




LOVIS ANDERSON¹, MARK TURNER², THORSTEN KOCH³

A generative deep learning approach towards decisions in transient gas networks

¹  0000-0002-4316-1862

²  0000-0001-7270-1496

³  0000-0002-1967-0077

Zuse Institute Berlin
Takustr. 7
14195 Berlin
Germany

Telephone: +49 30-84185-0
Telefax: +49 30-84185-125

E-mail: bibliothek@zib.de
URL: <http://www.zib.de>

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

A generative deep learning approach towards decision variables in transient gas networks

Lovis Anderson, Mark Turner, Thorsten Koch

January 4, 2021

Abstract

A decision support system relies on frequent re-solving of similar problems. These problems arise from the cyclical nature of such systems, and as such each problem has similar structure. We propose a generative neural network design for learning integer decision variables of mixed-integer programming (MIP) formulations of these problems. We utilise a deep neural network discriminator and a MIP solver as our oracle to train our generative neural network. As a final product, we present the results of our design applied to the transient gas optimisation problem, where we produce a feasible solution in 2.5s, which can be used as a warm-start solution to decrease global optimal solution solve time by 60.5%.

1 Introduction

Mixed-Integer Programming (MIP) is concerned with the modelling and solving of problems from discrete optimisation. These problems can represent real-world scenarios, where thanks to the integer variables, discrete decisions can be appropriately captured and modelled. In real-world scenarios a MIP problem is rarely modelled and solved only once. More frequently, near identical problems that describe different situations of the same problem are solved on a regular basis. This holds true in particular for decision support systems, which can utilise MIP to provide real-time optimal decisions on a continual basis, see [4] and [37] for examples in nurse scheduling and vehicle routing. The MIPs that these decision support systems solve have very similar structure due to their cyclical nature, and thus often have similar optimal solutions. Our aim is to exploit this repetitive structure, and create generative neural networks that generate binary decision encodings for subsets of important variables. These encodings can then be used in a primal heuristic by solving the induced subproblem following variable fixations, and then in a warm-start context to help improve solver performance in a globally optimal context. We demonstrate the performance of our neural network (NN) design on the transient gas optimisation problem [35], specifically on real instances embedded in day-ahead decision

support systems.

The design of our framework is inspired by the recent development of Generative Adversarial Networks (GANs) [16]. Our design consists of two NNs, a Generator and a Discriminator. The Generator is responsible for generating the binary decision values, while the Discriminator is tasked with predicting the optimal objective function value of the MIP induced by fixing these binary variables to their generated values. The Discriminator and Generator are separate NNs, but the Generator is trained exclusively via trying to beat the Discriminator, with a MIP oracle introduced to ensure that our Discriminator continues to learn.

Our NN design and its application to transient gas-network MIP formulations is an attempt to integrate Machine Learning (ML) into the MIP solving process. This integration has recently received an increased focus [40] [7] [15], which has been encouraged by the success of ML integration into other facets of combinatorial optimisation, see [5] for a thorough overview. Our contribution to this intersection of two fields is as follows: We introduce a new generative NN design for learning integer variables of parametric MIPs, which interacts with the MIP directly during training. We also apply our design to a much more difficult and convoluted problem than traditionally seen in similar papers, namely the transient gas transportation problem. This paper is to the best of our knowledge the first successful implementation of ML applied to discrete control in gas-transport.

2 Background and Related Work

As mentioned in the introduction, the intersection of MIP and ML is currently an area of active and growing research. For a thorough overview of Deep Learning (DL), the relevant subset of ML used throughout this article, we refer readers to [17], and for MIP to [1]. We will highlight previous research from this intersection that we believe is either tangential, or may have shared applications to that presented in this paper. Additionally, we will briefly detail the state-of-the-art in transient gas transport, and highlight why our design is of practical importance. It should be noted as-well, that there are recent research activities aiming at the reverse direction, with MIP applied to ML instead of the orientation we consider, see [42] for an interesting example.

Firstly, we summarise applications of ML to adjacent areas of the MIP solving process. Gasse et al. [15] creates a method for encoding MIP structure in a bipartite graph representing variable-constraint relationships. This structure is the input to a Graph Convolutional Neural Network (GCNN), which imitates strong branching decisions. The strength of their results stem from intelligent

network design and the generalisation of their GCNN to problems of a larger size, albeit with some generalisation loss. Zarpellon et al. [44] take a different approach, and use a NN design that incorporates the branch and bound (BB) tree state directly. In doing so, they show that information contained in the global BB tree state is an important factor in variable selection. Furthermore, they are one of the few publications to present techniques on heterogeneous instances. Etheve et al. [11] show a successful implementation of reinforcement learning for variable selection. Tang et al. [40] show preliminary results of how reinforcement learning can be used in cutting-plane selection. By restricting themselves exclusively to Gomory cuts, they are able to produce an agent capable of selecting better cuts than default solver settings for specific classes of problems.

There exists a continuous trade-off between model exactness and complexity in the field of transient gas optimisation, and as such, there is no standard model. Moritz [30] presents a piece-wise linear MIP approach to the problem, Burlacu et al. [8] a non-linear approach with a novel discretisation scheme, and Hennings et al. [23] and Hoppmann et al. [25] a linearised approach. For the purpose of our experiments, we use the model of Hennings et al. [23], which uses linearised equations and focuses on active element heavy subnetworks. Our usage of ML in gas transport is not a first, but current research is very preliminary. Pourfard et al. [34] use a dual NN design to perform online calculations of a compressor operating point to avoid re-solving the underlying model, but it ignores all integer decisions and experiments are done on a gunbarrel network. Mohamadi-Baghmolaei et al. [29] present a NN combined with a genetic algorithm for learning the relationship between compressor speeds and the fuel consumption rate in the absence of complete data. It is in related fields to transient gas transport where ML has to date been used more effectively however, with ML used to track the degradation of compressor performance [19], and forecast demand values at the boundaries of the network [32]. For a more complete overview of the transient gas literature, we refer readers to Rios et al. [35].

Our Discriminator design, which predicts the optimal objective value of an induced sub-MIP, can be considered similar to Baltean et al. [3] in what it predicts and similar to Ferber et al. [13] in how it works. In the first paper [3], a neural network is used to predict the associated objective value improvements on cuts. This is a smaller scope than our prediction, but is still heavily concerned with the MIP formulation. In the second paper [13], a technique is developed that performs backward passes directly through a MIP. It does this by solving MIPs exclusively with cutting planes, and then receiving gradient information from the KKT conditions of the final linear program. This application of neural network, which produces input to the MIP is very similar to our design. The differences arise in that we rely on a NN Discriminator to appropriately distribute the loss instead of solving a MIP directly, and that we generate variable values instead of parameter values with our Generator.

While our discriminator design is heavily inspired from GANs [16], it would be misleading to exclude adjacent and like research from the reinforcement learning community, specifically actor-critic algorithms [33]. These algorithms have shown some success for variable generation in MIP, and are notably different in that they sample from a generated distribution for down-stream decisions instead of always taking the decision with highest probability. Recently, Chen et al. [9] generated a series of coordinates for a set of UAVs using an actor-critic based algorithm, where these coordinates were continuous variables in a MIP formulation. The independence of separable sub-problems and the easily realisable value function within their formulation resulted in a natural Markov Decision Process interpretation. For a better comparison on the similarities between actor-critic algorithms and GANs, we refer readers to Pfau et al. [33].

Finally, we summarise existing research that also generate decision variable values for MIPs. Bertsimas et al. [6] [7] attempt to learn optimal solutions of parametric MILPs and MIQPs, which involves both outputting all integer decision variable values, and the active set of constraints. They do this mainly with Optimal Classification Trees in [6] and NNs in [7]. Their results are only shown on small problem sizes however, and are valid only when the parameters are sampled from a very limited range. Masti et al. [28] learn binary warm start decisions for MIQPs. They use NNs with a loss function that combines binary cross entropy and a penalty for infeasibility. Their end-goal of a primal heuristic is similar to ours, but their design is much simpler, and has only been shown to work effectively on very small problems. Ding et al. [10] presents a GCNN design which is an extension of [15], and use it to generate binary decision variable values. Their contributions are a tripartite graph encoding of MIP instances, and the inclusion of their aggregated generated values as branching decisions in the BB tree, both in an exact approach and in an approximate approach with local branching [14]. We believe that this is most similar to our work, as one of its goals is to accelerate the MIP solving process by finding quicker primal solutions through binary decision generation.

