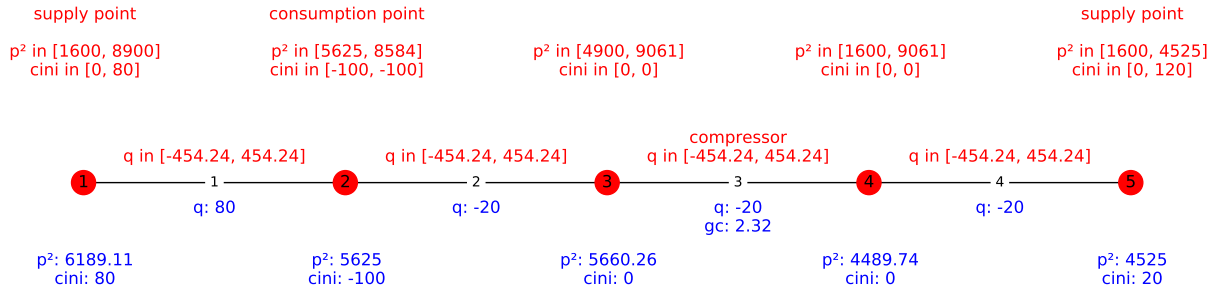

Figure 4: Gas network model including gas consumption with solution

Finally, let us add gas quality restrictions (equation (9)) to our example network as specified in figure 5. That is, we assume a calorific value *cv* of 10 for the gas introduced at node 1, a calorific value of 100 for the gas introduced at node 5, a valid range of 95 to 100 for the calorific value at the consumption node and a valid range of 10 to 100 for the remaining nodes.

Under these circumstances it is not sufficient to provide 20% of the gas coming from node 5, as this would result in a calorific value of $20\% \cdot 10 + 80\% \cdot 100 = 82 < 95$ in the consumption point (see equation (9)). In order to reach the required gas quality of at least 95 in the consumption point we have to introduce a flow of about 94.4 from node 5, and a flow of about 5.6 from node 1, yielding a gas quality of $5.6\% \cdot 10 + 94.4\% \cdot 100 \approx 95$ in the consumption point as is illustrated in figure 5.

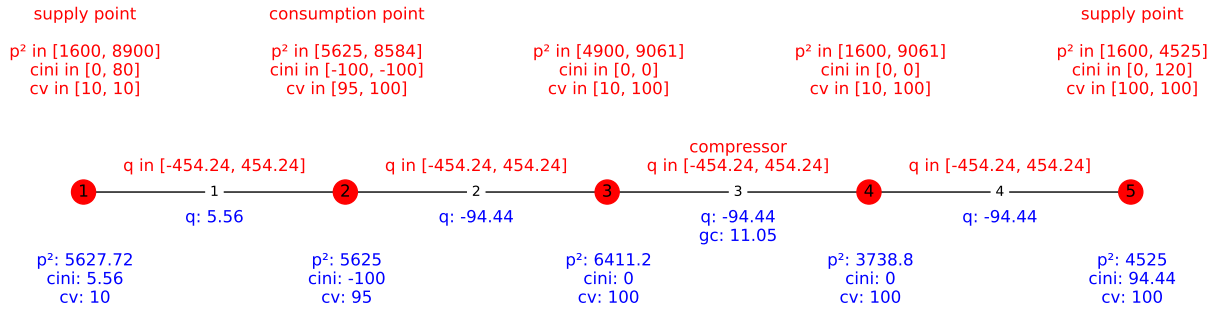

Figure 5: Gas network model including gas consumption and gas quality restrictions with solution

The examples in this chapter so far have been pretty simple and could easily be solved manually. However, for more complex networks this is not the case and sometimes not even standard solvers find a feasible solution for a given model. Let us consider for example a gas network with the restrictions shown in figure 6. If we apply the solver *Ipopt* with its default configuration to this network model, *Ipopt* does not find a feasible solution. However, the solver *Knitro* started with its default configuration returns the feasible solution drawn in figure 7 (with a rounded objective function value of 28.35). In the section Step sequences we will show a way to configure *Ipopt* in such a way that it finds the same solution *Knitro* found with its default configuration.

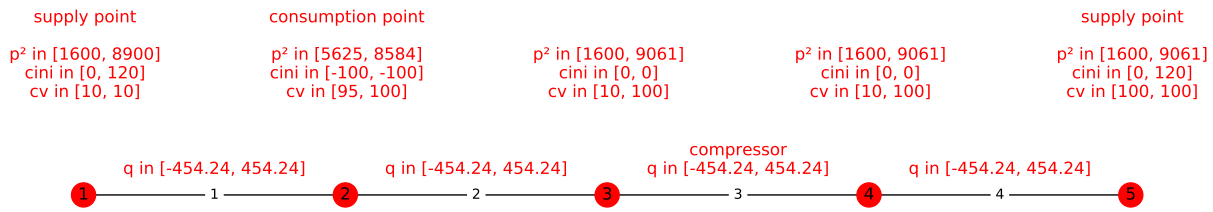

Figure 6: Gas network not immediately solvable by ipopt solver

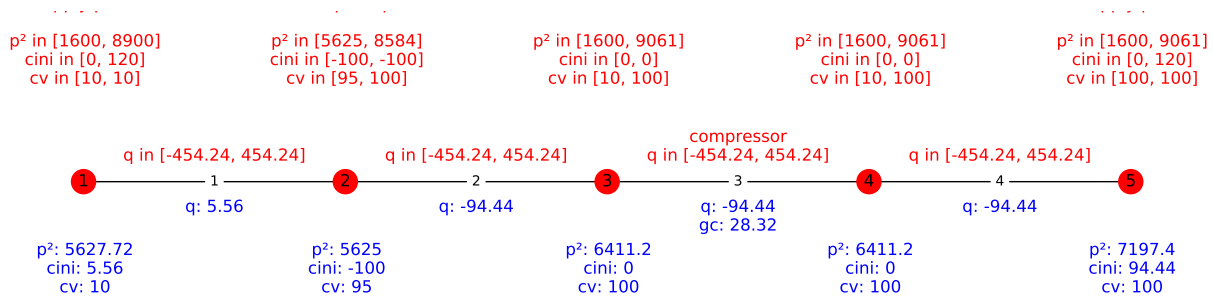

Figure 7: Gas network of figure 6 solved by knitro solver

As the solvers started with default configuration are not always able to find optimal solutions, the necessity for advanced solution methods arises which will be the topic of the next section.

2.4 Research goal

As we can see by the example in figure 6, it is not a trivial task to find a solver setup that can generate an at least feasible solution for a given gas network, let alone a feasible solution with a "low" objective function value. To enhance the process of choosing a promising solver configuration, in this thesis we aim to systematically analyze possible strategies to find configurations that have the potential to lead to desirable feasible solutions as stated by our research goal:

The objective of this thesis is to optimize the selection of configurations to solve gas network models. That is to say, we strive to systematically analyze which configurations are possible and compare the performance of different configurations in respect to

- their ability to solve gas network problems to feasibility,
- their objective function values,
- their slacks and
- their computation time.

Figure 8: Research Goal

Thus, we are confronted with a multiobjective optimization problem.

As gas networks vary in their characteristics, the performance of a configuration might depend on the concrete gas network scenario under consideration. Therefore, we aim to investigate the performance of configurations for different scenarios.

Part of our underlying assumptions is that we can apply the findings of optimizing configurations for the gas network model that we restrict ourselves to in this thesis to more complicated network models.

In the next section we will present strategies to achieve the objectives postulated above. To this end we will make use of existing techniques to solve mathematical optimizations problems described in the appendix, i.e. existing solvers and the *AMPL* modeling language.

3 Gas network solver configuration strategies

In this section we describe the techniques we apply in order to enhance the solution process of gas network optimization problems:

That is, we

- split the solution process into step sequences with different configurations per step (section 3.1),
- use a variety of simplifications and extensions of our gas network model in each step of a step sequence (section 3.2) and
- launch test batteries against different gas network scenarios (section 3.3).

3.1 Step sequences

As described in chapter 1, a difficult and important task in mathematical optimization consists in providing helpful initial solutions for the solvers. To tackle this problem and to discover promising initial solutions we propose to use sequences of steps with different step configurations. In each step of a sequence we consider a variation of the actual gas network problem of interest as specified in section 2.2. The solution of the model of a preceding step serves as initial solution for the next step. The actual gas network problem of interest always forms the last step of every step sequence.

Thus, just as we have gradually added components to the basic example network model in section 2.3, our approach to solve complicated gas network models involves employing sequences of network models with different components, generally - but not necessarily - starting with simpler network models.

For instance, as a proof of concept for this approach we might again consider the network shown in figure 6 of section 2.3: If we immediately try to solve this network with the solver *Ipopt* and its standard configuration, we do not obtain a valid solution - the solver regards the problem as "infeasible". Therefore, let us first contemplate a simplified version of the network, which ignores the gas quality restrictions but preserves all other requirements. For this simplified network *Ipopt* comes up with a solution, which is shown in figure 9.

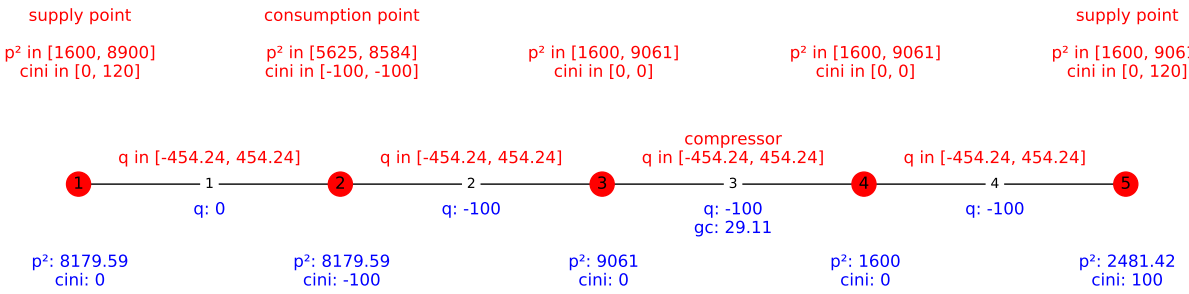

Figure 9: Simplified version of the gas network solved by ipopt solver

Interestingly, if we use the solution of this simplified network as initial solution for the complete network in figure 6, *Ipopt* is able to solve the complete network, that is, including the gas quality requirements and returns the solution given in figure 7. This works although the solution of the simplified network - i.e. the initial solution of the original network under study - is not the optimal solution returned by *Ipopt* for the original network when providing this solution as initial solution.

Having motivated our idea, our straightforward goal is to generalize this procedure and to try to solve generic - not immediately solvable - gas network models by varying its components and using the solutions of the modified networks as initial solutions for the solvers to obtain solutions for the actual network of interest.

In addition to simplifying and extending models and their alpha parameters we will also examine the effects of the following configuration parameters, which likewise may vary per step:

- *AMPL* options,
- solvers, including
- solver options

In the next section we will concretize our approach by listing all simplifications and extensions we allow for the gas networks under consideration in this thesis.

3.2 Model components

3.2.1 Model decomposition

We decompose the complete gas network specification (see section Gas network specification), i.e. the actual model of interest, into three components called *base*, *gas consumption* (abbreviated **gc**) and *gas quality* (**gq**). Thereby, we seek to generate simpler models that are easier to solve and that might lead to promising initial solutions for the solvers when solving the actual model of interest. Table 4 shows the elements of the complete specification (see section Gas network specification) that the three components comprise, respectively:

Table 4: Decomposition of the complete model specification into components

Model component	Variables with bounds	Constraints	Objective function terms
base	$q_{(j,k)}, p_k^2$, $p_{\text{avg}_{(j,k)}}$, p_{extra_k}, c_{ini_k}, $Z_{(j,k)}$	Flow conservation (1), Pressure upper bound (2), Pressure loss (3), compressbility factor (4), (5), Mean pressure (6)	$\alpha_{\text{pextra}} \sum_{k \in N} p_{\text{extra}_k}$
gas consumption (gc)	$g_{(j,k)}$, $\Delta_{q_{(j,k)}}$	Difference of square pressures in compressor (7), Gas consumption (8)	$\alpha_{gc} \sum_{(j,k) \in EC} g_{(j,k)}$, $\alpha_{\text{difpres}} \sum_{(j,k) \in EC} \Delta_{q_{(j,k)}}$
gas quality (gq)	cv_k, $q^+_{(j,k)}$, $q^-_{(j,k)}$	Gas quality (9), Decomposition of the mass flow (10)	$\alpha_{\text{absqual}} \sum_{(j,k) \in E} (q^+_{(j,k)} + q^-_{(j,k)})$

In each step of a step sequence we can solve a model consisting of a combination of the components shown in table 4. However, every valid combination has to contain the *base* model component. Every valid combination of model components is a model itself.

3.2.2 Model extensions

In addition to the decomposition of the complete model, we introduce a new *gas quality penalized* **(gqp)** component, which can be added to a model. The specification of the *gas quality penalized component* as an extension to a given model is presented in table 5.

Table 5: Model extensions

Model extension component	Removed constraint	New objective function term
gas quality penalized (gqp)	Gas quality (9)	$\alpha_{gq} \sum_{k \in N} (cv_k (\sum_{(k,l) \in E} q^-{}_{(k,l)} + \sum_{(j,k) \in E} q^+{}_{(j,k)}) - (\sum_{(k,l) \in E} (q^-{}_{(k,l)} cv_l) + \sum_{(j,k) \in E} (q^+{}_{(j,k)} cv_j)))^2$

Instead of ensuring gas quality via the gas quality constraint (9), the *gas quality component* penalizes gas quality deviations in the objective functions to help the solvers in the solution process (see section Mathematical optimization). The parameter α_{gq} has a default value of 0.01.

3.2.3 Model simplifications

Furthermore, we propose model simplifications to simplify models and facilitate the search for promising initial solutions. Table 6 shows the three simplifications that we consider in this thesis. These simplifications redefine and simplify constraints of the complete model.