3 The Solution Framework

A traditional MIP instance takes the form:

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & c \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, x \in \mathbb{R}^{n-p} \times \mathbb{Z}^p \end{aligned}$$

For the purpose of more easily highlighting problems where we believe our framework will be effective, we introduce an additional MIP form. Note that this

notation can still describe a general MIP instance as above.

$$\begin{aligned}
\mathbb{P}_\pi := \quad & \min \quad c_1^\top x_1 + c_2^\top x_2 + c_3^\top z_1 + c_4^\top z_2 \\
& \text{s.t.} \quad A_\pi \begin{bmatrix} x_1 \\ x_2 \\ z_1 \\ z_2 \end{bmatrix} \leq b_\pi \\
& c_k \in \mathbb{R}^{n_k}, k \in \{1, 2, 3, 4\}, A_\pi \in \mathbb{R}^{m \times n}, b_\pi \in \mathbb{R}^m \\
& x_1 \in \mathbb{R}^{n_1}, x_2 \in \mathbb{R}^{n_2}, z_1 \in \mathbb{Z}^{n_3}, z_2 \in \mathbb{Z}^{n_4}
\end{aligned} \tag{1}$$

We are interested in the class of parametric MIPs where the values of A_π and b_π are parameterised by π drawn from some distribution Π . That is, the feasible region can change, but the search direction and variable dimensions are constant.

The framework is as follows: The Generator \mathbb{G}_{θ_1} , where θ_1 is the complete set of parameters that defines the NN, takes as input π . The input π is a set of constants from which the parameterised A_π and b_π can be computed. \mathbb{G}_{θ_1} outputs values for the variables z_1 , namely \hat{z}_1 . These variable values \hat{z}_1 alongside π are then input into the Discriminator \mathbb{D}_{θ_2} , where θ_2 fully describes the parameter space of the NN. \mathbb{D}_{θ_2} finally outputs a prediction of the optimal objective function value of \mathbb{P}_π with values of z_1 fixed to \hat{z}_1 , namely $\hat{\mathbb{P}}_\pi^{\hat{z}_1}$. More formally this is:

$$\hat{z}_1 = \mathbb{G}_{\theta_1}(\pi), \quad \hat{z}_1 \in \mathbb{Z}^{n_3} \tag{2}$$

$$\hat{f}(\mathbb{P}_\pi^{\hat{z}_1}) = \mathbb{D}_{\theta_2}(\hat{z}_1, \pi), \quad \hat{f}(\mathbb{P}_\pi^{\hat{z}_1}) \in \mathbb{R} \tag{3}$$

The hat notation is used to signal quantities a NN has approximated, and $f(\mathbb{P}_\pi)$ refers to the optimal objective function value of \mathbb{P}_π . We use superscript notation to create the following instances:

$$\mathbb{P}_\pi^{\hat{z}_1} \approx (1) \quad \text{s.t.} \quad z_1 = \hat{z}_1 \tag{4}$$

Note that the values of \hat{z}_1 must be appropriately rounded when explicitly solving $\mathbb{P}_\pi^{\hat{z}_1}$ s.t they are feasible w.r.t their integer constraints. See Figure 3 for the general NN layout.

The goal of this framework is to produce good initial solution values for z_1 , and thus leave an induced sub-MIP, $\mathbb{P}_\pi^{\hat{z}_1}$, whose optimal solution is a good primal to the original problem. This primal solution can then be used to warm-start \mathbb{P}_π . In the case of infeasibilities or non acceptance of primal solutions, one can introduce continuous slack variables to ensure that all decisions generated by \mathbb{G}_{θ_1} result in a feasible $\mathbb{P}_\pi^{\hat{z}_1}$. Penalising these slacks in the objective then feeds naturally into our design, where \mathbb{G}_{θ_1} aims to minimise the induced optimal objectives. This

is accounted for in our MIP definition (1), where x_2 can be considered slack variables that ensure problem feasibility for all reasonable choices of z_1 . For the purpose of our application it should be noted that z_1 and z_2 are binary variables instead of integer. We will now describe the design of \mathbb{G}_{θ_1} and \mathbb{D}_{θ_2} .

3.1 Generator and Discriminator Design

\mathbb{G}_{θ_1} and \mathbb{D}_{θ_2} are NNs whose structure is inspired by the invention of GANs [16], as well as both inception blocks and residual NNs, which have greatly increased large scale model performance [39]. We use the block design from Resnet-v2 [39], see Figure 1, albeit with slight modifications for the case of transient gas-network optimisation. Namely, as we consider our input to be one dimensional, with that dimension being time, we primarily use 1-D convolutions. Additionally, we separate initial input streams by their characteristics, and when joining two streams, use 2-D convolutions, where the second dimension is of size 2 and quickly becomes one dimensional again. See Figure 2 for an example of this process. The final layer of \mathbb{G}_{θ_1} contains a softmax activation function with temperature. As the temperature increases, this activation function's output approaches a one-hot vector encoding. The final layer of \mathbb{D}_{θ_2} contains a softplus activation function. All other intermediate layers of $\mathbb{N}_{\{\theta_1, \theta_2\}}$ use the ReLU activation function. We refer readers to Goodfellow et al. [17] for a thorough overview of deep learning, and to Figure 14 in the Appendix for our complete design.

For a vector $x = (x_1, \dots, x_n)$, the Softmax function with temperature $T \in \mathbb{R}$ (5), ReLU function (6), and Softplus function with parameter $\beta \in \mathbb{R}$ (7) are:

$$\sigma_1(x, T) := \frac{\exp(Tx_i)}{\sum_{j=1}^n \exp(Tx_j)} \quad (5)$$

$$\sigma_2(x_i) := \max(0, x_i) \quad (6)$$

$$\sigma_3(x_i, \beta) := \frac{1}{\beta} \log(1 + \exp(\beta x_i)) \quad (7)$$

We attach \mathbb{D}_{θ_2} to the end of \mathbb{G}_{θ_1} , as in Figure 3, meaning that our entire network is:

$$\mathbb{N}_{\{\theta_1, \theta_2\}}(\pi) := \mathbb{D}_{\theta_2}(\mathbb{G}_{\theta_1}(\pi), \pi) \quad (8)$$

3.2 Interpretations

In a similar manner to GANs and actor-critic algorithms [33], the design of $\mathbb{N}_{\{\theta_1, \theta_2\}}$ has a bi-level optimisation interpretation. Here we list the explicit objectives of both \mathbb{G}_{θ_1} and \mathbb{D}_{θ_2} , and how their loss functions represent these objectives.

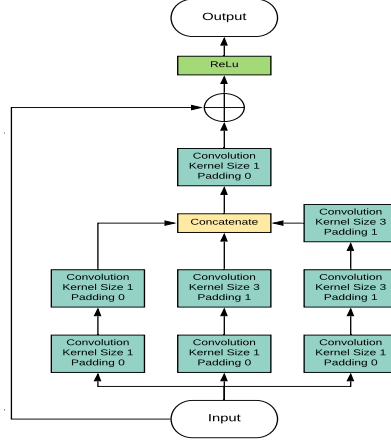


Figure 1: 1-D Resnet-v2 Block Design

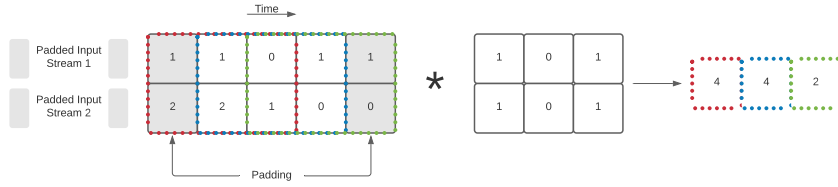


Figure 2: Method of merging two 1-D input streams

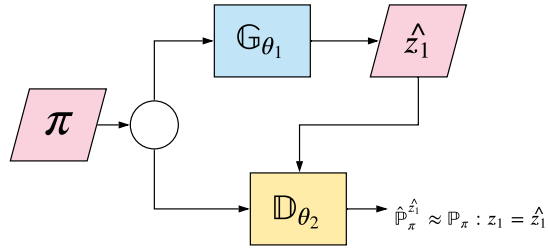


Figure 3: The general design of $\mathbb{N}_{\{\theta_1, \theta_2\}}$

The objective of \mathbb{D}_{θ_2} is to predict $f(\mathbb{P}_{\pi}^{\hat{z}_1})$, the optimal induced objective values of $\mathbb{P}_{\pi}^{\hat{z}_1}$. Its loss function is thus:

$$L(\theta_2, \pi) := |\mathbb{D}_{\theta_2}(\mathbb{G}_{\theta_1}(\pi), \pi) - f(\mathbb{P}_{\pi}^{\mathbb{G}_{\theta_1}(\pi)})| \quad (9)$$

The objective of \mathbb{G}_{θ_1} is to induce as low a value as possible from \mathbb{D}_{θ_2} . Its loss function is thus:

$$L'(\theta_1, \pi) := |\mathbb{D}_{\theta_2}(\mathbb{G}_{\theta_1}(\pi), \pi)| \quad (10)$$

These objectives can be formulated into the following bi-level optimisation problem:

$$\begin{aligned} \min_{\theta_1} \quad & \mathbb{E}_{\pi \sim \Pi} [\mathbb{D}_{\theta_2}(\mathbb{G}_{\theta_1}(\pi), \pi)] \\ \text{s.t.} \quad & \min_{\theta_2} \quad \mathbb{E}_{\pi \sim \Pi} [|\mathbb{D}_{\theta_2}(\mathbb{G}_{\theta_1}(\pi), \pi) - f(\mathbb{P}_{\pi}^{\mathbb{G}_{\theta_1}(\pi)})|] \end{aligned} \quad (11)$$

3.3 Training Method

It would be pointless to train \mathbb{G}_{θ_1} , the upper-level optimisation task in (11), without prior training of \mathbb{D}_{θ_2} , the lower-level task in (11). We therefore give \mathbb{D}_{θ_2} a head-start by training it independently in the beginning. The following loss function is used for this training:

$$L''(\theta_2, \pi) := |\mathbb{D}_{\theta_2}(z_1, \pi) - f(\mathbb{P}_{\pi}^{z_1})| \quad (12)$$

Performing this initial training requires generating instances of $\mathbb{P}_{\pi}^{z_1}$. This generation is done before training, at the very beginning of the process in an offline manner, where for $\mathbb{P}_{\pi}^{z_1}$ we sample choices of z_1 alongside π , and solve the instance to optimality before storing the optimal objective value.