Table 6: Model simplifications

Model simplification component	Redefined constraint	New constraint definition
delete heights (dh)	Pressure loss (3)	$s_l(p_j^2 - p_k^2) = s_l \frac{16 \lambda_{(j,k)} L_{(j,k)} Z_{(j,k)} R \theta \lvert q_{(j,k)} \rvert q_{(j,k)}}{\pi^2 D^5_{(j,k)}} \ \forall (j,k) \in EP$
mean pressure without denominator (mpwd)	Mean pressure (6)	$s_{p_{\text{avg}}}(3 p_{\text{avg}_{(j,k)}} (\sqrt{s_p p_j^2} + \sqrt{s_p p_k^2})) = s_{p_{\text{avg}}}(2(s_p p_j^2 + s_p p_k^2 + (\sqrt{s_p p_j^2} \sqrt{s_p p_k^2}))) \ \forall (j,k) \in EP$
simple mean pressure (smp)	Mean pressure (6)	$s_{p_{\text{avg}}}(p_{\text{avg}_{(j,k)}}) = s_{p_{\text{avg}}}(1/2(\sqrt{p_j^2} + \sqrt{p_k^2})) \ \forall (j,k) \in EP$

3.3 Configuration simulations

To recapitulate, the given combination of scenario and configuration determines if and how well we can reach our objectives, which means solving the scenario to feasibility with low objective function value, little slack and little computation time (see section Research Goal). The model that we ultimately try to solve is always the actual model of interest. In the last two sections we have identified possible **configuration** parameters. In this section we address the problem of evaluating the performance of configurations for multiple **scenarios**.

In this master's thesis it is infeasible to compare the performance of configurations analytically due to the complexity of the problem. Therefore, we propose to tackle the problem of evaluating the performance of different configurations by computationally launching different types of test batteries consisting of combinations of configurations and simulated scenarios and by measuring the performance of these combinations in respect

to our objectives when solving the actual model of interest. To solve models with a given combination of configuration and scenario we make use of an existing software described in (Bermúdez et al. (2015)).

The scenarios are generated randomly, based on the example gas network of section 2.3. The sheer quantity of possible combinations of configurations and scenarios makes it impossible to test all possible configurations on all possible scenarios, so that representative samples have to be drawn. In our simulations the generated gas networks function as experimental units to which we apply the treatment - i.e. the configuration - and for which we measure the results (objective function value, time and slack). In the Results chapter we will present the combinations we chose and their respective performances.

Summing up this chapter, our approach to enhance the solution process of gas networks models consists in computationally comparing different solver configurations. These configurations are made up of steps in which the solution of a previous step serves as initial solution for the succeeding step. We have identified the following configuration parameters per step: solver, gas network model, (penalizing) alpha parameters, *AMPL* options and solver options.

As a technical basis for this master's thesis serves an already existing software, which includes utilities to launch the solution process of gas network models with given scenarios and configurations (Bermúdez et al. (2015), Á. M. G. Rueda (2017)). In the next chapter we present a software framework, which extends the functionality of the existing software by providing tools to easily execute and analyze simulations of different configurations. The developed simulation framework is a key contribution of this thesis as it enables us to carry out our research approach.

4 Simulations framework

4.1 Conceptual model

In the course of this thesis we have developed an object-oriented software framework, which enables us to launch and analyze the test batteries proposed in section 3.3. In this chapter we will explain the elements of the conceptual model of the framework as sketched in figure 10. The notation mostly follows the conventions of the Unified Modeling Language (UML) (Rumbaugh, Jacobson, and Booch (2004)). Blue and yellow rectangles represent classes, lines represent relationships between the classes. A line with an open arrowhead symbolizes a navigable relationship in the direction of the arrow. For example, we can navigate from **steps** to their **configuration**. Unfilled arrowheads resemble generalizations, for example **GasConsumption** objects are specializations of **ModelComponent** objects. Diamonds at the end of an arrow denote that objects of the class at the end of the arrow are made up of objects of the class at the other side of the arrow, e.g. a **StepSequence** object *consists of* **Step** objects. Selected meaningful scalar attributes are drawn inside the rectangles, for example alpha parameters for **ModelComponent** objects. Ellipses drawn with dotted lines represent non-scalar attributes.

A **step sequence** forms part of a **configuration simulation**, which is the unit of of execution. A configuration simulation is linked to a unique **scenario**. Thus, all steps of the configuration simulation operate on the same gas network scenario. Conceptually, scenarios consist of objects of the classes **Node** and **Edge**. Steps implement the logic to launch the solution process of a given scenario with a given configuration. Also, steps store references to the **results** we obtain. In particular, we need to analyze **constraint results** (see section 2.2) to verify whether a configuration has **solved a scenario to feasibility**. Attributes of a step configuration are the **solver**, the **model**, **solver options** and ***AMPL* options**.

In every step we launch a distinctive model, i.e., a specification consisting of an objective function and restrictions. Models consist of model components in a variation of the composite pattern (Gamma et al. (1995)). A model always includes the **base** class, which provides basic functionality, especially access to the underlying ***AMPL*** API calls. All combinations of model components constitute new model components themselves. Nevertheless, not all combinations of model components are valid models, for example a model may not be composed of the *gas quality penalized* component without simultaneously also comprehending the *gas quality* component (see section 3.2).

Model component objects may be

- **parts of the complete model specification**, for example the ***gas consumption component*** (see sections 2.2. and section 3.2.1), or
- **model extensions**, for example the ***gas quality penalized component*** (see section 3.2.2), or
- **model simplifications**, for example the ***simple mean pressure component*** (see section 3.2.3), or
- **combinations** of other model components.

The **GasConsumption+GasQualityModel** is an example of a model component class that is generated by combining and thereby specializing multiple model component classes.

4.2 Implementation of the conceptual model

We implemented the conceptual model in Python. Blue rectangles in figure 10 represent classes which are not defined in the Python code but whose objects are generated via duck typing, i.e. by implementing the attributes and methods of the base classes. Thereby, we can cope with the large number of possible model classes. Thus, the inheritance arrows in figure 10 are not really specified in the program, but enforced by means of contract. Hence, relying on duck typing in our implementation we define a **model** to be the sum of **model components**, where the sum includes an object of the **Base** class as a summand. This allows for a valid concise Python syntax to create a complete model based on overloading the "+"-operator. In the following code chunk we create the object for the ***actual model of interest*** in Python syntax by summing up its components:

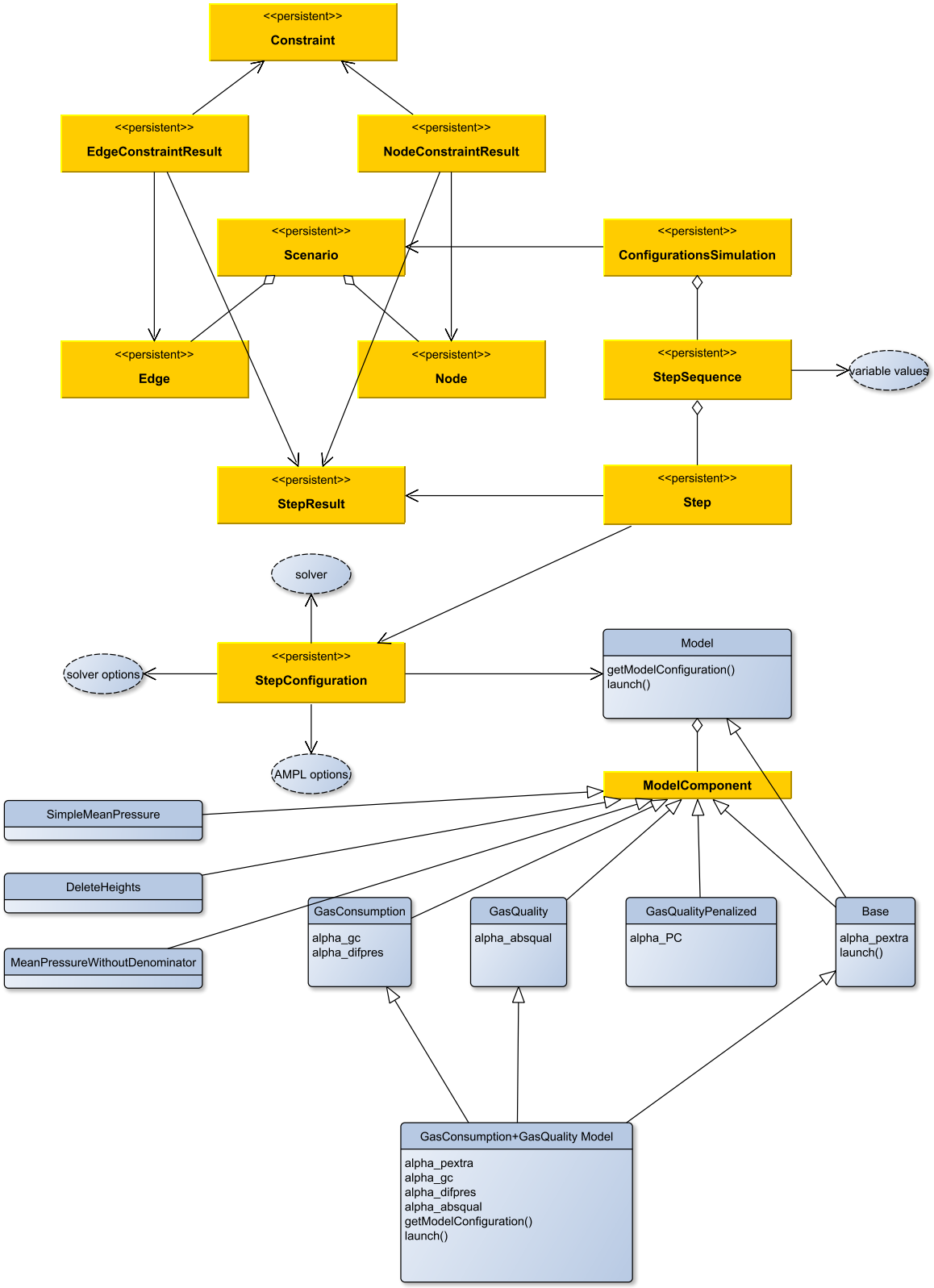

Figure 10: Simulations framework class diagram

```
actual_model_of_interest = (base_model_component + gas_consumption_component +
                            gas_quality_component)
```

Thereby, model component objects themselves serve as prototype objects for the generation of more complex model components. The *launch()* method of the model gets dispatched to the actual implementation in the model components. Objects of classes labeled with the ***UML*** stereotype *<<persistent>>* are stored to an *SQLite* database for further analysis. The object-relational mapping between objects in the framework and entities in the database is handled by the *sqlalchemy* Python package.

5 Results

5.1 Experimental setup

In each of the following six sections we investigate the effect of changing one particular configuration parameter and show the resulting outcomes. More precisely, we usually fix the values of the remaining configuration parameters - that is, the configuration parameters that we do not change in the respective section - to the best performing configuration settings of the previous section, as testing all possible combinations of configuration settings is infeasible.

Thus, per section we will describe the results of one particular test battery. We created test batteries focusing on:

- Model sequences
- *AMPL* presolve option
- Solver options
- Simplified and penalized model sequences
- Alpha parameters
- Solver ensembles

In the Discussion section we will summarize the results of all sections.

All parameters of the models were set to their default values (see table 2), if not specified otherwise.

A configuration has a complex structure, consisting of an arbitrary number of steps with arbitrary configuration settings per step. Therefore, it is not a trivial task to extract meaningful single independent factors to predict the outcomes.. For example, in each step the solver could be changed, so that a factor "solver" on its own would not necessarily be helpful. Hence, we mostly restrict our analysis to complete configurations as factors.

We created 259 different scenarios for our simulations. These scenarios were generated computationally with random elements and are based on the example network in the section Example gas network scenario. The scenarios are the connected components which resulted from random Erdos-Rényi graphs with 20 nodes and with a probability of an edge between two nodes of 6% (Erdös and Rényi (1959)). The attributes of the nodes and edges, for example whether a node is a supply point or a consumption point, were also randomly set, usually according to normal and uniform distributions, while paying attention to keeping the scenarios solvable. Nevertheless, we also created - by chance - some scenarios that could not be solved by any configuration and that might not have a valid solution at all.

The maximum number of nodes of a scenario is 18 and the maximum number of edges of a scenario is 21. Figure 11 shows the joint frequency distribution of nodes and edges of the generated scenarios.

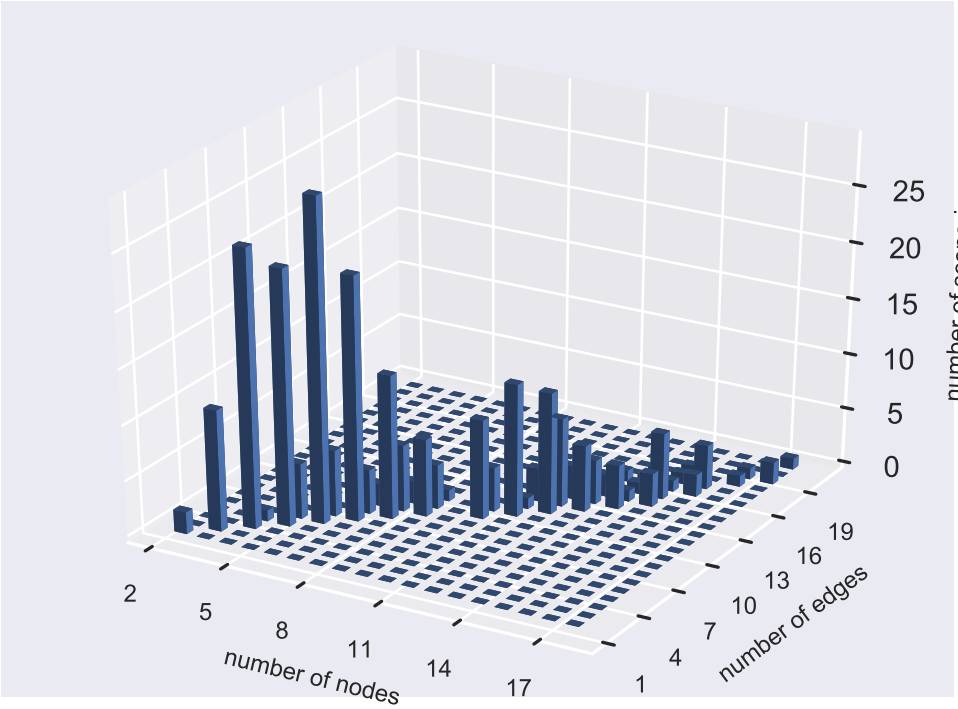

Figure 11: Joint Frequency distribution of nodes and edges of the generated scenarios

To make the results of different configurations comparable, we launched each tested configuration on every generated scenario so that we are dealing with paired data.

We compared the performance of four different solvers: *BARON* (Sahinidis (1996)), *Ipopt* (Wächter and Biegler (2006)), *Knitro* (Byrd, Nocedal, and Waltz (2006)) and *MINOS* (Murtagh and Saunders (2003)). If not specified otherwise, we launch all solvers with their default solver options. As *BARON* searches optimization problems to global optimality, this solver tends to take more time than would be practicable for our experiments. Thus, we start *BARON* with a timeout of one minute.

We conducted our experiments on a computer with an Intel Core i5-7200 processor and 8 GB of RAM.

5.2 Model sequences

We begin the presentation of our results with a test battery consisting of configurations with different models in each step. In each step of the same configuration the same solver was used. To allow comparisons with simpler configurations we also included configurations consisting of one single step. Moreover, we included replications of some configurations to verify if and in how far our results change when running the same test configurations multiple times on the same scenarios. The results of the replications were obtained by launching a second test battery.

In detail, we investigated the following model sequences (see section Model simplifications and extensions) :

1. one model sequence containing only one model, which of course has to be the *gas quality* model, as this is the model we are actually aiming to solve and therefore this model is always the last model launched (see section 3.1). This model includes *base*, *gas consumption* and *gas quality* components and is the actual model of interest. In the figures we abbreviate this model sequence by **gq**.
2. one model sequence with three different models, with the model used in each step being successively more difficult, that is the *base* model in the first step, the *gas consumption* model in the second step and the *gas quality* model in the third step. The *gas consumption* model includes *base* and *gas consumption* components. We abbreviate this model sequence by **base gc gq**.
3. To check whether possible effects of the model sequence described under 2. were due to the model sequence or - instead - due to the mere fact of launching multiple steps, we created a control group. This control group was composed of a model sequence with three models in which every model is the same model - the actual model of interest. The abbreviation for this model is **gq gq gq**.

We launched every model sequence with every solver available (*BARON*, *Ipopt*, *Knitro* and *MINOS*) .

The result of a configuration is equivalent to the result of the last step of the step sequence of the configuration, with the exception of the total time, which is the sum of the times of all steps (see section 3.1).

In figures 12 until 15 we show the results per configuration. As we launched each configuration with a specific solver, to compose the labels of configurations in the graphics we prefix the name of the the model sequence with the name of the solver used. We launched the single step model sequences described under 1. a second time on the same scenarios and suffixed these configurations with **replicate**. We will preserve these naming conventions throughout this chapter. Other configuration parameters not described here were set to their default values. In the figures we have sorted the configurations by solver name and by model sequence name.

In this and in the following sections we usually show the same following diagrams:

- a feasibility diagram, in which we show the number of solved scenarios per configuration. We considered a scenario to be solved if the minimum slack of all constraints was greater than -0.1. This threshold was the lowest value for which a solver reported “solved” for all our test batteries.
- a quality diagram, which demonstrates the quality of the objective function value returned. The quality is measured relatively to the lowest objective function value of all configurations that have solved the scenario. solved.
- a boxplot diagram indicating the distribution of the minimum slack per configuration.
- a boxplot diagram indicating the distribution of the sum of the computation times of all steps per configuration. The times are given as reported by *AMPL*.

  In the last three diagrams mentioned we only consider configurations that have solved a minimum number of 10 scenarios. In the quality diagram and in the slack diagram only scenarios that all listed configurations have solved are taken into account.

  The whiskers in the boxplots are drawn at a distance of 1.5 times the difference between the third and the first quartile from the edge of the box. Points outside the whiskers depict outliers. As outliers may be widely spread, the diagrams with the boxplots do not necessarily show all outliers.

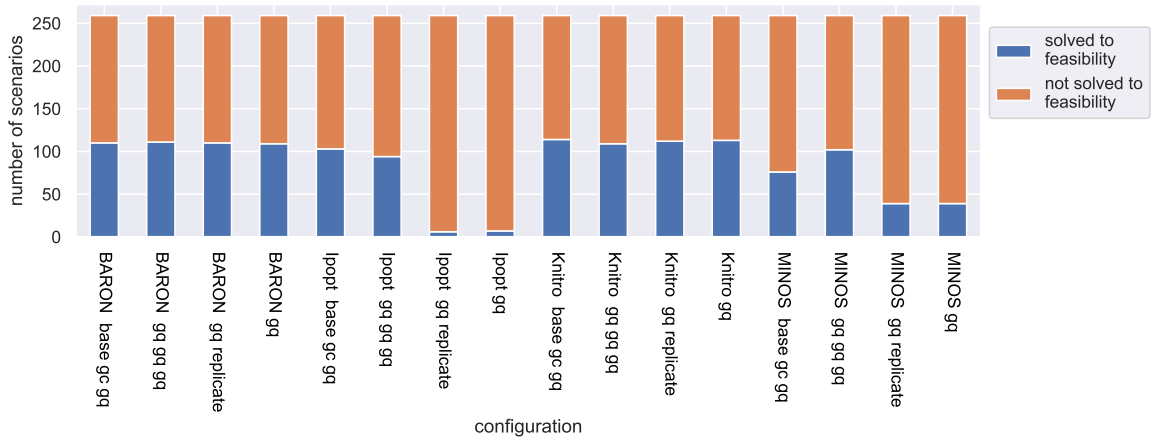

Figure 12: Number of scenarios solved to feasibility by model sequence configuration

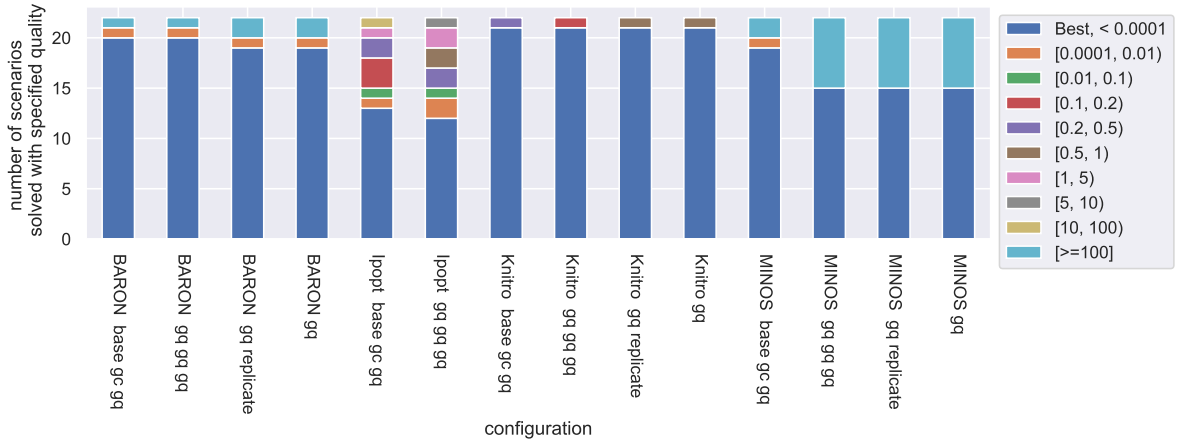

Figure 13: Quality of objective function by model sequence configuration relative to best performing configuration. Only scenarios that were solved by all of the listed configurations were considered.

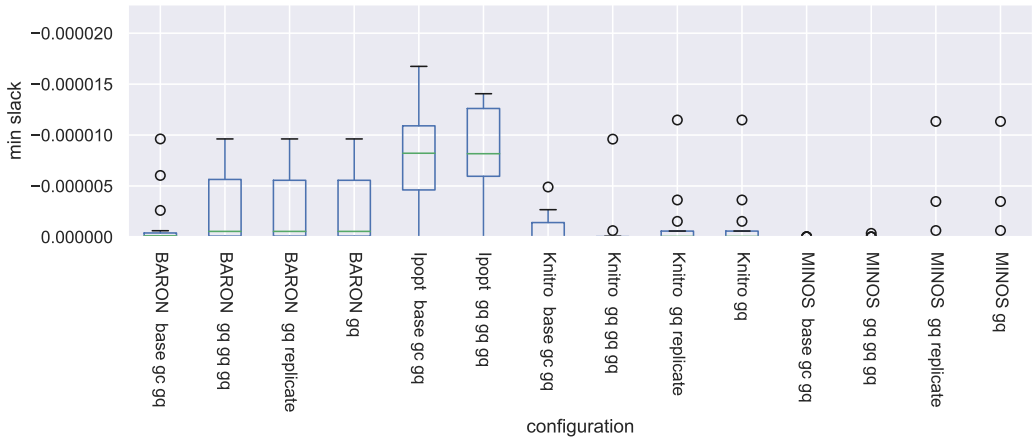

Figure 14: Boxplots of minimum slack by model sequence configuration. Only scenarios that were solved by all of the listed configurations were considered.

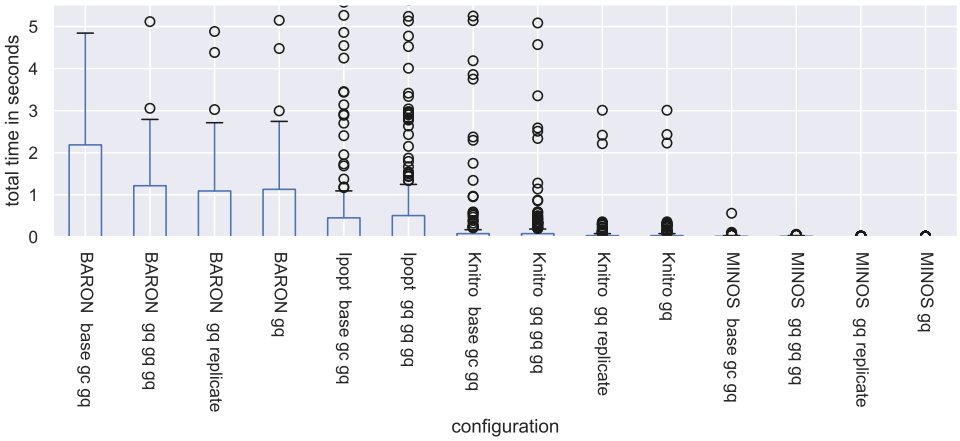

Figure 15: Boxplots of sum of times of all steps of sequence by model sequence configuration

The solver *Knitro* in conjunction with the model sequence *base model - gas consumption model - gas quality model* solved most scenarios - 114, as can be seen in figure 12. *Knitro* also achieved the highest quality of the objective function value (figure 13) while *MINOS* in general reported solutions with the lowest amount of slack (figure 14) and took the shortest time to solve (figure 15).

In the replicate runs the numbers of solved scenarios slightly changed. Thus, when interpreting our results we have to take into account that our results are not deterministic, but instead include a certain degree of uncertainty. On the other hand, the number of scenarios solved in the replicate runs only differs by at most one. Therefore, it seems that at least for this test battery the differences between configurations are much greater than between replications of the same configuration.

Accordingly, we observed differences between configurations that do appear to be systematic, for example, *Ipopt* and *MINOS* with their default options perform much better if run in sequence.

The only solver that solved scenarios that no other solver solved was MINOS. That is to say, every minimal configuration combination contains a configuration with the solver *MINOS*.[2]

5.3 *AMPL* presolve option

In this section we investigate configurations launched with the *AMPL* presolve option set to *false*, i.e. *without* the presolve phase. We chose the same configurations as in the previous section but added configurations that only differed in the *AMPL* presolve option. As stated in (Fourer, Gay, and Kernighan 1989, 275), "AMPL's presolve phase attempts to simplify a problem instance after it has been generated but before it is sent to a solver". Per default *AMPL* carries out a presolve phase, as happened in the test battery of the previous section as well. However, based on previous experiences the presolve phase might affect the results, so that we studied the effect of turning it off.

In figures 16 until 19 we show the results of this test battery. Figures 17 and 18 only include configurations that solved at least 10 scenarios.

[2]There were 14 different minimal configuration combinations, which we do not list here. In following sections with a smaller number of minimal configuration combinations we will show the respective minimal configuration combinations.

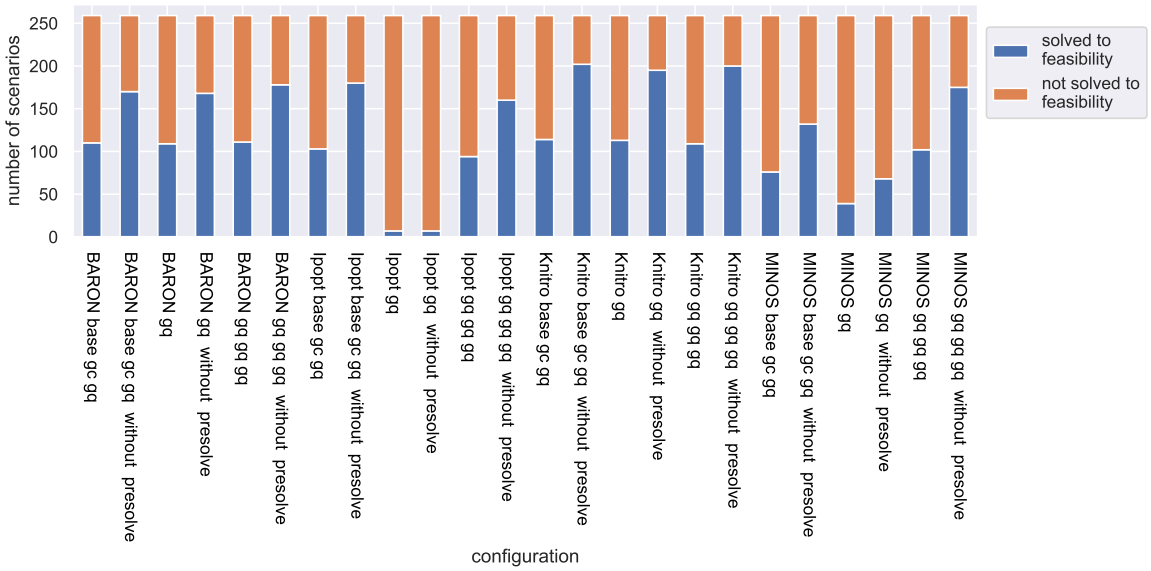

Figure 16: Number of scenarios solved to feasibility by AMPL presolve option configuration

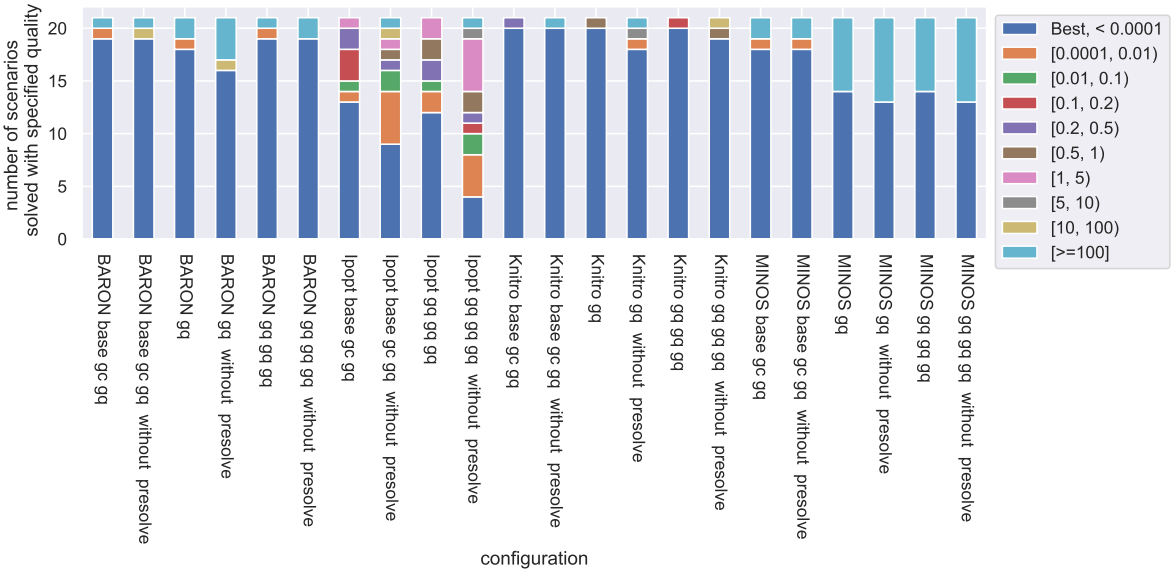

Figure 17: Quality of objective function by AMPL presolve option configuration relative to best performing configuration. Only scenarios that were solved by all of the listed configurations were considered.