After the initial training of \mathbb{D}_{θ_2} , we train \mathbb{G}_{θ_1} as a part of $\mathbb{N}_{\{\theta_1, \theta_2\}}$, using samples $\pi \in \Pi$, the loss function found in (10), and disabling updates of θ_2 . The issue of \mathbb{G}_{θ_1} outputting continuous values for \hat{z}_1 is overcome by the final layer's activation function of \mathbb{G}_{θ_1} . The softmax with temperature (5) ensures that adequate gradient information still exists to update θ_1 , and that the results are near binary. When using these results to explicitly solve $\mathbb{P}_{\pi}^{\hat{z}_1}$, we round our result to a one-hot vector encoding along the appropriate dimension.

After both these initial training methods, we then cycle through different loss functions and introduce a MIP oracle. These methods are briefly summarised below:

- \mathbb{D}_{θ_2} training:
 - As in the initial training, using loss function (12).
 - In an online fashion, using predictions from \mathbb{G}_{θ_1} and loss function (9).
- \mathbb{G}_{θ_1} training:
 - As mentioned above, using loss function (10).

By alternating these training methods, one appropriately captures information from the MIP. This is due to the network structure, where the binary variables z_1 are nodes of an intermediate layer of $\mathbb{N}_{\{\theta_1, \theta_2\}}$, and as such they are updated from down-stream information. This is largely different to other methods, many of which rely on using binary cross entropy loss against optimal solutions of \mathbb{P}_π . Our advantage over these is that the contribution to the objective function we are trying to minimise of each variable decision in z_1 can be calculated. This has an added benefit of generated suboptimal solutions being much more likely to be near-optimal, as they are trained in a manner to minimise the objective rather than copy previously observed optimal solutions.

For our application, transient gas network optimisation, methods for sampling instances currently do not exist. In fact, even gathering data is notoriously difficult, see Kunz et al. [27] and Yüksel-Ergün et al. [43] for some of the difficulties. For this reason, we introduce our own method for generating training data in section 5.

4 The Gas Transport Model

To verify the performance of our approach, we test our framework on the transient gas optimisation problem, see [35] for an overview of the problem and associated literature. This problem is much more complex, and involves more convoluted constraints than MIP formulations usually considered in similar papers. We believe that the natural modelling of transient gas networks as time-expanded networks lends itself well to our framework however, due to the static underlying network and repeated constraints at each time-step.

We use the description of transient gas networks by Hennings et al. [23]. The advantages of this description for our framework is a natural separation of z_1 variables, which induce feasible $\mathbb{P}_\pi^{z_1}$ regions for all reasonable choices due to the existence of slack variables. The gas network is modelled as a directed graph $G = (\mathcal{V}, \mathcal{A})$ where \mathcal{A} is the set of arcs representing network elements, e.g. pipes, and the nodes \mathcal{V} represent junctions between adjacent elements. Every arc $a \in \mathcal{A}$ models a specific element with $\mathcal{A} = \mathcal{A}^{\text{pi}} \cup \mathcal{A}^{\text{va}} \cup \mathcal{A}^{\text{rs}} \cup \mathcal{A}^{\text{rg}} \cup \mathcal{A}^{\text{cs}}$,

i.e., pipes, valves, resistors, regulators, and compressors. The node set \mathcal{V} also contains multiple element types, with $\mathcal{V} = \mathcal{V}^b \cup \mathcal{V}^0$ partitioned into boundary and inner nodes respectively. Note that boundary nodes represent nodes which are incident to the ‘outside’ world, and as such have prescribed pressure and flow values as a forecast over time.

It should be noted that this description focuses on *network stations*, the beating hearts of gas networks. Network stations are commonly the intersections of major pipelines, and contain nearly all the heavy machinery found in the entire gas network. We will now run through some necessary constraints, particularly those which we believe make the transient gas transportation problem a good use-case of our framework. For a full definition of the MIP, please see [23].

As we optimise the transient problem, we deal with a time horizon, namely $\mathcal{T}_0 := \{0, \dots, k\}$. We aim to calculate a network state for each $t \in \mathcal{T} := \mathcal{T}_0 \setminus \{0\}$, i.e. control decisions for all future time points. As such, the initial gas network state at time 0 contains a complete description of live information and is immutable, while all future time points contain, before optimising, only forecasted pressure and flow values at \mathcal{V}^b . We denote $\tau(t)$ as the time difference in seconds from time step 0.

4.1 Pipe Equations

Pipes constitute the majority of any large scale gas transmission network. The dynamics of flow through pipes are governed by the Euler Equations. We consider the isothermal case, and then discretise with a technique developed by Hennings in [22]. Consider the pipe $a = (u, v)$, $a \in \mathcal{A}^{\text{pi}}$, where $u, v \in \mathcal{V}$ are the two incident nodes. We attach a flow-in ($q_{u,a,t}$) and flow-out ($q_{v,a,t}$) variable to each pipe. Additionally, each incident node has an attached pressure variable, namely $(p_{u,t})$ and $(p_{v,t})$. Moreover, these flow-in, flow-out, and pressure values also appear for each time step. The variables R_s , z_a , T , D_a , L_a , s_a , A_a , g , and λ_a are all assumed to be constant. The above constant assumptions are quite common in practice [35]. It is only after setting the velocity of gas within each individual pipe, $|v_{w,a}|$ to be constant that all non-linearities are removed however. We do this via a method developed by Hennings [22] and seen in Fang et al. [12] so that our problem becomes linear. The resulting pipe equations are:

$$p_{u,t_2} + p_{v,t_2} - p_{u,t_1} - p_{v,t_1} + \frac{2R_s T z_a (\tau(t_2) - \tau(t_1))}{L_a A_a} (q_{v,a,t_2} - q_{u,a,t_2}) = 0 \quad (13)$$

$$p_{v,t_2} - p_{u,t_2} + \frac{\lambda_a L_a}{4D_a A_a} (|v_{u,a}| q_{u,a,t_2} + |v_{v,a}| q_{v,a,t_2}) + \frac{g s_a L_a}{2R_s T z_a} (p_{u,t_2} + p_{v,t_2}) = 0 \quad (14)$$

As nodes are simply junctions between network elements, and thus have no volume in which to store any gas, we require flow conservation constraints. In the below equations, $d_{v,t}$ represents the inflow resp. outflow of entry and exit nodes in the network at time $t \in \mathcal{T}_0$. Note that network elements that aren't pipes have only one associated flow variable, instead of the in-out flow exhibited by pipes. This is due to them having no volume, and as such no ability to store gas over time, i.e. line-pack.

$$\begin{aligned} & \sum_{(u,w)=a \in \mathcal{A}^{\text{pi}}} q_{w,a,t} - \sum_{(w,v)=a \in \mathcal{A}^{\text{pi}}} q_{w,a,t} \\ & + \sum_{(u,w)=a \in \mathcal{A} \setminus \mathcal{A}^{\text{pi}}} q_{a,t} - \sum_{(w,v)=a \in \mathcal{A} \setminus \mathcal{A}^{\text{pi}}} q_{a,t} + d_{w,t} = 0 \quad \forall w \in \mathcal{V}^b \end{aligned} \quad (15)$$

$$\begin{aligned} & \sum_{(u,w)=a \in \mathcal{A}^{\text{pi}}} q_{w,a,t} - \sum_{(w,v)=a \in \mathcal{A}^{\text{pi}}} q_{w,a,t} \\ & + \sum_{(u,w)=a \in \mathcal{A} \setminus \mathcal{A}^{\text{pi}}} q_{a,t} - \sum_{(w,v)=a \in \mathcal{A} \setminus \mathcal{A}^{\text{pi}}} q_{a,t} = 0 \quad \forall w \in \mathcal{V}^0 \end{aligned} \quad (16)$$

4.2 Operation Modes

Operation modes represent binary decisions in our gas network, and it is these binary decisions that correspond to z_1 from our description of a general MIP (1). Let \mathcal{O} represent the set of operation modes, and $m_{o,t}^{\text{om}}$ the associated variables. Operation Modes are very important in our modelling context as they describe every allowable combination of discrete decisions associated with *valves* and *compressors*.