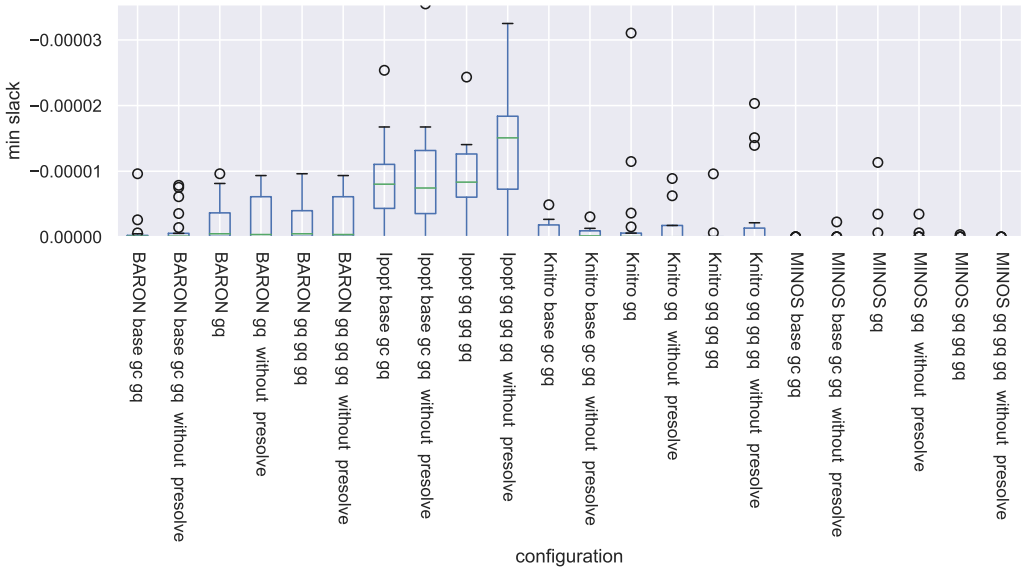

Figure 18: Boxplots of minimum slack by AMPL presolve option configuration. Only scenarios that were solved by all of the listed configurations were considered.

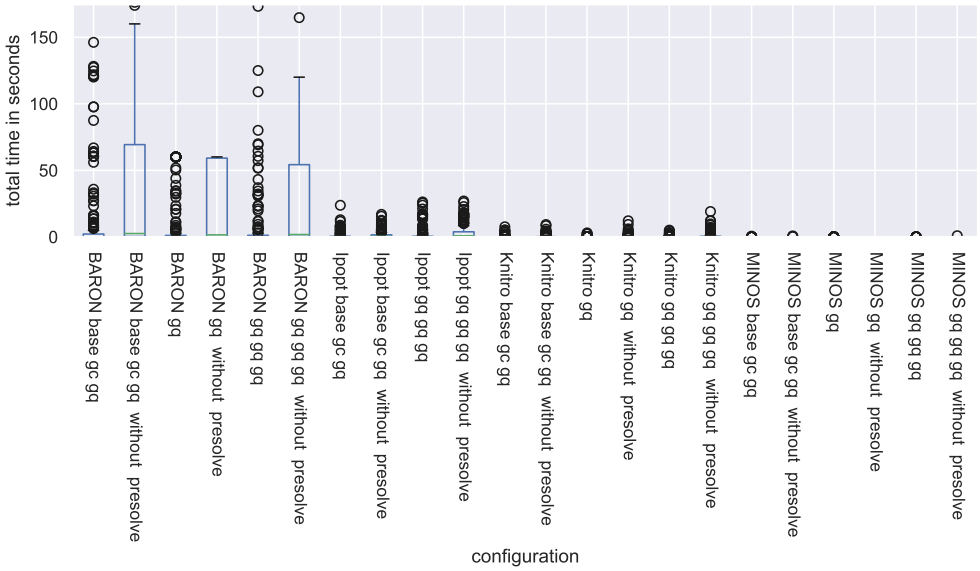

Figure 19: Boxplots of sum of times of all steps of sequence by AMPL presolve option configuration

The number of solved scenarios of two configurations that only differ in the presolve option was always higher for the configuration with the presolve option set to *false*. Furthermore, the performance of the configurations in respect to the objective function value was also better without a presolve phase. Thus, for the test batteries that we will describe in the following sections we always set the presolve option to *false*.

Like in the previous section, the model sequence *base model* - *gas consumption model* - *gas quality model* with the solver *Knitro* solved most scenarios, albeit this time with the presolve option set to false. This configuration without the presolve phase solved 202 scenarios, whereas the same configuration with a presolve phase solved 114 scenarios.

5.4 Solver options

In addition to the *AMPL* presolve option in the last section we also tested specific solver options for the solvers *Knitro* and *Ipopt* based on previous experience. The results are shown in figures 20 until 23

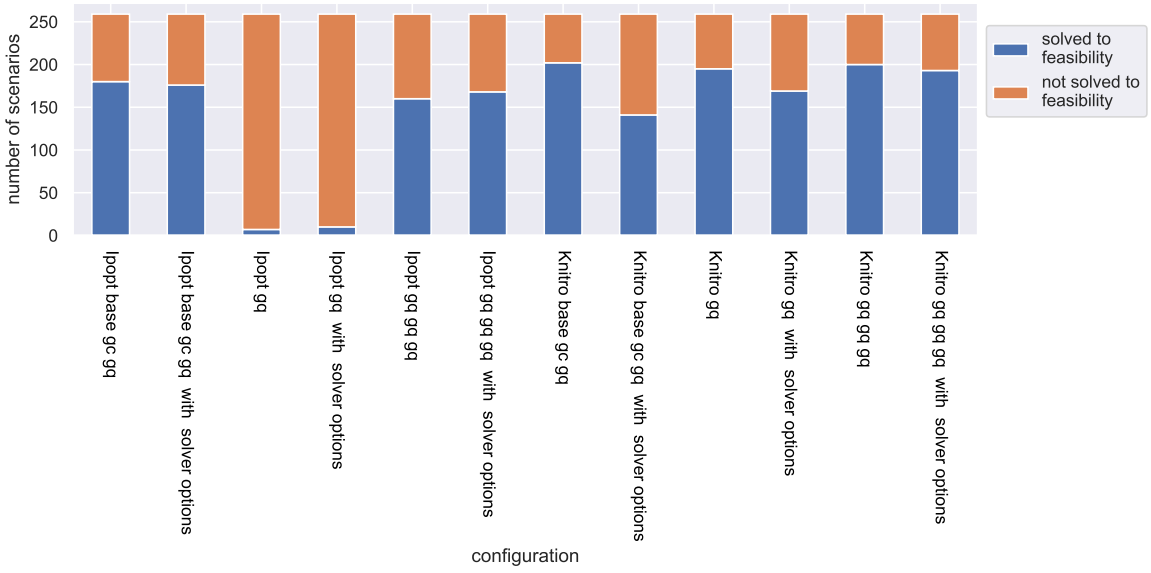

Figure 20: Number of scenarios solved to feasibility by solver options configuration

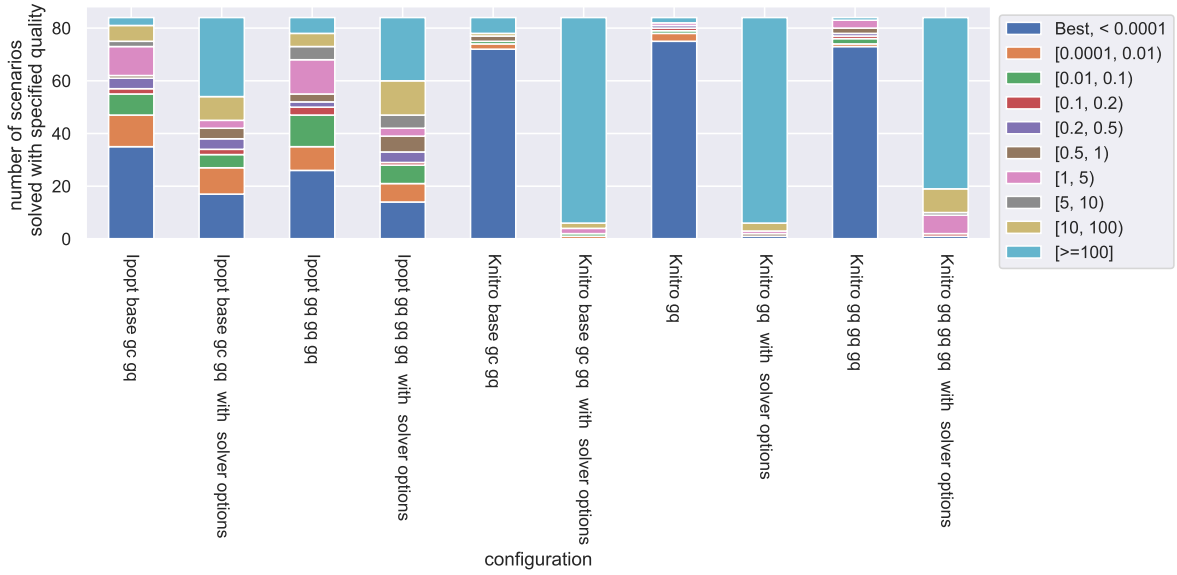

Figure 21: Quality of objective function by solver options configuration relative to best performing configuration. Only scenarios that were solved by all of the listed configurations were considered.

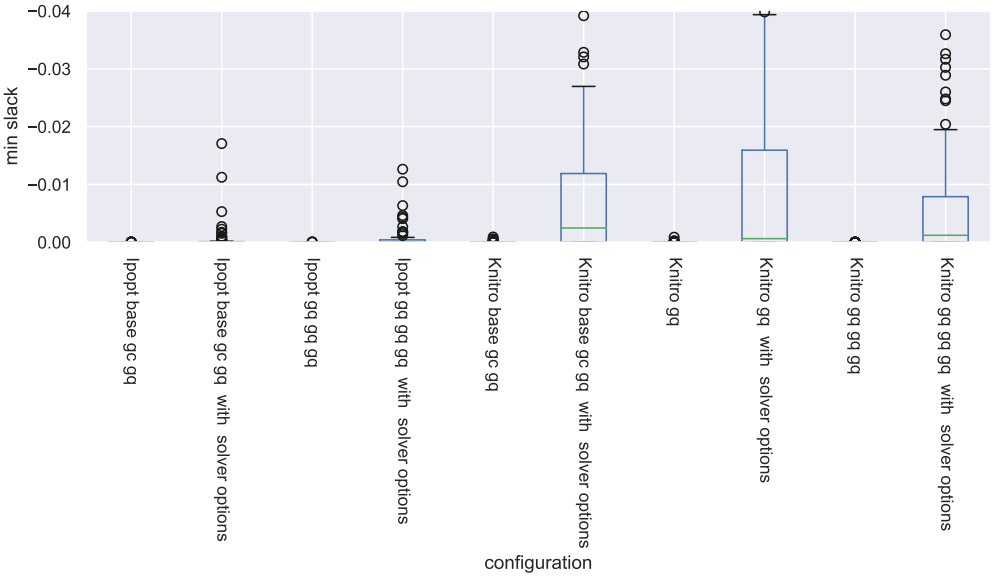

Figure 22: Boxplots of minimum slack by solver options configuration. Only scenarios that were solved by all of the listed configurations were considered.

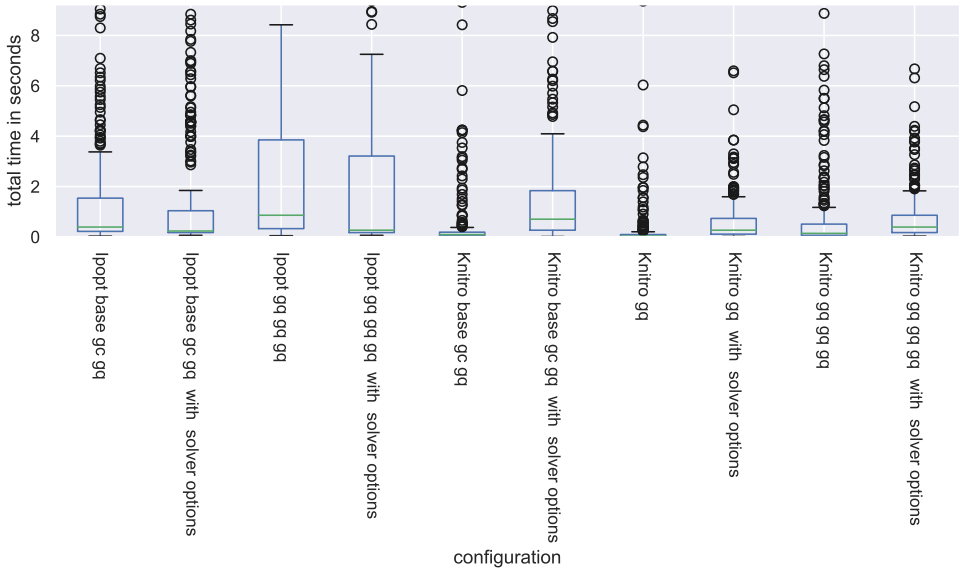

Figure 23: Boxplots of sum of times of all steps of sequence by solver options configuration

As can be seen from figures 20 to 23, the specific solver options did not in general improve the performance of the solvers *Ipopt* and *Knitro*. In particular, the quality of the solutions worsened if compared with the respective configuration without solver options. In the following sections we will launch all solvers with their default options again.