4.2.1 Compressors

Compressors are typically set up as a compressor station consisting of multiple compressor units, which represent the union of one single compressor machine and its associated drive. These compressor units are dynamically switched on or off and used in different sequences to meet the current needs in terms of compression ratios and flow rates. Out of the theoretically possible arrangements of compressor units, the set of technically feasible arrangements are known as

the *configurations* of a compressor station.

Selecting an operation mode results in fixed configurations for all compressor stations. The binary variables associated with a compressor station $a = (u, v) \in \mathcal{A}^{\text{cs}}$ at time $t \in \mathcal{T}_0$ are $m_{a,t}^{\text{by}}$ (bypass), $m_{a,t}^{\text{cl}}$ (closed), and $m_{c,a,t}^{\text{cf}} \forall c \in \mathcal{C}_a$ (active). \mathcal{C}_a denotes the set of configurations associated to compressor station a , where the configuration's operating range is a polytope in space $(p_{u,t}, p_{v,t}, q_{u,a,t})$. The polytope of configuration c is represented by the intersection of half-spaces, $\mathcal{H}_c = \{(\alpha_0, \alpha_1, \alpha_2, \alpha_3) \in \mathbb{R}^4\}$.

$$1 = \sum_{c \in \mathcal{C}_a} m_{c,a,t}^{\text{cf}} + m_{a,t}^{\text{by}} + m_{a,t}^{\text{cl}} \quad (17)$$

$$\alpha_0 p_{c,a,t}^{\text{u-cf}} + \alpha_1 p_{c,a,t}^{\text{v-cf}} + \alpha_2 q_{c,a,t}^{\text{cf}} + \alpha_3 m_{c,a,t}^{\text{cf}} \leq 0 \quad (18)$$

$$\forall (\alpha_0, \alpha_1, \alpha_2, \alpha_3) \in \mathcal{H}_c \quad \forall c \in \mathcal{C}_a$$

Note that the variables in (18) have an extra subscript and superscript compared to those in (13) and (14). This is due to our use of the convex-hull reformulation, see Balas [2]. The additional subscript refers to the configuration in question, and the superscript the mode, with the pressure variables having an additional node identifier. It should also be noted that the continuous variables attached to a compressor station are not fixed by a choice in operation mode or configuration, but rather the operation mode restricts the variables to some polytope.

4.2.2 Valves

Valves decide the allowable paths through a network, and can separate areas, decoupling their pressure levels. They are a network element, and thus modelled as an arc $a = (u, v)$, whose discrete decisions can be decided by an operation mode choice. Valves have two modes, namely open and closed. When a valve is open, similar to a compressor station in bypass, flow can flow freely and there exists no pressure difference between the valves start and endpoints. Alternatively in the closed mode, a valve allows no flow to pass, and decouples the pressure of the start- and endpoints of the arc. The variable $m_{a,t}^{\text{op}}$ represents a valve being open with value 1 and closed with value 0. The notation \underline{x} and \bar{x} refer to lower and upper bounds of variable x , with the constraints describing valves being as follows:

$$p_{u,t} - p_{v,t} \leq (1 - m_{a,t}^{\text{op}})(\bar{p}_{u,t} - \underline{p}_{v,t}) \quad (19)$$

$$p_{u,t} - p_{v,t} \geq (1 - m_{a,t}^{\text{op}})(\underline{p}_{u,t} - \bar{p}_{v,t}) \quad (20)$$

$$q_{a,t} \leq (m_{a,t}^{\text{op}})\bar{q}_{a,t} \quad (21)$$

$$q_{a,t} \geq (m_{a,t}^{\text{op}})\underline{q}_{a,t}. \quad (22)$$

4.2.3 Valid Operation Modes

As mentioned earlier, not all combinations of compressor station configurations and valve states are possible. We thus define a mapping $M(o, a)$ from operation mode $o \in \mathcal{O}$ to the discrete states of all $a \in \mathcal{A}^{\text{va}} \cup \mathcal{A}^{\text{cs}}$

$$\begin{aligned} M(o, a) &:= x \text{ where } x \text{ is the mode or configuration of arc } a \\ &\quad \text{in operation mode } o \quad \forall o \in \mathcal{O} \quad \forall a \in \mathcal{A}^{\text{va}} \cup \mathcal{A}^{\text{cs}} \\ \text{with} \quad &x \in \{\text{op}, \text{cl}\} \text{ if } a \in \mathcal{A}^{\text{va}} \\ &x \in \{\text{by}, \text{cl}\} \cup \mathcal{C}_a \text{ if } a \in \mathcal{A}^{\text{cs}} \end{aligned}$$

Using this mapping we can then define a set of constraints for all valid combinations of compressor station and valve discrete states for each $t \in \mathcal{T}$. The variable $m_{o,t}^{\text{om}}$, $o \in \mathcal{O}$ $t \in \mathcal{T}$, is a binary variable, where the value 1 represents the selection of o at time step t .

$$\sum_{o \in \mathcal{O}} m_{o,t}^{\text{om}} = 1 \quad (23)$$

$$m_{a,t}^{\text{op}} = \sum_{o \in \mathcal{O}: M(o,a)=\text{op}} m_{o,t}^{\text{om}} \quad \forall a \in \mathcal{A}^{\text{va}} \quad (24)$$

$$m_{a,t}^{\text{by}} = \sum_{o \in \mathcal{O}: M(o,a)=\text{by}} m_{o,t}^{\text{om}} \quad \forall a \in \mathcal{A}^{\text{cs}} \quad (25)$$

$$m_{a,t}^{\text{cl}} = \sum_{o \in \mathcal{O}: M(o,a)=\text{cl}} m_{o,t}^{\text{om}} \quad \forall a \in \mathcal{A}^{\text{cs}} \quad (26)$$

$$m_{c,a,t}^{\text{cf}} = \sum_{o \in \mathcal{O}: M(o,a)=c} m_{o,t}^{\text{om}} \quad \forall c \in \mathcal{C}_a \quad \forall a \in \mathcal{A}^{\text{cs}} \quad (27)$$

$$m_{o,t}^{\text{om}} \in \{0, 1\} \quad \forall o \in \mathcal{O}.$$

4.3 Flow Directions

Flow Directions formally define the sign of flow values over the boundary nodes of a network station. With regards to our MIP they are a further set of decision variables. We avoid generating these decisions with our deep learning framework as not all combinations of operation modes and flow directions are feasible. These variables thus exist as integer variables in \mathbb{P}_{π}^{z1} , but are few in number due to the limited combinations after the operation modes are fixed.

4.4 Boundary Nodes and Slack

Boundary nodes, unlike inner nodes, have a prescribed flow and pressure values for all future time steps. For each boundary node $v \in \mathcal{V}^{\text{b}}$ and $t \in \mathcal{T}$, we have

$\sigma_{v,t}^{p+}$ and $\sigma_{v,t}^{p-}$, which capture the positive and negative difference between the prescribed and realised pressure. In addition to these pressure slack variables, we have the inflow slack variables $\sigma_{v,t}^{d+}$ and $\sigma_{v,t}^{d-}$ which act in a similar manner but for inflow. The relationships between the slack values, prescribed values, and realised values can be modelled for each $v \in \mathcal{V}^b$ and $t \in \mathcal{T}$ as:

$$\hat{p}_{v,t} = p_{v,t} - \sigma_{v,t}^{p+} + \sigma_{v,t}^{p-} \quad \forall v \in \mathcal{V}^b \quad (28)$$

$$\hat{d}_{v,t} = d_{v,t} - \sigma_{v,t}^{d+} + \sigma_{v,t}^{d-} \quad \forall v \in \mathcal{V}^b \quad (29)$$

Note that unlike the model from [23], we do not allow the inflow over a set of boundary nodes to be freely distributed according to which group they belong to. This is an important distinction, as each single node has a complete forecast.

4.5 Initial State

In addition to the forecast mentioned in subsection 4.4, we also start our optimisation problem with an initial state. This initial state contains complete information of all discrete states and continuous values for all network elements at $t = 0$.

4.6 Objective function

The objective of our formulation is to both minimise slack usage, and changes in network operation. Specifically, it is a weighted sum of changes in the active element modes, changes in the active points of operation, and the deviations from given pressure and flow demands. For the exact objective function we refer readers to Hennings et al. [23].

5 Computational Experiments

In this section we propose an experimental design to determine the effectiveness of our neural network design. We outline how we generate synthetic training data, and show the exact architecture and training method we use for our neural network. Our final test set consists of 15 weeks of real-world data provided by our project partner OGE.

5.1 Data Generation

As mentioned previously, acquiring gas network data is notoriously difficult [43] [27]. Perhaps because of this difficulty, there exists no standard method for generating valid states for a fixed gas network. Below we outline our methods for generating synthetic transient gas instances for training purposes, i.e. generating $\pi \in \Pi$ and artificial z_1 values. For our application of transient gas instances, π is a tuple of a boundary forecast, and an initial state.

5.1.1 Boundary Forecast Generation

We consider network stations as our gas network topology. They contain all heavy machinery, and at most only minor segments of large scale transport pipelines. As such, our gas networks cannot be used to store large amounts of gas. We thus aim to generate balanced demand scenarios, with the requirement described as follows:

$$\sum_{v \in \mathcal{V}^b} \hat{d}_{v,t} = 0 \quad \forall t \in \mathcal{T} \quad (30)$$

The distribution of gas demand scenarios is not well known. Hence we naively assume a uniform distribution, and using the largest absolute flow value found over any node and time step in our real-world data, create an interval as follows:

$$\begin{aligned} M_q &= \max_{v \in \mathcal{V}^b, t \in \mathcal{T}} |\hat{d}_{v,t}| \\ \hat{d}_{v,t} &\in [-1.05M_q, 1.05M_q] \end{aligned} \quad (31)$$

In addition to the above, we require three MIP formulation specific requirements. The first is that the flow does not differ too largely between any two adjacent time steps for the same node. Secondly, the sign of the generated flow values must match the attribute of the boundary node, i.e., entry (+), exit (-). Thirdly, the flow values do not differ too largely between boundary nodes of the same *fence group* within the same time step. A fence group is denoted by $g \in \mathcal{G}$, and enforces the sign of all nodes in the group to be identical. These constraints are described below:

$$\begin{aligned} |\hat{d}_{v,t} - \hat{d}_{v,t-1}| &\leq 200 \quad \forall t \in \mathcal{T}, \quad v \in \mathcal{V}^b \\ \text{sign}(\hat{d}_{v,t}) &= \begin{cases} 1 & \text{if } v \in \mathcal{V}^+ \\ -1 & \text{if } v \in \mathcal{V}^- \end{cases} \quad \forall t \in \mathcal{T}, \quad v \in \mathcal{V}^b \\ |\hat{d}_{v_1,t} - \hat{d}_{v_2,t}| &\leq 200 \quad \forall t \in \mathcal{T}, \quad v_1, v_2 \in g, \quad g \in \mathcal{G}, \quad v_1, v_2 \in \mathcal{V}^b \end{aligned} \quad (32)$$

To generate demand scenarios that satisfy constraints (30) and (31), we use a method proposed by Rubin [36]. This method's original purpose was to generate samples from the Dirichlet distribution, but can be used for a special case of the Dirichlet distribution that is equivalent to a uniform distribution over a simplex in 3-dimensions. Such a simplex is exactly described by a single time step of (30) and (31). We then perform this for all time-steps and simply reject all samples via this method that do not satisfy constraints (32). Note that

this method is insufficient for network stations with more than 3 entry-exit pairs.

In addition to flow demands, we require pressure demands. Our only requirements for these is that the pressure demands between adjacent time steps for a single node not fluctuate heavily, and that the bounds are respected. We retrieve a bound on the range of pressure values by finding maximum and minimum values over all nodes and time steps in our test set. We once again assume our samples to be uniformly distributed and sample appropriately over (33) with rejection of samples that don't respect constraint (34). Note that many scenarios generated by this approach are unrealistic, as the pressure and flow profiles may not match.

$$\begin{aligned} M_p^+ &= \max_{v \in \mathcal{V}^b, t \in \mathcal{T}} \hat{p}_{v,t} & M_p^- &= \min_{v \in \mathcal{V}^b, t \in \mathcal{T}} \hat{p}_{v,t} \\ \hat{p}_{v,t} &\in [M_p^- - 0.05(M_p^+ - M_p^-), M_p^+ + 0.05(M_p^+ - M_p^-)] \end{aligned} \quad (33)$$

$$|\hat{p}_{v,t} - \hat{p}_{v,t-1}| \leq 5 \quad \forall t \in \mathcal{T}, \quad v \in \mathcal{V}^b \quad (34)$$

Combining the above generation methods, we create an algorithm for generating artificial forecast data:

Algorithm 1: Boundary Prognosis Generator

Result: A forecast of pressure and flow values over the time horizon
flow_forecast = Sample simplex (30), (31) uniformly, rejecting via (32) ;
pressure_forecast = Sample (33) uniformly, rejecting via (34) ;
return (flow_forecast, pressure_forecast)

5.1.2 Operation Mode Sequence Generation

During offline training, \mathbb{D}_{θ_2} requires optimal solutions for a fixed z_1 . In Algorithm 2 we outline a naive yet effective approach of generating reasonable z_1 values, i.e., operation mode sequences:

Algorithm 2: Operation Mode Sequence Generator

Result: An Operation Mode per time step

```
operation_modes = [ ] ;  
for  $t = 1$ ;  $t < |\mathcal{T}|$ ;  $t = t + 1$  do  
    if  $t == 1$  then  
        | new_operation_mode = rand( $\mathcal{O}$ ) ;  
    end  
    else if  $\text{rand}(0,1) \geq 0.9$  then  
        | new_operation_mode = rand( $\mathcal{O} \setminus \text{new\_operation\_mode}$ ) ;  
    end  
    operation_modes.append(new_operation_mode) ;  
end  
return operation_modes
```

5.1.3 Initial State Generation

As the underlying network topology of the gas-network does not change, only the element states therein, certain coefficients of A_π are invariant. Additionally, other coefficients are found by substituting into equations multiple constants that describe gas properties. This information is contained in the initial state. We generate these constants in a similar manner to our boundary forecasts:

$$c_{\text{state}} \in \{\text{Temperature, Inflow Norm Density, Molar Mass, Pseudo Critical Temperature, Pseudo Critical Pressure}\} \quad (35)$$

$$M_c^+ = \max_{\text{state} \in \text{initial states}} c_{\text{state}} \quad M_c^- = \min_{\text{state} \in \text{initial states}} c_{\text{state}} \quad (36)$$
$$c_{\text{state}} \in [M_c^- - 0.05(M_c^+ - M_c^-), M_c^+ + 0.05(M_c^+ - M_c^-)]$$

We now have all the necessary tools to generate synthetic initial states for gas networks. The process to do so is as follows:

Algorithm 3: Initial State Generator

Input: Desired time-step distance, $j \in [1, \dots, k]$

Result: An initial state to the transient gas optimisation problem
flow_forecast, pressure_forecast = Boundary Prognosis Generator() ^a ;

gas_constants = Sample (36) uniformly ;

initial_state = Select random state from real-world data ;

$\pi = (\text{flow_forecast}, \text{pressure_forecast}, \text{gas_constants}, \text{initial_state})$ ^b ;

$z_1 = \text{Operation Mode Sequence Generator}()$ ^c ;

$\mathbb{P}_\pi^{z_1} = \text{generate from } \pi \text{ and } z_1$;

(state_1, \dots , state_k) = Optimal solution states from solving $\mathbb{P}_\pi^{z_1}$;

return state_j

^aSee Algorithm 1

^bNote that in general our π does not include gas_constants. This is because the information is generally encoded in initial_state. Our gas_constants in this context are randomly generated however, and may not match the initial_state. This does not affect solving as these values are simply taken as truths.

^cSee Algorithm 2

This method is imperfect yet effective. It outputs varied and valid initial states w.r.t our MIP formulation. There do exist flaws in this process however. Firstly, the underlying distribution of demand scenarios for both flow and pressure are probably not uniform. Moreover, the sampling range we use is significantly larger than that of our test set as we take single maximum and minimum values over all nodes. Secondly, the choice of operation modes that occur in reality is also not uniform. In reality, some operation modes occur with a much greater frequency than others. Our data is thus more varied than reality, and likely to contain operation mode choices that do match the demand scenarios. Finally, we rely on a MIP solver to generate new initial states in our final step. Hence we cannot exclude the possibility of a slight bias. An example of where this might occur, is in the case of a repeated scenario having many optimal solutions, and the MIP solver always returning an identical solution.

In the case of initial state generation, we believe that further research needs to be performed. Our method is effective in the context of machine learning where we are after a diverse set of data, but it is naive and incapable of ensuring that generated boundary scenarios are realistic.

5.1.4 Complete Transient Gas Instance Generation

To train \mathbb{D}_{θ_2} and \mathbb{G}_{θ_1} , we need both the transient gas transportation scenario, and the optimal solution of said scenario. Combining the generation methods

for synthetic data in subsections 5.1.1, 5.1.2, and 5.1.3, we create Algorithm 4:

Algorithm 4: Synthetic Gas Data Generator

Input: num_states, num_scenarios, time_step_difference
Result: num_scenarios many gas instances and their optimal solutions
initial_states = [] ;
for $i = 0; i < \text{num_states}$ **do**
 | initial_states.append(Initial State Generator(time_step_difference))^a;
end
forecasts = [] ;
for $i = 0; i < \text{num_scenarios}$ **do**
 | flow_forecast, pressure_forecast = Boundary Prognosis Generator()^b;
 | forecasts.append((flow_forecast, pressure_forecast)) ;
end
solve_data = [] ;
for $i = 0; i < \text{num_scenarios}$ **do**
 | z_1 = Operation Mode Sequence Generator()^c ;
 | initial_state = Uniformly select from initial_states ;
 | $\pi = (\text{forecasts}[i], \text{initial_state})$;
 | $\mathbb{P}_\pi^{z_1}$ = Create MIP from π and z_1 ;
 | solution = Solve $\mathbb{P}_\pi^{z_1}$;
 | solve_data.append((z_1, π , solution)) ;
end
return solve_data

^aSee Algorithm 3

^bSee Algorithm 1

^cSee Algorithm 2

5.2 Experimental Design

We generate initial training and validation sets offline. To do so we use Algorithm 4 with inputs:

$$(\text{num_states} = 10^4, \text{num_scenarios} = 4 \times 10^6, \text{time_step_difference} = 8)$$

This initial training data is used to train \mathbb{D}_{θ_2} , and split into a training set of size 3.2×10^6 , a test set of 4×10^5 , and a validation set of 4×10^5 .