It remains to be said, that the only minimal configuration combination of the test battery of this section consists of the three configurations **Ipopt gq gq gq with solver options**, **Ipopt base gc gq with solver options** and **Knitro base gc gq**. Hence, neither only configurations with *Knitro* as solver nor only configurations with *Ipopt* as solver solved all scenarios that were solved with any configuration of this test battery. Interestingly, although the configuration **Ipopt base gc gq** solved most scenarios of all configurations with the solver *Ipopt*, it is not part of the minimal configuration combination.

5.5 Simplified and penalized model sequences

Having analyzed different model sequences in the previous sections without detecting a general best-performing model sequence, we continued with testing model sequences with larger differences between the different models of the sequence. More precisely, we launched models with simplifications and penalizations (see section Model simplifications and extensions):

- The first model of the model sequence **smp base gq** is a model that consists of the *base* component simplified by the *simple mean pressure* component (see section Model simplifications and extensions). The names of the next two models in the sequence follow our usual naming convention, i.e. they consist of the *base* model component and the *gas quality* component, respectively.
- The model sequence **gqp gqp gq** contains a penalized model as the first and second model in the sequence, consisting of the actual model of interest with the addition of the **gas quality penalized** component.

Also, we included the configurations of section 5.3 without the *AMPL* presolve phase as control groups.

In figures 24 until 27 we show the results of this test battery. Figures 25 and 26 only include configurations that solved at least 10 scenarios.

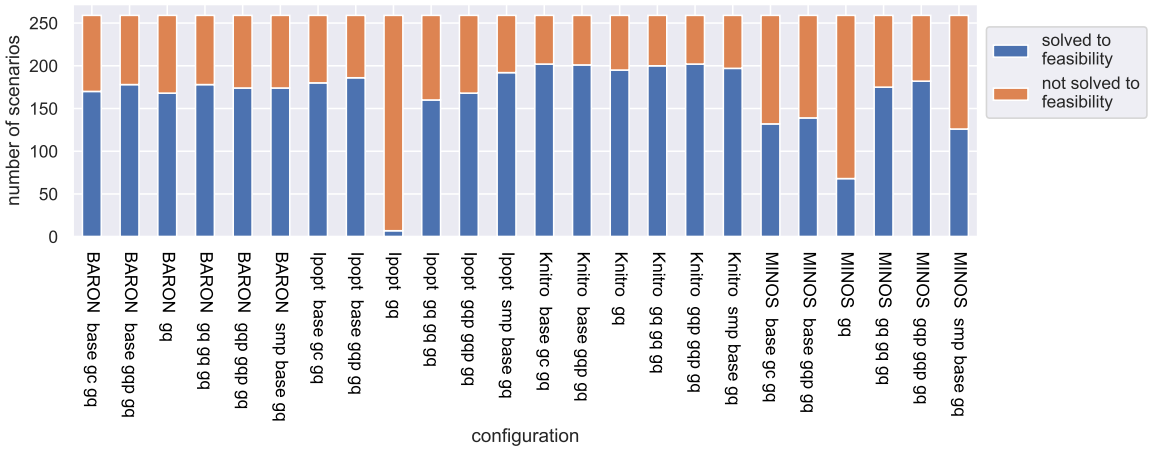

Figure 24: Number of scenarios solved to feasibility by simplifications and penalizations configuration

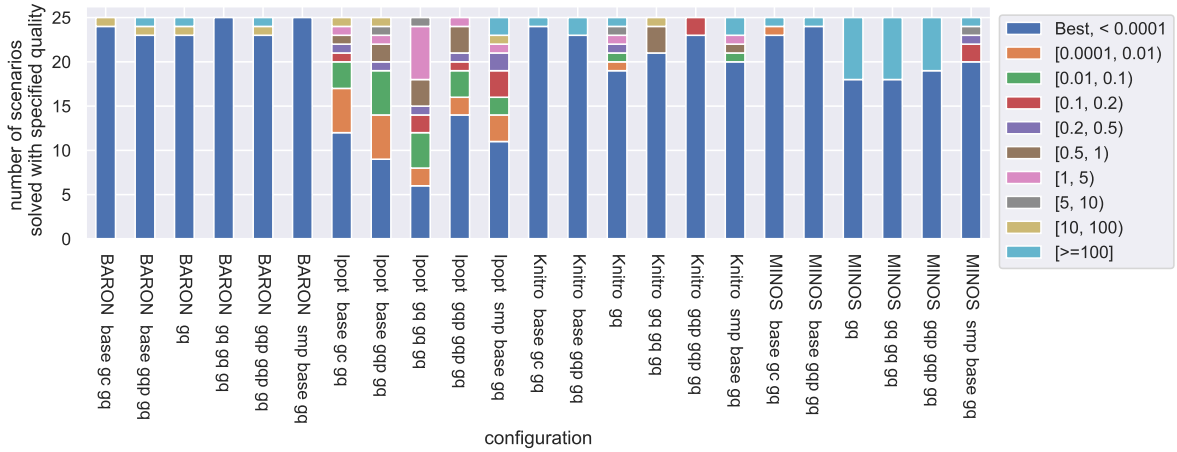

Figure 25: Quality of objective function by simplifications and penalizations configuration relative to best performing configuration. Only scenarios that were solved by all of the listed configurations were considered.

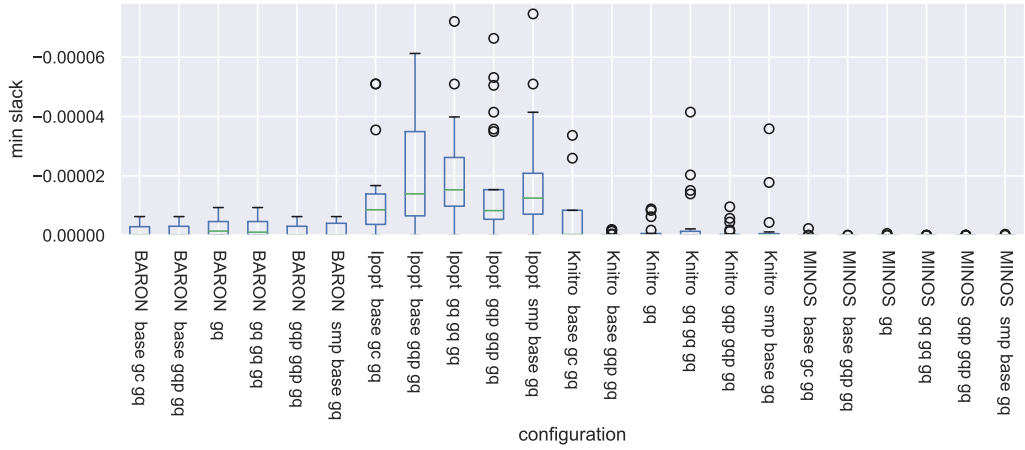

Figure 26: Boxplots of minimum slack by simplifications and penalizations configuration. Only scenarios that were solved by all of the listed configurations were considered.

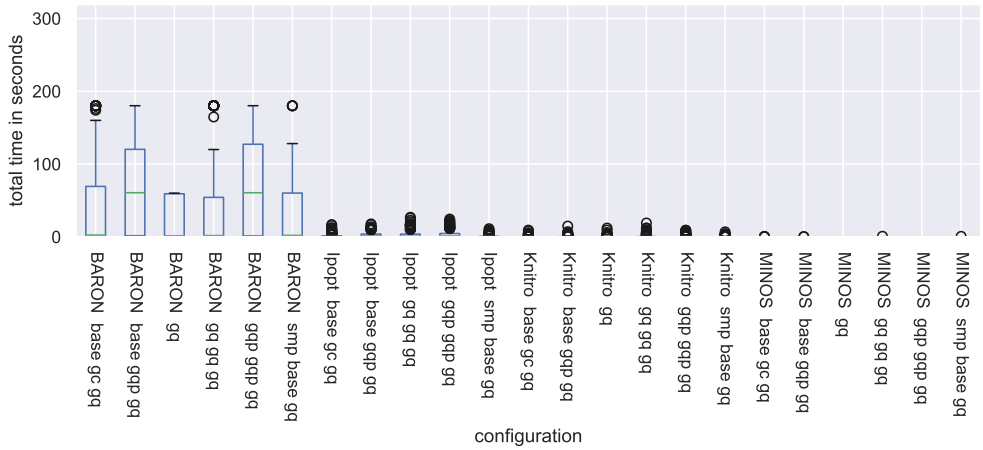

Figure 27: Boxplots of sum of times of all steps of sequence by simplifications and penalizations configuration

As long as two configurations were of the same length, we could not detect a particular step sequence that performs best for all solvers, which includes the penalized and simplified model sequences.

In other respects, the results of this section confirm what we already observed in the previous sections. For example in general *Knitro* solved most scenarios and the objective function value of the solved scenarios using *Ipopt* is often higher and the objective function value of the solved scenarios using *Baron* is usually lower than the objective function value of the other solvers tested. *MINOS* excels regarding slack and computation time.

5.6 Alpha parameters

In this section we investigate different alpha parameters of the models. To this aim we selected the best performing configurations per solver of the previous section but launched them with different alpha parameters. While in the previous sections we always chose default values for the alpha parameters, for the test battery of this section we multiplied the default alpha parameters by discrete powers of 10.

Specifically, we parametrized each model with the following alpha parameter values:

- For each alpha parameter and each selected configuration we created two configurations in which we set the alpha parameter to 10 times and 0.1 times its default value (see table 2, respectively). For example, the configuration **Knitro base gc gq alpha gc*10** denotes the configuration with the solver Knitro and the model sequence *base* - *gas consumption* - *gas quality* and the parameter alpha_gc set to 10 times its default value (albeit that the parameter alpha_gc does not affect the first **base** model)
- Also, for each configuration we created two configurations in which we set **all** alpha parameters simultaneously to 10 times and 0.1 times their default values, respectively, for example denoted by **Knitro gq gq gq all alphas*10**.

We did not change the alpha parameters in the last step as that would change the objective function value and make comparisons with other test batteries difficult.

Figures 28 until 31 show the results for the different alpha parameter values.

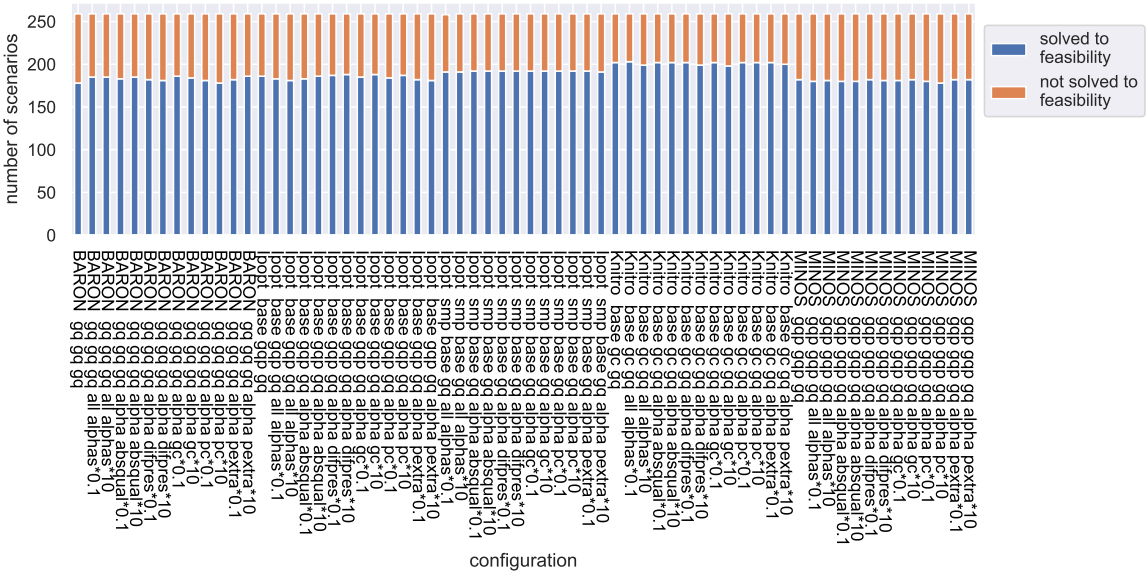

Figure 28: Number of scenarios solved to feasibility by alpha parameters configuration

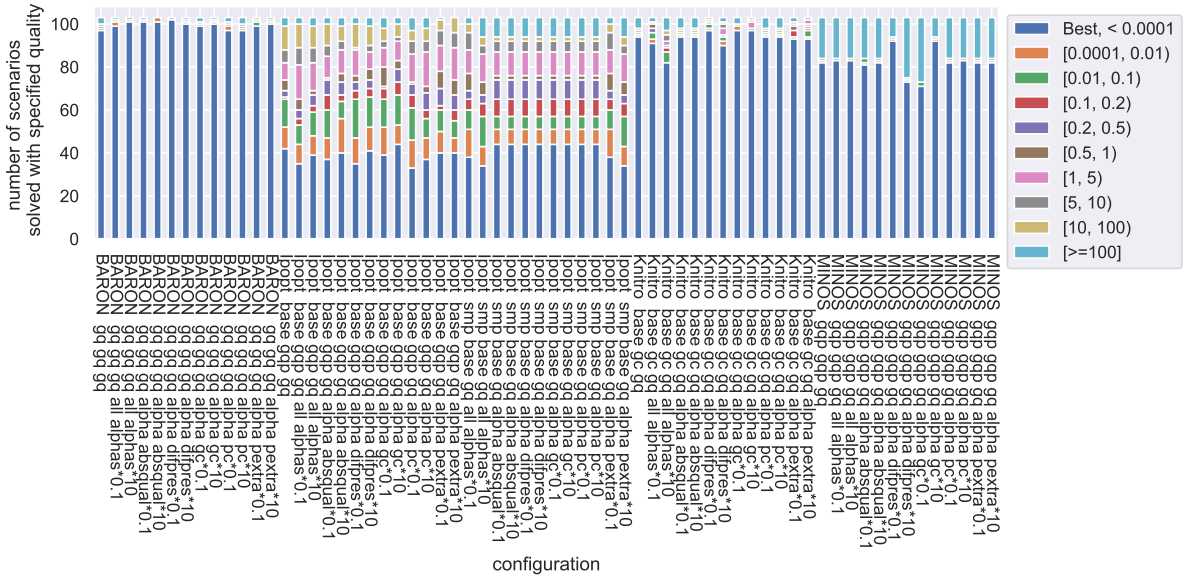

Figure 29: Quality of objective function by alpha parameters configuration relative to best performing configuration. Only scenarios that were solved by all of the listed configurations were considered.

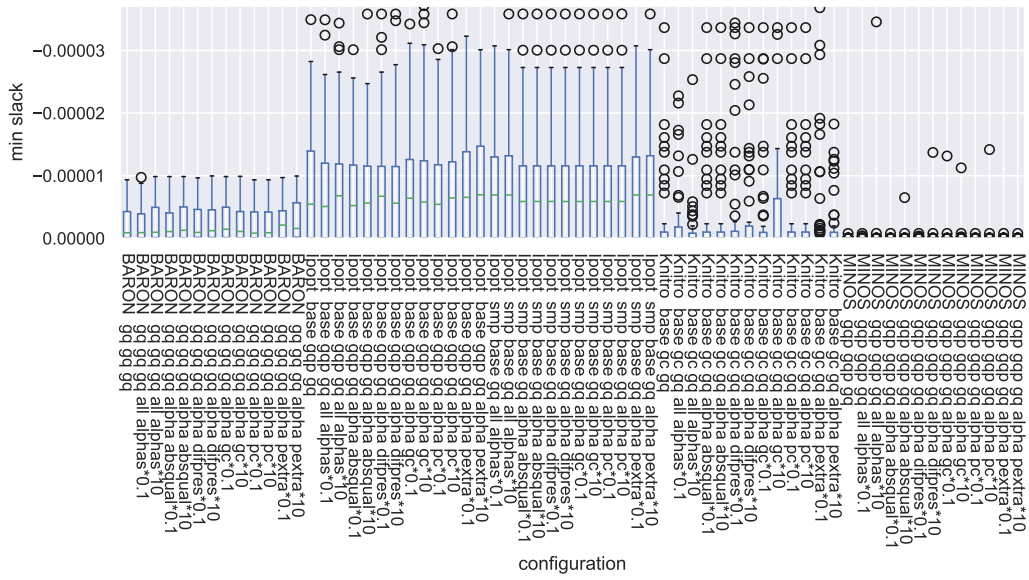

Figure 30: Boxplots of minimum slack by alpha parameters configuration. Only scenarios that were solved by all of the listed configurations were considered.

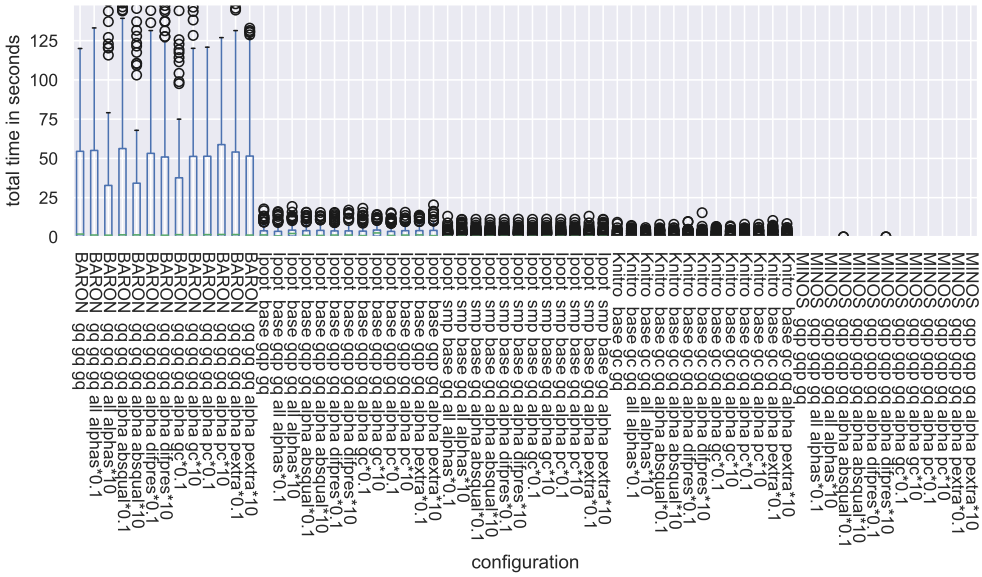

Figure 31: Boxplots of sum of times of all steps of sequence by alpha parameters configuration

All in all the effects on the performances of the selected configurations when changing alpha parameter values appear to be rather small in comparison to other parameters like the employed solvers.

Also, as we had shown in the section Model sequences the results contain a certain degree of randomness, so that a difference of one more solved scenario might be due to chance.

Clearly, *Knitro* combined with the model sequence *base - gas consumption - gas quality* solved most scenarios (up to 203), followed by *Ipopt* with the simplified model sequence (up to 192). In fact, the number of solved scenarios for all configurations with *Knitro* was always higher than the number of solved scenarios of configurations with any other solver.

Figure 32 shows scenarios grouped by their number of nodes that were solved by four selected configurations.

From the graphic in figure 32 we could not detect clear evidence that a certain configuration might be more suited for certain scenarios dependent on the number of nodes. Nevertheless, it comes as a surprise that the configuration **MINOS gqp gqp gq**, a configuration with relatively few solved scenarios, was the only configuration that was able to solve three scenarios with the highest number of nodes (18).

Summing up the results of this section, it can be said that the solvers and step sequences seem to have a much larger impact than the alpha parameters - at least in respect to the parameter values we restricted ourselves to.

5.7 Solver ensembles

In the last test battery we investigated effects of **solver ensembles**. By "solver ensembles" we mean step sequences with different solvers in different steps, where as usual the solution of the previous step serves as initial solution for the next step.

Again, we selected well-performing configurations of previous sections per solver, this time including the best

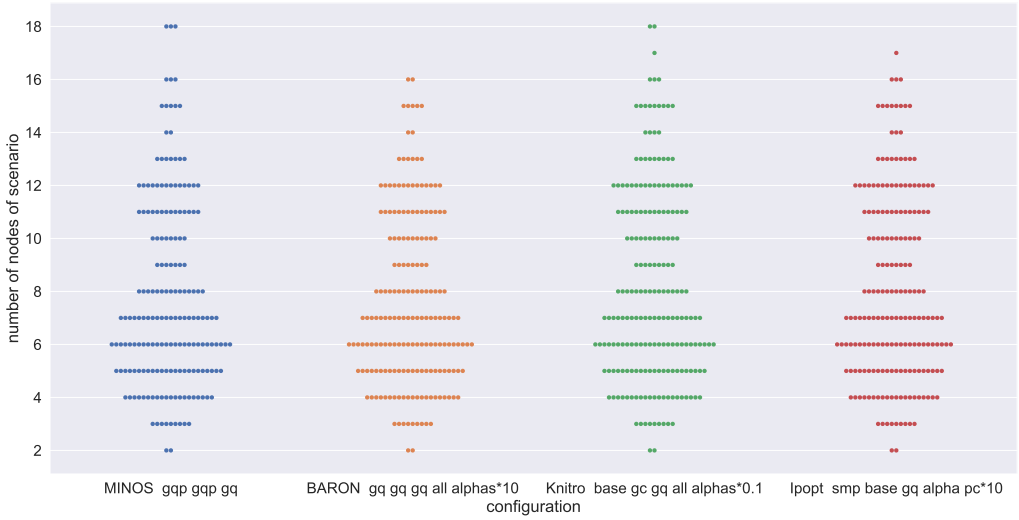

Figure 32: Solved scenarios by number of nodes and configuration

alpha parameter values of the configurations as retrieved in the previous section. We launched the selected configurations in sequence, for example first the best sequence with *Knitro*, then the best sequence with *BARON* - started with the initial value given by the preceding sequence with *Knitro* - and so forth.

We tested different orders of the selected configuration sequences with each sequence being once at every position in the total sequence. For example, the configuration **Knitro base gc gq MINOS gqp gqp gq BARON gq gq gq Ipopt smp base gq** consists of twelve steps in total, of which the first three steps belong to the selected step sequence for the solver *Knitro*, comprising the model sequence *base gc gq*.

To measure the effect of these extensive configurations we compare these configurations with two different other types of configurations:

- configurations that launch the actual model of interest directly with each solver in a single step.
- configurations that launch the actual model of interest directly with each solver in a single step, yet also concatenating the different configurations in sequence, resulting in four steps per configuration in total.

In figures 33 to 36 we present the results of the different ensembles.

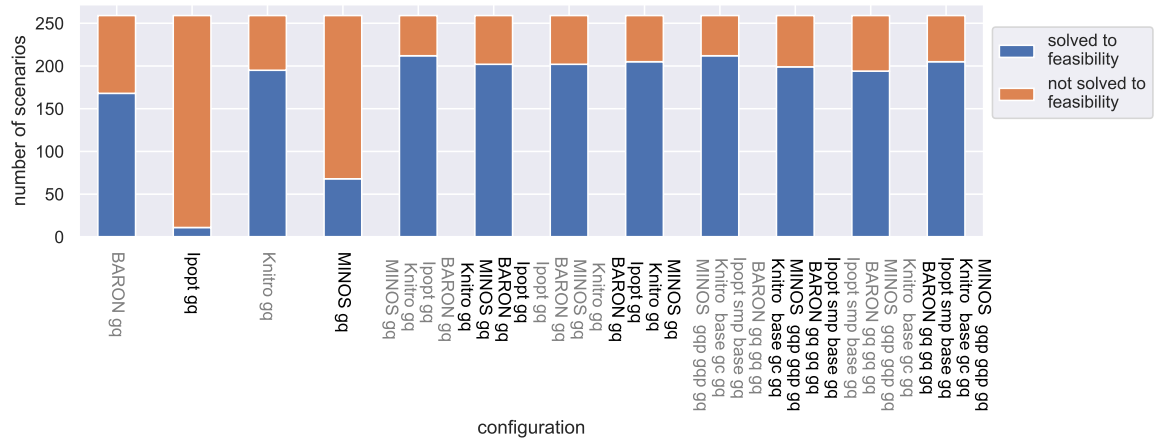

Figure 33: Number of scenarios solved to feasibility by solver ensembles configuration

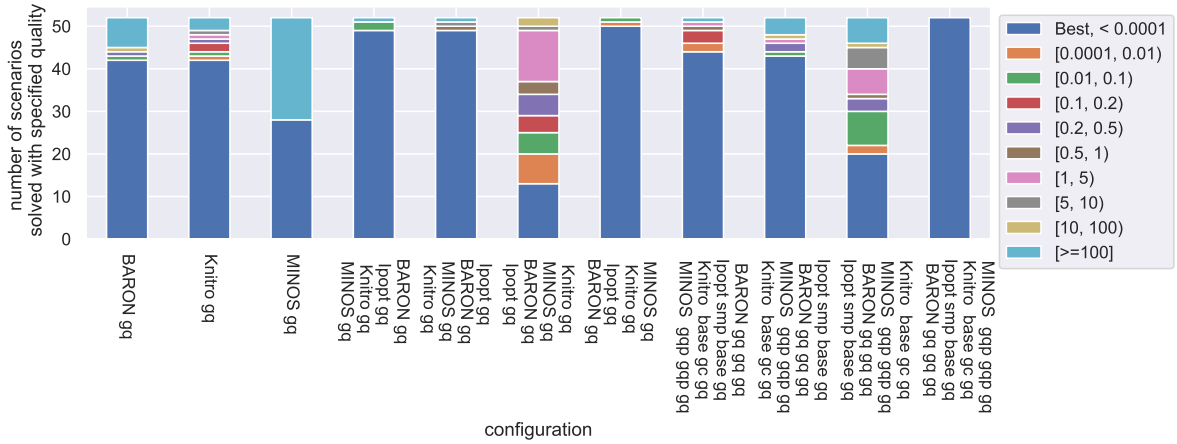

Figure 34: Quality of objective function by solver ensembles configuration relative to best performing configuration. Only scenarios that were solved by all of the listed configurations were considered.

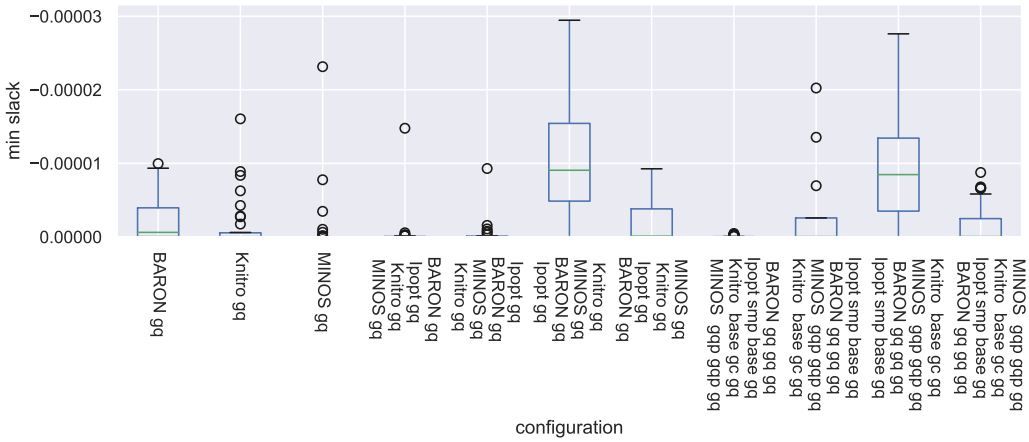

Figure 35: Boxplots of minimum slack by solver ensembles configuration. Only scenarios that were solved by all of the listed configurations were considered.