The test set is checked against at every epoch, while the validation set is only referred to at the end of the initial training. Following this initial training, we begin to train $\mathbb{N}_{\{\theta_1, \theta_2\}}$ as a whole, alternating between \mathbb{G}_{θ_1} and \mathbb{D}_{θ_2} . The exact algorithm is given in 7, which references functions provided in the appendix. For training, we used the Adam algorithm [26] as our descent method. The associated parameters to this algorithm and a complete set of other training parameters are listed in Table 4. In the case of a parameter being non-listed, the default value was used. A large intention behind our training method is to

ensure that no real-world data is given to $\mathbb{N}_{\{\theta_1, \theta_2\}}$ prior to its final evaluation. With this method we hope to show that synthetic data is sufficient for training purposes, and that $\mathbb{N}_{\{\theta_1, \theta_2\}}$ successfully generalises to additional data sets. We should note that our initial state creation method does use real-world data as a starting point from which to generate artificial data however, see Algorithm 3.

As well as training $\mathbb{N}_{\{\theta_1, \theta_2\}}$ and comparing the induced \hat{z}_1 variables and $\mathbb{P}_\pi^{\hat{z}_1}$ instances, we consider the solution of $\mathbb{P}_\pi^{\hat{z}_1}$ as a primal heuristic for the original problem \mathbb{P}_π . This is because of our use of slack, where any valid solution of $\mathbb{P}_\pi^{\hat{z}_1}$ is a valid solution of \mathbb{P}_π . We aim to incorporate $\mathbb{N}_{\{\theta_1, \theta_2\}}$ in a global MIP context, and will do this by using a partial solution of $\mathbb{P}_\pi^{\hat{z}_1}$ as a warm-start suggestion for \mathbb{P}_π . The partial solution consists of \hat{z}_1 , an additional set of binary variables called the flow directions, and $p_{v,t} \forall v \in \mathcal{V}^b, t \in \mathcal{T}$. We only use a partial solution as our instances are numerically difficult. In doing so, we hope to generate valid solutions quickly, and speed up the global solution process. Both primal heuristic and warm-start algorithms can be seen in Algorithms 5 and 6 respectively.

Algorithm 5: Primal Heuristic

Input: \mathbb{P}_π
 $\hat{z}_1 = \mathbb{G}_{\theta_1}(\pi)$;
 $\mathbb{P}_\pi^{\hat{z}_1} = \text{Create MIP from } \pi \text{ and } \hat{z}_1$;
solution = Solve $\mathbb{P}_\pi^{\hat{z}_1}$;
return solution ;
Result: Optimal solution of $\mathbb{P}_\pi^{\hat{z}_1}$, primal solution of \mathbb{P}_π .

Algorithm 6: Warm Start Algorithm

Input: \mathbb{P}_π
primal_solution = Primal Heuristic(\mathbb{P}_π) ^a ;
optimum = Solve \mathbb{P}_π using primal_solution as a warm-start suggestion ;
Result: Optimal solution of \mathbb{P}_π

^aSee Algorithm 5

For our experiments we used PyTorch 1.4.0 [31] as our ML modelling framework, Pyomo v5.5.1 [20] [21] as our MIP modelling framework, and Gurobi v9.02 [18] as our MIP solver. The MIP solver settings are available in Table 5 in the appendix. $\mathbb{N}_{\{\theta_1, \theta_2\}}$ was trained on a machine running Ubuntu 18, with 384 GB of RAM, composed of 2x *Intel(R) Xeon(R) Gold 6132* running @ 2.60GHz, and 4x *NVIDIA Tesla V100 GPU-NVTV100-16*. The final evaluation times were performed on a cluster using 4 cores and 16 GB of RAM of a machine composed of 2x *Intel Xeon CPU E5-2680* running @ 2.70 GHz.

Algorithm 7: Neural Network Training

Input: Neural network $\mathbb{N}_{\{\theta_1, \theta_2\}}$, prelabelled_data
Result: Trained neural network $\mathbb{N}_{\{\theta_1, \theta_2\}}$

```
set_trainable( $\mathbb{D}_{\theta_2}$ );  
set_untrainable( $\mathbb{G}_{\theta_1}$ );  
Discriminator Pretraining( $\mathbb{D}_{\theta_2}$ , prelabelled_data) a ;  
softmax_temperature = 0;  
data = [];  
for  $i = 0$ ;  $i < num\_epochs$  do  
    set_trainable( $\mathbb{G}_{\theta_1}$ );  
    set_untrainable( $\mathbb{D}_{\theta_2}$ );  
    for  $i = 0$ ;  $i < num\_generator\_epochs$  do  
        softmax_temperature += 1;  
        set( $\mathbb{G}_{\theta_1}$ , softmax_temperature);  
        loss = Generator Training( $\mathbb{N}_{\{\theta_1, \theta_2\}}$ ) b ;  
        if  $loss \leq stopping\_loss\_generator$  then  
            break;  
        end  
    end  
    set_trainable( $\mathbb{D}_{\theta_2}$ );  
    set_untrainable( $\mathbb{G}_{\theta_1}$ );  
    data = Prepare Discriminator Training Data( $\mathbb{N}_{\{\theta_1, \theta_2\}}$ , data) c ;  
    mixed_data = MixData(data, prelabelled_data, num_prelabelled);  
    training_data, test_data = split_data(mixed_data, ratio_test);  
    optimizer = Adam(learning_rate, weight_decay) d ;  
    lr_scheduler = ReduceLROnPlateau e (patience, factor);  
    dataloader = DataLoader(training_data, batch_size, shuffle=True);  
    for  $i = 0$ ;  $i < num\_discriminator\_epochs$  do  
        Discriminator Training Loop( $\mathbb{D}_{\theta_2}$ , dataloader, optimizer) f ;  
        lr_scheduler.step();  
        test_loss = compute_L1Loss( $\mathbb{D}_{\theta_2}$ , test_data);  
        if  $test\_loss \leq stopping\_loss\_discriminator$  then  
            break;  
        end  
    end  
end  
return  $\mathbb{N}_{\{\theta_1, \theta_2\}}$ 
```

^aSee Algorithm 9

^bSee Algorithm 10

^cSee Algorithm 8

^dSee Kingma et al. [26] pytorch.org/docs/stable/optim.html?highlight=adam#torch.optim.Adam.

^eSee pytorch.org/docs/stable/optim.html#torch.optim.lr_scheduler.ReduceLROnPlateau.

^fSee Algorithm 11

Our validation set for the final evaluation of $\mathbb{N}_{\{\theta_1, \theta_2\}}$ consists of 15 weeks of live real-world data from our project partner OGE. Instances are on average 15 minutes apart for this period and total 9291.

All instances, both in training and test, contain 12 time steps (excluding the initial state) with 30 minutes between each step. Additionally, we focus on Station D from [23], and present only results for this station. The statistics for Station D can be seen in Table 1, and its topology in Figure 4. Station D can be thought of as a T intersection, and is of middling complexity compared to the stations presented in [23]. The station contains 6 boundary nodes, but they are paired, such that for each pair only one can be active, i.e., have non-zero flow. Due to this, our sampling method in subsection 5.1.1 exists in 3-dimensions and is uniform $\forall t \in \mathcal{T}$.

Name	$ \mathcal{V} $	$ \mathcal{A} $	$\frac{\sum_{a \in \mathcal{A}^{\text{pi}}} L_a}{ \mathcal{A}^{\text{pi}} }$	$ \mathcal{C}_a \ \forall a \in \mathcal{A}^{\text{cs}}$	$ \mathcal{O} $	$ \mathcal{V}^{\text{b}} $	$ \mathcal{A}^{\text{va}} $
D	31	37	0.404 km	2, 6	56	3x2	11

Table 1: Overview of different properties of station D.

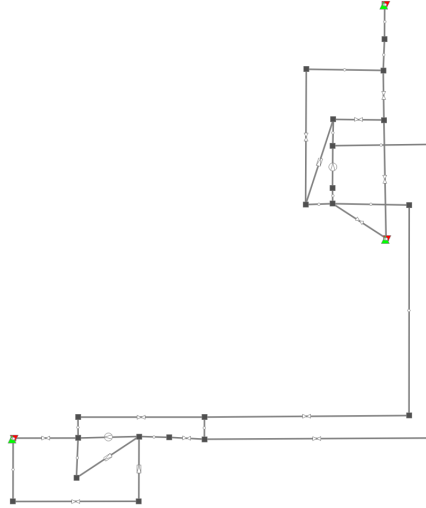


Figure 4: Topology of Station D.