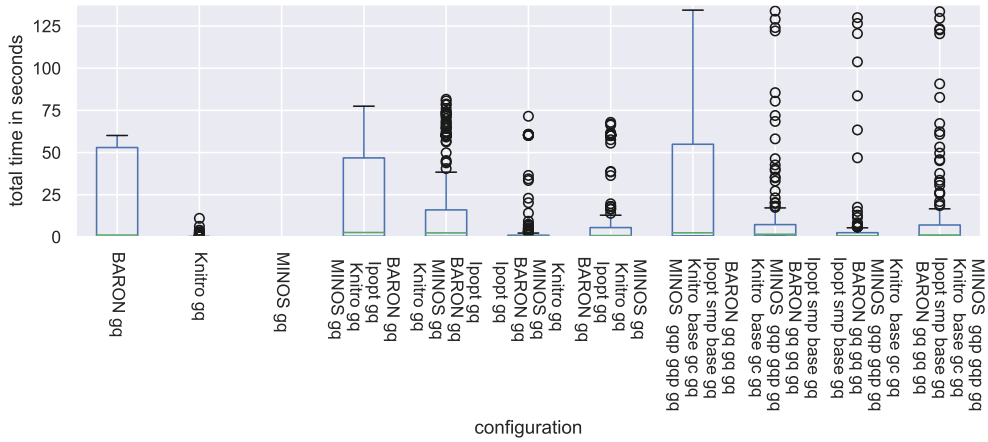

Figure 36: Boxplots of sum of times of all steps of sequence by solver ensembles configuration

As we can see in figures 33 to 36, the two configurations with ensembles of the solvers in the order *BARON-Kitro-Ipopt-MINOS* solved most scenarios (212), regardless of whether they were run individually in step sequences or in a single step. Moreover, each of these configurations represents a minimal configuration combination. No other combination of configurations of this test battery constitutes a minimal configuration combination. The second highest number of scenarios solved by a configuration was 205, which was reached by both ensemble configurations with the solver order *MINOS-Knitro-Ipopt-BARON*.

Thus, launching sequences with different solvers in different steps yielded better results in respect to the number of scenarios solved than running the solvers individually and combining the results. However, the length of the sequences did not seem to have an effect: directly launching the solver configurations with the actual model of interest in sequence, we obtain results which appear to be in general just as good as launching step sequences with multiple steps for each solver in sequence.

The order of the ensembles also plays an important role in the quality of the objective function value: if *BARON* with its default configuration is the last solver in the sequence, the objective function value is generally lower, and if *Ipopt* with its default configuration is the last solver in the sequence, the objective function value is generally higher.

Finally, we examined whether the differences we observed between the proportion of solved scenarios per configuration are statistically significant for the distribution behind the randomly generated scenarios. To this aim we compared two well-performing configurations of the test battery of this section:

- **Knitro gq**, the best performing configuration of a single solver in a single step with 195 solved scenarios, and
- **BARON gq Ipopt gq Knitro gq MINOS gq**, the best performing configuration of all test batteries with 212 solved scenarios.

The contingency table of these two configurations in respect to the number of solved scenarios is as follows:

Table 7: Contingency table of solved scenarios for best ensemble configuration and best single-solver configuration

	BARON gq Ipopt gq Knitro gq MINOS gq	
	not solved	**solved**
Knitro gq		
not solved	47	17
solved	0	195

To this table we apply the McNemar test to check for differences between the proportions of individuals of two populations that verify a certain characteristic given paired samples (Cao Abad et al. (2001)). As a result, the McNemar test with the Edward's correction yields a p-value of 0.0001042. Thus, we have observed significant differences between the proportions of solved scenarios by the two configurations at a significance level of 0.01

5.8 Discussion

Summarizing our results, solvers run in sequence in general performed better than being run in a single step, i.e. they solved more scenarios and reached a higher quality of the objective function. Solvers that in a single

step solved fewer scenarios than all single-step configurations did on average, i.e. the solvers *MINOS* and *Ipopt*, reacted more strongly to being run in sequences than other solvers (*BARON* and *Knitro*).

Launching different models per step had an effect, albeit small and hard to predict. We did not observe large systematic effects of solver options or of using simplified and penalized models or of switching alpha parameters. Turning the presolve phase off always increased the number of solved scenarios.

Configurations with the solver *Knitro* and the model sequence *base* - *gas consumption* - *gas quality* were the best performing single-solver configurations with respect to the number of solved scenarios for all test batteries. With the presolve phase turned off this configuration reached a total of 202 solved scenarios (of a total of 259 scenarios) and by multiplying all alpha parameter values by 0.1 we increased the number of solved scenarios marginally by 1 to 203.

Solver ensembles, i.e. configurations with different solvers in different steps, reached the highest number of solved scenarios (up to 212). The best solver ensembles solved all scenarios that any other configuration of all test batteries found and also solved scenarios that no other configuration of all test batteries found.

The quality of the objective function was usually best with *BARON* and worst with *Ipopt*. *MINOS* in general achieved minimum slack and time.

Also, we confirmed that we have observed statistically significant differences between the proportions of solved scenarios of different configurations for our randomly generated network scenarios.

6 Conclusion

In this thesis we have identified and analyzed possible configuration parameters for the solution process of gas network models. The configuration parameters we proposed consist in sequences of steps where the solution of a previous step serves as initial solution for the succeeding step. In each step a combination of solver, gas network model, penalization parameters (alphas), *AMPL* options and solver options forms the step configuration.

To compare performances of different configurations we generated random gas network scenarios and solved these scenarios computationally with different configurations.

While the solvers *Ipopt* and *MINOS* with their standard options profit from being run in sequence, we could not detect a systematic effect of different simplifications and extensions of gas network models on the final solution. Setting the *AMPL presolve* option to *false* allowed for more scenarios to be solved.

Configurations that consist of different solvers in different steps (solver ensembles) yielded the highest number of solved scenarios and could solve all scenarios that were solved by any configuration we tested. Thus, in general, we recommend to use solver ensembles to solve gas network models.

Different solvers with their standard options excel at different tasks under equal conditions: the solver *Knitro* could in general solve more scenarios than all other solvers (except for solver ensembles), *BARON* (with a timeout of one minute) usually achieved the lowest objective function values and *MINOS* returned solutions with the lowest amount of time and slack.

Due to the manifold of different configurations and scenarios we could not analyze all possible configuration parameters and scenarios. In particular, it might well be possible that adapted individual solver options might lead to improved results. Moreover, the distribution which we used to generate network scenarios might not be representative for real gas networks, although it was based on experience with existing gas networks.

Also, in this master's thesis we restricted ourselves to *mean* performances of solver configurations for *arbitrary* scenarios. An interesting starting point for future research would be an analysis of performances of solver configurations for *individual* scenarios. That would imply identifying meaningful features of scenarios and predicting the best configuration for a given feature combination, i.e., for a given scenario. Simple features of a scenario might for example be the number of nodes or the number of compressor stations. This task could be approached by machine learning algorithms, if sufficient data is available (Bishop (2006)). Furthermore, such approaches could individually weigh the importance of different objectives, for example one could introduce weights for the time needed or for the objective function value of the model under investigation. The findings in this master's thesis might prove beneficial for such further research, e.g. the configurations and objectives we identified and the effects we already pointed out might be applicable to individual scenarios and objectives as well.

Appendix

Definitions

This section is intended to serve as a quick lookup of relevant terms used throughout this thesis. For the purpose of employing a concise common language we define the main concepts in a rather informal but intuitive way as follows:

Scenario: concrete real-world application which should be optimized. In this thesis we focus on scenarios given by gas networks, made up of components like pipes, compressor stations, consumption points, entry points, etc., and modeled for example with pipes as edges and consumption and exit points as nodes (see section 2.2).

Model: mathematical formulation of an optimization problem for a given range of scenarios as input, i.e. the formulation of an objective function and restrictions. A model can be parameterized by alphas, which in general are real numbers used to calibrate terms in the model (see section 2.2).

Model component: part of a model, e.g. gas quality restrictions (see section 3.2.1).

Actual model of interest: Our goal in this thesis is to optimize a special gas network model that consists of all components specified in section 2.2, including gas consumption and gas quality components. We call this model the actual model of interest, hoping that our findings will also apply to more complicated gas network models. However, in order to find good feasible solutions for the actual model of interest we make use of more complicated as well as of simpler models, which may provide us with helpful initial solutions for the actual model of interest (see section 3.2.1).

***AMPL*:** mathematical modelling language ("AMPL - Streamlined Modeling for Real Optimization" 2018) that allows specifying models in .mod files and data of the models in .dat files. Also, models can be solved by launching .run-files and employing solvers. *AMPL* is at the same time the name for a commercial software package that supports the *AMPL* modelling language and various solvers (see section AMPL).

Solver: tool that can be used to solve a model. *AMPL* provides an interface to solvers. Solvers can be free or commercially licensed (see section 2.2).

Step configuration: combination of solver, model, alpha parameters, *AMPL* options and solver options (see section 3.1).

Step: pair of step configuration and result. The result is obtained by solving the model of the configuration with the other components specified by the configuration, i.e. the solver, alpha parameters, ampl options and solver options (see section 3.1).

Step sequence: sequence of steps, where the solution of a previous step serves as initial solution for the succeeding step. The last model of a step sequence is always the actual model of interest and the result of a step sequence is the result of the last step of the sequence (see section 3.1).

(Solver) Configuration: sequence of step configurations. A configuration belongs to a step sequence. As the models of a configuration are evaluated by solvers we will also refer to a configuration as "solver configuration". In this thesis we compare the performances of different configurations with respect to their ability to solve gas network problems to feasibility (see section Experimental setup), their objective function values, their slacks (see section Nonlinear programming) and their computation time (see section 3.1).

Configurations simulation: unit of execution of a set of step sequences for a given scenario (see section 3.3).

Solved to feasibility: We say that a configuration has solved a given scenario (to feasibility) if the sequences of the steps launched with the respective step configuration returned a feasible solution, that is, a

solution that only violates each constraint by a specified amount of slack (see section Mathematical optimization).

Minimal configuration combination (of a test battery): Let S_B be the set of scenarios solved by a test battery B. A set of of configurations C is a minimal configuration combination if every scenario $s \in S_B$ is solved by at least one of the configurations of C and if no subset of C solves every $s \in S_B$ (see section 5.2).

Solver ensembles: configurations where in different steps different solvers are used (See section 5.7).

Mathematical optimization

Nonlinear programming

As outlined in the introduction, this thesis deals with mathematical optimization problems for which no general solution methods exist that reliably return global optima of arbitrary problem instances in a reasonable amount of time.

Serving as a reference point, in the following section we briefly define what is commonly understood by terms like "optimization problem" and show some standard optimization techniques. By convention we write vectors in bold letters like $\mathbf{x}$. The terminology used mostly stems from Bazaraa, Sherali, and Shetty (2006), and from the lecture notes of a class in mathematical programming held by Julio González Díaz at the University of Santiago de Compostela in 2017.

Accordingly, González defines an **optimization problem** in a general manner as given by a pair (F, c), where F is the domain of feasible points and c is the **cost (or objective) function**. The problem consists in finding a feasible point $\mathbf{x} \in F$ such that

$$\forall \mathbf{y} \in F,\ c(\mathbf{x}) \leq c(\mathbf{y}).$$

Every point $\mathbf{x}$ that fulfills this condition is a **global optimum**.

More specifically, a **nonlinear programming problem (or program) P** consists in solving the problem P, i.e. finding an $\mathbf{x}$, such that:

$$\text{minimize } c(\mathbf{x}), \mathbf{x} \in R^n$$

subject to the inequality and equality **constraints** (or **restrictions**), respectively:

$$g_i(\mathbf{x}) \leq 0,\ \ i = 1, ..., m$$

$$h_j(\mathbf{x}) = 0,\ \ j = 1, ..., l$$

where c, $g_1, ..., g_m$, $h_1, ..., h_l$ are functions defined on R^n. In this thesis we will also refer to nonlinear optimization problems simply as optimization problems (similar to Snyman and Wilke 2018, 3). We call all vectors $\mathbf{x} \in R^n$ **solutions** of the problem P and we say that a vector $\mathbf{x}$, that satisfies all constraints of P, is a **feasible solution** of P. To give an exact definition of local and global optima we follow (Bazaraa, Sherali, and Shetty 2006, 2, 45) and quote: «If $\bar{\mathbf{x}} \in S$ and $f(\mathbf{x}) \geq f(\bar{\mathbf{x}})$ for each $\mathbf{x} \in S$, $\bar{\mathbf{x}}$ is called a [...] *global optimal solution.* [...] If $\bar{\mathbf{x}} \in S$ and if there exists an ϵ-neighborhood $N_\epsilon(\bar{\mathbf{x}})$ around $\bar{\mathbf{x}}$ such that $f(\mathbf{x}) \geq f(\bar{\mathbf{x}})$ for each $\mathbf{x} \in S \cap N_\epsilon(\bar{\mathbf{x}})$, $\bar{\mathbf{x}}$ is called a *local optimal solution.*[...] An ϵ-neighborhood [...], given a point $\mathbf{x} \in R^n$,» is defined as «the set $N_\epsilon(\mathbf{x}) = \{\mathbf{y} : ||\mathbf{y} < -\mathbf{x}|| < \epsilon\}$».

We define the **slack** of a constraint $g_i(\mathbf{x})$ as $-g_i(\mathbf{x})$ and we define the slack of a constraint $h_j(\mathbf{x})$ as $-|h_j(\mathbf{x})|$. Therefore, if the slack of a constraint is negative for a given $\mathbf{x}$, this constraint is violated and the slack shows the negative amount of the violation. For numerical reasons we will allow a small amount of infeasibility in our results. That is to say, we will still regard $\mathbf{x}$ as a feasible solution if the slacks of all constraints are greater than a specified threshold, although the constraints might not be strictly satisfied.

Furthermore, we refer to optimization problems, in which the objective c and the constraints g_i and h_j are linear functions, as **linear optimization problems**. While to solve linear optimization problems there exist polynomial-time algorithms and the well-proven non-polynomial simplex algorithm, general – nonlinear – mathematical programming is NP-hard (Hochbaum 2007), meaning that for nonlinear optimization problems no general solving algorithm with a polynomial number of computation steps is known and might not even exist.
To solve nonlinear problems without restrictions we can for example use **Newton's method** or the **conjugate gradient method** by starting from an initial solution and following in the direction of descent for a certain distance, as long as we do not leave the feasible region. We can obtain a direction of descent of a function by calculating the gradient, as the gradient points in the opposite direction of the direction of the steepest

descent (Bazaraa, Sherali, and Shetty 2006, 384).
A popular technique to solve **constrained** nonlinear optimization is the **penalty method**, which eliminates constraints by adding them as penalty functions to the objective function to obtain a series of *unconstrained* nonlinear optimization problems or *constrained* nonlinear optimization problems with constraints that are easy to satisfy. Thereby, the penalty method penalizes leaving the feasible region so that a minimum is searched for in which the penalizations or slacks are 0. Alternatively, and similar to the penalty method, there exist **barrier methods**, that add a term B to the objective function. The term B grows unboundedly when the objective function approaches the border of the feasible region, impeding that the feasible region is left (see Bronstein, Semendjajew, and Musiol 2008, 945–46).