5.3 Exact Network Designs

As a large portion of our input data into both \mathbb{G}_{θ_1} and \mathbb{D}_{θ_2} is time-expanded data, we originally believed that the ideal design would be a series of

LSTMs [24]. Preliminary results however showed that convolutional neural networks (CNNs) were more effective for our problem, and this became especially so when using Inception Blocks [39].

The exact block design used in $\mathbb{N}_{\{\theta_1, \theta_2\}}$ can be seen in Figure 1, and the general layout in Figure 3. For the complete network design we refer readers to Figure 14 and Table 6 in the Appendix.

6 Computational Results

We partition our results into three subsections. The first focuses on the training results of $\mathbb{N}_{\{\theta_1, \theta_2\}}$, the second on our data generation methods, while the third is concerned with our results on the 15 weeks of real-world transient gas data. Note that when training we scaled $f(\mathbb{P}_\pi^{z_1})$ values by 500 to reduce the magnitude of the losses. For visualisation purposes of comparing the performance of $\mathbb{N}_{\{\theta_1, \theta_2\}}$ and our data generation methods, we rescaled all results.

6.1 Training Results

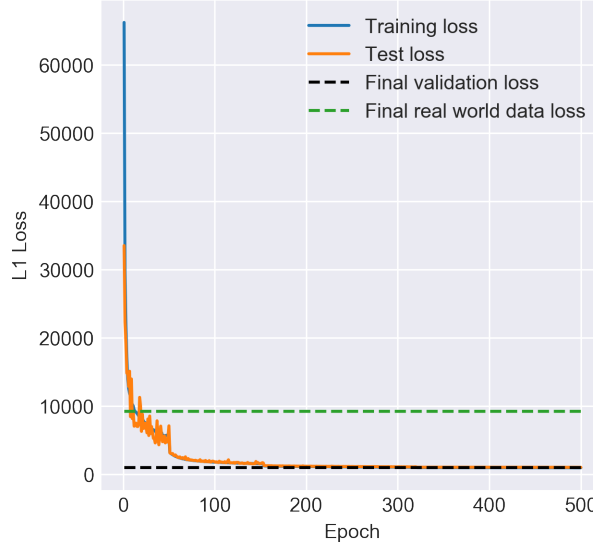


Figure 5: The loss per epoch of \mathbb{D}_{θ_2} during the initial training of Algorithm 9. The dashed lines show the performance of \mathbb{D}_{θ_2} after $\mathbb{N}_{\{\theta_1, \theta_2\}}$ has been completely trained.

Figure 5 shows the training loss throughout the initial offline training. We see

that \mathbb{D}_{θ_2} learns how to accurately predict $f(\mathbb{P}_{\pi}^{z_1})$ as the loss decreases. This is a required result, as without a trained discriminator we cannot expect to train a generator. Both the training and test loss converge to approximately 1000, which is excellent considering the generated $f(\mathbb{P}_{\pi}^{z_1})$ range well into the millions. As visible by both the test loss and final validation loss, we see \mathbb{D}_{θ_2} generalises to $\mathbb{P}_{\pi}^{z_1}$ instances of our validation set that it has not seen. This generalisation ability doesn't translate perfectly to real-world data however. This is due to the underlying distribution of real-world data and our generated data being substantially different. Despite this we believe that an L1 loss, in this case simply the average distance between $\hat{f}(\mathbb{P}_{\pi}^{z_1})$ and $f(\mathbb{P}_{\pi}^{z_1})$, of 10000 is still very good. We discuss the issues of different distributions in subsection 6.2.

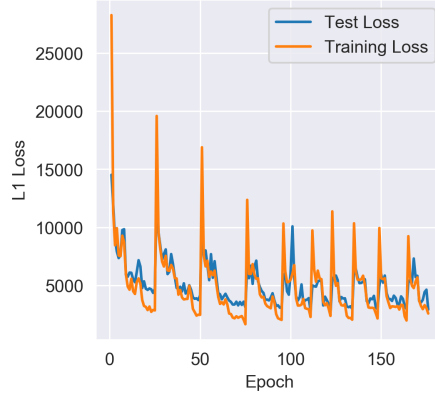


Figure 6: The loss per epoch of \mathbb{D}_{θ_2} as it is trained using Algorithm 7

The loss during training using Algorithm 7 for \mathbb{D}_{θ_2} is shown in Figure 6, and for \mathbb{G}_{θ_1} in Figure 7. The cyclical nature of the \mathbb{D}_{θ_2} loss is caused by the re-training of \mathbb{G}_{θ_1} , which learns how to induce sub-optimal predictions from the then static \mathbb{D}_{θ_2} . These sub-optimal predictions are quickly re-learned, but highlight that learning how to perfectly predict $f(\mathbb{P}_{\pi}^{z_1})$ over all possibilities, potentially due to the rounded nature of \hat{z}_1 , is unlikely without some error. Figure 7 (left) shows the loss over time of \mathbb{G}_{θ_1} as it is trained, with Figure 7 (right) displaying magnified losses for the final epochs. We observe that \mathbb{G}_{θ_1} quickly learns important z_1 decision values. We hypothesise that this quick descent is helped by \hat{z}_1 that are unlikely given our generation method in Algorithm 2. The loss increases following this initial decrease in the case of \mathbb{G}_{θ_1} , showing the ability of \mathbb{D}_{θ_2} to further improve. It should also be noted that significant step-like decreases in loss are absent in both (left) and (right) of Figure 7. Such steps would indicate \mathbb{G}_{θ_1} discovering new important z_1 values (operation modes). The diversity of produced operation modes however, see Figure 12, implies that early in training a complete spanning set of operation modes is derived, and the usage of their ratios is then learned and improved.

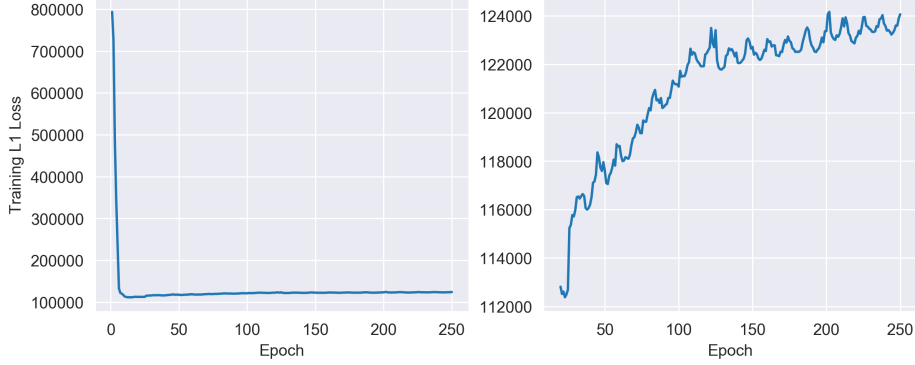


Figure 7: (Left) The loss per epoch of \mathbb{G}_{θ_1} as it is trained using Algorithm 7. On the left the loss over all epochs is shown. (Right) A magnified view of the loss starting from epoch 20.

6.2 Data Generation Results

As an interlude between results from $\mathbb{N}_{\{\theta_1, \theta_2\}}$, we outline the performance of our synthetic gas network data generation methods. Figure 8 (left) shows how our generated flow prognosis compares to that of historic real-world data. We see that Nodes A, B, and C are not technically entry or exits, but over historical data are dominated by a single orientation for each node. Specifically, Node C is the general entry, and Nodes A / B are the exits. In addition to the general orientation, we see that each node has significantly different ranges and distributions. These observations highlight the simplicity of our data generation methods, as we see near identical distributions for all nodes over the artificial data. We believe this calls for further research in prognosis generation methods. Figure 8 (right) shows our pressure prognosis compared to that of historic values. Unlike historic flow values, we observe little difference between historic pressure values of different nodes. This is supported by the optimal choices z_1^* over the historic data, see Figure 12, as in a large amount of cases compression is not needed and the network station is in bypass. Note that each corresponding entry (+) and exit (-) have identical pressure distributions due to the way they are constructed.