Under certain regularity conditions the so-called **Karush-Kuhn-Tucker conditions** give us a necessary condition for the **local optimality** of a solution $\mathbf{x}$, which might be helpful in finding an optimum (Bazaraa, Sherali, and Shetty 2006, 190). In the case that the problem is convex, the Karush-Kuhn-Tucker conditions in conjunction with the regularity condition are both necessary as well as sufficient conditions for local optimality (Bazaraa, Sherali, and Shetty 2006, 195). An optimization problem is **convex** if c and all g_i are convex and all h_j are linear. As each local minimum of a convex program is also a global minimum (Bazaraa, Sherali, and Shetty 2006, 125), for convex problems the Karush-Kuhn-Tucker conditions are necessary and sufficient conditions for **global optimality** under regularity conditions.

Problems in which some variables can only take on integer values form another category of nonlinear programming problems called **integer programming**. Like nonlinear programming, integer programming, which as well as convex and linear programming is a subset of nonlinear programming, also is NP-complete (Hochbaum 2007, 290). A renowned algorithm to solve integer programming problems is for example the branch-and-bound algorithm (Land and Doig 1960).

Besides the difficulty of finding global optima, another problematic aspect in mathematical programming is that when solving difficult mathematical optimization problems in practice with the help of computers we are often facing numerical problems.

Nonlinear programming example

To illustrate some of the definitions of the previous section let us consider the following mathematical program S:

$$
\begin{aligned}
&\text{minimize } f(\mathbf{x}) = (x_1 - 3)^2 + (x_2 - 4)^2, \mathbf{x} \in R^2 \\
&\text{subject to :} \\
&g_1(\mathbf{x}) = 6x_1 + 4x_2 - 24 \leq 0, \\
&g_2(\mathbf{x}) = x_1 + 2x_2 - 6 \leq 0, \\
&g_3(\mathbf{x}) = x_1 + x_2 - 1 \leq 0, \\
&g_4(\mathbf{x}) = x_2 - 2 \leq 0.
\end{aligned}
$$

Figure 37 shows a graphical representation of the program S. The red lines embody contour lines, i.e. the value of the objective function is equal for all points $\mathbf{x} = (x_1, x_2)$ that lie on the same line. In this example the contour lines are concentric circles around the point $\mathbf{x} = (3, 4)$, the global optimum of the function f without considering the restrictions. The feasible region is colored in gray and is bordered by the functions g_1, g_2, g_3 and g_4. All points within the feasible region are feasible solutions. The arrows point in the direction of the gradient of the given functions at the points $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$. The only optimal solution of this simple example is $\mathbf{x}_3 = (2, 2)$ with an objective function value of 5.

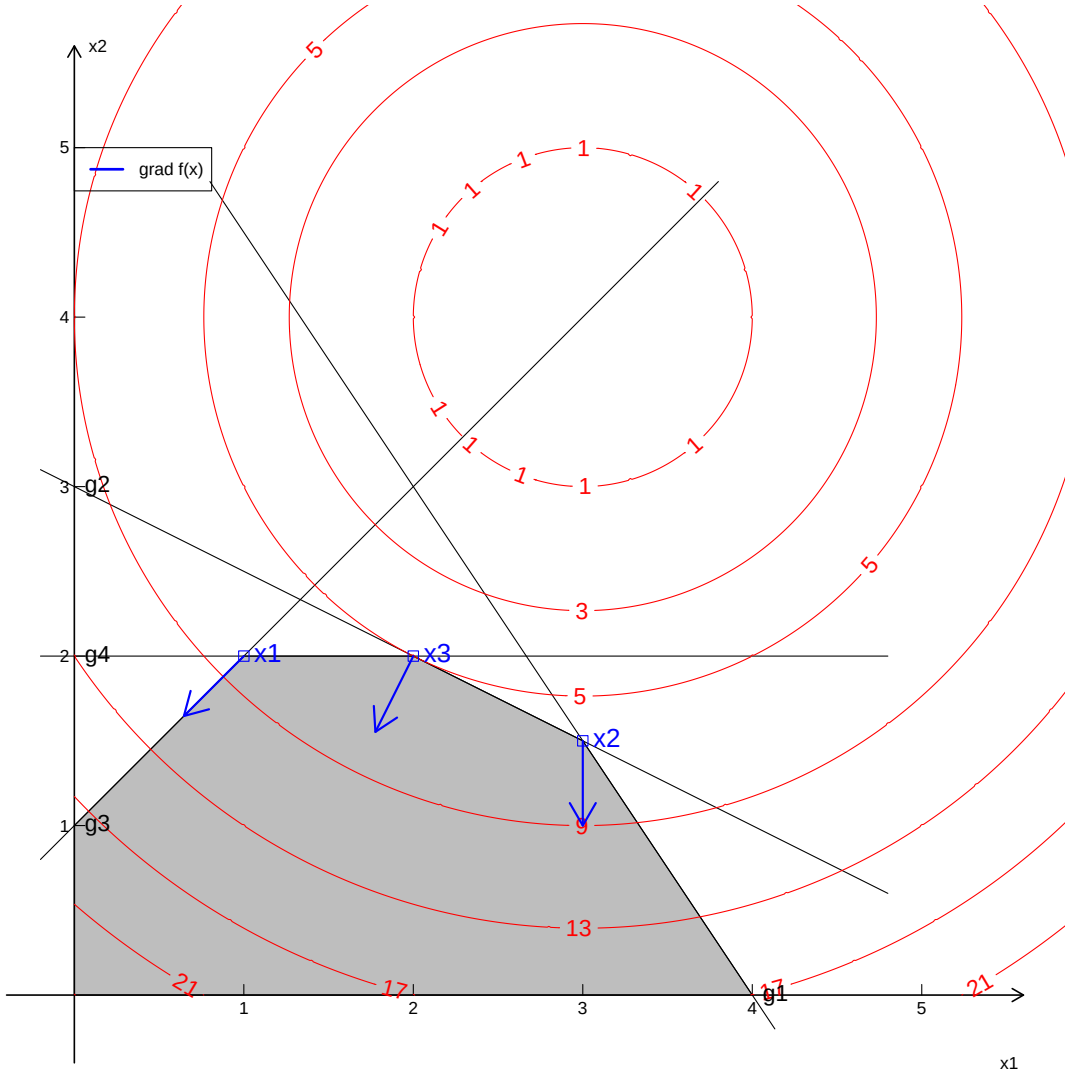

Figure 37: Graphical representation of the program S

AMPL

A tool to formulate and solve mathematical optimization problems is the *AMPL (A Mathematical Programming Language)* system which we will employ in this thesis ((Fourer, Gay, and Kernighan 1989)), ("AMPL - Streamlined Modeling for Real Optimization" 2018).

AMPL allows to specify the objective function and restrictions of a mathematical optimization problem in a standard definition language and provides access to solvers which can solve the given problem.

Additionally, in *AMPL* we distinguish between parameters and variables. Parameters represent concrete numbers that are passed to the solvers and cannot be changed by the solvers. The values of variables on the other hand represent components of the vector $\mathbf{x}$ as presented in the previous sections. The values of

variables are chosen by the algorithms to minimize the objective function and to satisfy the constraints.

The following code snippet shows the formulation and solution of the example problem S of the previous section in *AMPL* and the output (preceded by ">>") of the solver *BARON* which returns the correct solution $\mathbf{x} = (2, 2)$:

```
var x;
var y;
subject to g_1: 6*x+4*y <= 24;
subject to g_2: x+2*y <= 6;
subject to g_3: -x + y <= 1;
subject to g_4: y <= 2 ;
minimize f:(x-3)^2 + (y-4)^2;
option solver baron;
solve;
>>BARON 17.4.1 (2017.04.01). 0 iterations, optimal within tolerances.
>>Objective 5
ampl.display x;
>>x = 2
ampl.display y;
>>y = 2
```

Code excerpt - Launching test batteries

The following two python files illustrate the launching of test batteries taking as example the test battery of the section Model sequences.

File battery_model_sequence.py:

```
from test_battery_launcher import *
from model_sequence import ModelSequence
from pretty_print import get_configuration_name


def get_model_sequences_configurations():
    selected_model_sequences = [ModelSequence.GQ,
                                ModelSequence.GQ_GQ_GQ,
                                ModelSequence.BASE_GC_GQ]
    selected_configurations = []

    for solver_name in domain.get_all_solver_names():
        for model_sequence in selected_model_sequences:
            list_of_steps = []
            for model in model_sequence.get_models():
                list_of_steps.append(Step(model, solver_name,
                                          alpha_parameters=None,
                                          ampl_options=None))
            # in this test battery we use the same solver in
            # every step of the same step sequence
            # and default alpha and ampl parameters

            step_sequence = StepSequence(list_of_steps)
            step_sequence.step_sequence_name = get_configuration_name(
                solver_name=solver_name,
                model_sequence=model_sequence)
            selected_configurations.append(step_sequence)

    return selected_configurations


def main():
    scenarios = sqlalchemy_session.get_session().query(domain.Scenario)
    for scenario in scenarios:
        test_battery = ConfigurationSimulation(get_model_sequences_configurations(),
                                               scenario,
                                               "model_sequences")
        test_battery.launch()
        sqlalchemy_session.get_session().add(test_battery)

    sqlalchemy_session.close_session_and_commit()


if __name__ == "__main__":
    main()
```

File model_sequence.py:

```
from model_component_objects import *
from enum import Enum

class ModelSequence(Enum):
    GQ = ("gq",
          # we return new model objects
          # so that the caller can change them
          # (e.g. their alpha parameters)
          # without affecting other model sequences:
          lambda: [create_gq_model()])

    BASE_GC_GQ = ("base_gc_gq",
                  lambda: [create_base_model_component(),
                           base_model_component
                           + gas_consumption_component,
                           # addition of model components
                           # returns new ModelComponent instance
                           create_gq_model()])

    GQ_GQ_GQ = ("gq_gq_gq",
                lambda: [create_gq_model(),
                         create_gq_model(),
                         create_gq_model()])

    BASE_GQP_GQ = ("base_gqp_gq",
                   lambda: [create_base_model_component(),
                            create_gq_model()
                            + gas_quality_component_penalized,
                            create_gq_model()])

    GQP_GQP_GQ = ("gqp_gqp_gq",
                  lambda: [create_gq_model()
                           + gas_quality_component_penalized,
                           create_gq_model()
                           + gas_quality_component_penalized,
                           create_gq_model()])

    SMP_BASE_GQ = ("smp_base_gq",
                   lambda: [base_model_component
                            + simple_mean_pressure_component,
                            create_base_model_component(),
                            create_gq_model()])

    def __init__(self, model_sequence_name, get_models):
        self.name = model_sequence_name
        self.get_models = get_models


def create_gq_model():
    return (base_model_component + gas_consumption_component +
            gas_quality_component)
```

References

"AMPL - Streamlined Modeling for Real Optimization." 2018. https://ampl.com/.

Bazaraa, M.S., H.D. Sherali, and C.M. Shetty. 2006. *Nonlinear Programming: Theory and Algorithms*. Wiley.

Bermúdez, A., J. González-Díaz, F. González-Diéguez, A. Rueda, and M. Fernández de Córdoba. 2015. "Simulation and Optimization Models of Steady-State Gas Transmission Networks." *Energy Procedia* 64: 130–39.

Bishop, Christopher M. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag.

Bronstein, Ilja N., Konstantin A. Semendjajew, and Gerhard Musiol. 2008. *Taschenbuch Der Mathematik*. Harri Deutsch.

Byrd, Richard H., Jorge Nocedal, and Richard A. Waltz. 2006. "Knitro: An Integrated Package for Nonlinear Optimization." In *Large-Scale Nonlinear Optimization*, edited by G. Di Pillo and M. Roma, 35–59. Boston, MA: Springer US. doi:10.1007/0-387-30065-1_4.

Cao Abad, Ricardo, Mario Francisco Fernández, Salvador Naya Fernández, Manuel Antonio Presedo Quindimil, Margarita Vázquez Brage, José Antonio Vilar Fernández, and Juan Manuel Vilar Fernández. 2001. *Introducción a la estadística y sus aplicaciones*. Ediciones Pirámide.

Erdös, P., and A. Rényi. 1959. "On Random Graphs I." *Publicationes Mathematicae Debrecen* 6: 290.

Fourer, R., D. M. Gay, and B. Kernighan. 1989. "Algorithms and Model Formulations in Mathematical Programming." edited by Stein W. Wallace, 150–51. Berlin, Heidelberg: Springer-Verlag.

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Hochbaum, Dorit S. 2007. "Complexity and Algorithms for Nonlinear Optimization Problems." *Annals of Operations Research* 153 (1): 257–96.

Land, A. H., and A. G. Doig. 1960. "An Automatic Method of Solving Discrete Programming Problems." *Econometrica* 28 (3). The Econometric Society: pp. 497–520.

Murtagh, Bruce A., and Michal A. Saunders. 2003. "MINOS 5.51 User's Guide." http://www.stanford.edu/group/SOL/guides/minos551.pdf.

"Reganosa. From Experience to Innovation in Natural Gas Infrastructures." 2018. http://www.reganosa.com/en/get-know-us.

Rueda, Ángel Manuel González. 2017. "Gas Transmission Networks: Optimization Algorithms and Cost Allocation Methodologies." PhD thesis, Universidad de Santiago de Compostela.

Rumbaugh, James, Ivar Jacobson, and Grady Booch. 2004. *Unified Modeling Language Reference Manual, the (2Nd Edition)*. Pearson Higher Education.

Sahinidis, Nikolaos V. 1996. "BARON: A General Purpose Global Optimization Software Package." *Journal of Global Optimization* 8 (2): 201–5. doi:10.1007/BF00138693.

Snyman, J.A., and D.N. Wilke. 2018. *Practical Mathematical Optimization: Basic Optimization Theory and Gradient-Based Algorithms*. Springer International Publishing.

Wächter, Andreas, and Lorenz T. Biegler. 2006. "On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming." *Mathematical Programming* 106 (1): 25–57. doi:10.1007/s10107-004-0559-y.