A further comparison of how our generated data compares to historic data can be seen in Figure 9. Here one can see the distribution of $\hat{f}(\mathbb{P}_{\pi}^{z_1^*})$ and $f(\mathbb{P}_{\pi}^{z_1^*})$ for the generated validation set, and $\hat{f}(\mathbb{P}_{\pi}^{z_1^*})$ and $f(\mathbb{P}_{\pi})$ for the real-world data. As expected, the distributions are different depending on whether the data is artificial or not. Our data generation was intended to be simplistic, and as

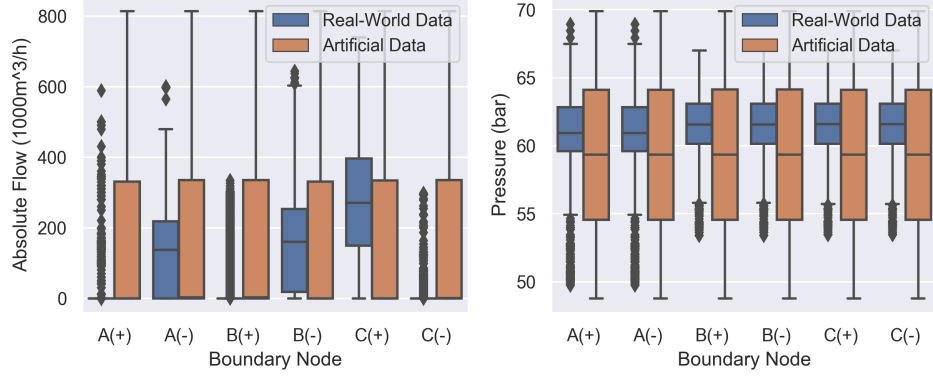


Figure 8: Comparison of generated flow (Left) / pressure (Right) value distributions per node vs. the distribution seen in real-world data.

independent as possible from the historic data. As such, the average scenario has optimal solution larger than that of any real-world data point. The performance of \mathbb{D}_{θ_2} is again clearly visible here, with $\hat{f}(\mathbb{P}_{\pi}^{\hat{z}_1})$ and $f(\mathbb{P}_{\pi}^{\hat{z}_1})$ being near identical over the artificial data, keeping in mind that these data points were never used in training. We see that this ability to generalise is relatively much worse on real-world data, mainly due to the the lower values of $f(\mathbb{P}_{\pi})$ over this data. Figure 9 (right) shows the results with log-scale axes to better highlight this disparity. It should be noted that the real-world instances with larger $f(\mathbb{P}_{\pi})$ are predicted quite well, and all real-world instances have an L1 distance between $\hat{f}(\mathbb{P}_{\pi}^{\hat{z}_1})$ and $f(\mathbb{P}_{\pi})$ that is small in terms of absolute differences.

6.3 Real-World Results

We now present results of our fully trained $\mathbb{N}_{\{\theta_1, \theta_2\}}$ applied to the 15 weeks of real-world data. Note that we had to remove 651 instances from our 9291 instances, as the warm-start resulted in an optimal solution value further away than the optimality tolerances we set. These instances have been kept in the graphics, but are marked and conclusions will not be drawn from them. We believe the problems with reproducibility are caused by the numeric difficulties in managing the pipe equality constraints.

Figure 10 shows the comparison of $f(\mathbb{P}_{\pi}^{\hat{z}_1})$ and $f(\mathbb{P}_{\pi})$. In a similar manner to \mathbb{D}_{θ_2} , we see that \mathbb{G}_{θ_1} struggles with instances where $f(\mathbb{P}_{\pi})$ is small. This is visible in the bottom left, where we see $f(\mathbb{P}_{\pi}^{\hat{z}_1})$ values much larger than $f(\mathbb{P}_{\pi})$ for like π . This comes as little surprise given the struggle of \mathbb{D}_{θ_2} with small $f(\mathbb{P}_{\pi})$ values. Drawing conclusions becomes more complicated for instances with larger $f(\mathbb{P}_{\pi})$ values, because the majority hit the time limit. We can clearly see however,



Figure 9: $\hat{f}(\mathbb{P}_\pi^{\hat{z}_1})$ for the validation set, and $\hat{f}(\mathbb{P}_\pi^{\hat{z}_1^*})$ for real-world data, compared to $f(\mathbb{P}_\pi^{\hat{z}_1})$ and $f(\mathbb{P}_\pi)$ respectively. Linear scale (Left) and log-scale (Right).



Figure 10: A comparison of $f(\mathbb{P}_\pi^{\hat{z}_1})$ and $f(\mathbb{P}_\pi)$ for all real-world data instances.

the value of our primal heuristic. There are many cases, those below the line $f(\mathbb{P}_\pi^{\hat{z}_1}) = f(\mathbb{P}_\pi)$, where our primal heuristic retrieves a better solution than the MIP solver does in one hour. Additionally, we see that no unsolved point above the line is very far from the line, showing that our primal heuristic produced a comparable, sometimes equivalent solution in a much shorter time frame. For a comparison of solve-times, see Table 2.

	Mean	Median	STD	Min	Max
$\mathbb{N}_{\{\theta_1, \theta_2\}}$ Inference Time (s)	0.009	0.008	0.001	0.008	0.017
Warmstarted \mathbb{P}_π Time (s)	100.830	9.380	421.084	0.130	3600.770
\mathbb{P}_π Time (s)	147.893	24.380	463.279	3.600	3601.280
$\mathbb{P}_\pi^{\hat{z}_1}$ + Warmstarted \mathbb{P}_π Time (s)	103.329	12.130	424.543	0.190	3726.110
$\mathbb{P}_\pi^{\hat{z}_1}$ Time (s)	2.499	1.380	12.714	0.060	889.380

Table 2: Solve time statistics for different solving strategies.

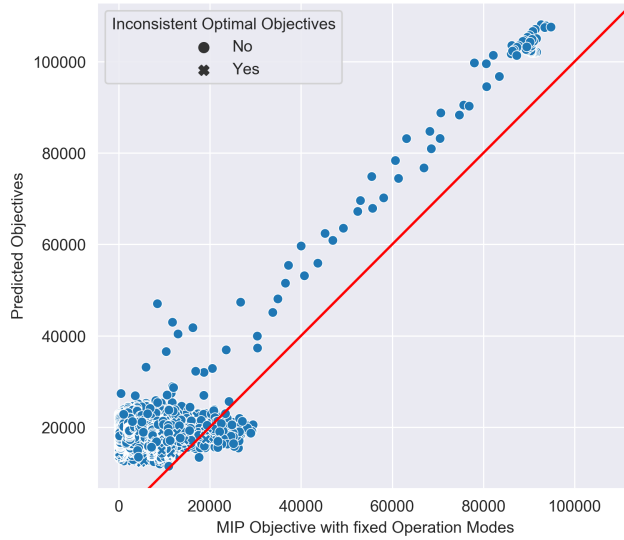


Figure 11: A comparison of $\hat{f}(\mathbb{P}_\pi^{\hat{z}_1})$ and $f(\mathbb{P}_\pi^{\hat{z}_1})$ for all real-world data instances.

Figure 11 shows the performance of the predictions $\hat{f}(\mathbb{P}_\pi^{\hat{z}_1})$ compared to $f(\mathbb{P}_\pi^{\hat{z}_1})$. Interestingly, \mathbb{D}_{θ_2} generally predicts $\hat{f}(\mathbb{P}_\pi^{\hat{z}_1})$ values slightly larger than $f(\mathbb{P}_\pi^{\hat{z}_1})$. We expect this for the smaller valued instances, as we know that \mathbb{D}_{θ_2} struggles with $f(\mathbb{P}_\pi^{\hat{z}_1})$ instances near 0, but the trend is evident for larger valued instances too. The closeness of the data points to the line $\hat{f}(\mathbb{P}_\pi^{\hat{z}_1}) = f(\mathbb{P}_\pi^{\hat{z}_1})$ show that \mathbb{D}_{θ_2} can adequately predict \hat{z}_1 solutions from \mathbb{G}_{θ_1} despite the change in data sets. Figure 10 showed that \mathbb{G}_{θ_1} successfully generalised to a new data set, albeit with difficulties around instances with $f(\mathbb{P}_\pi)$ valued near 0. From Figures 10 and 11,

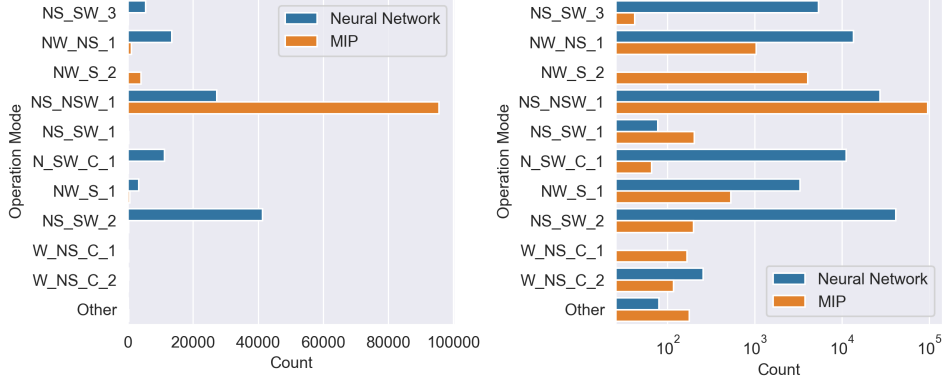


Figure 12: Frequency of operation mode choice by \mathbb{G}_{θ_1} compared to MIP solver for all real-world instances. (Left) Linear scale, and (Right) log scale.

	<i>NW_NS_1</i>	<i>NS_SW_2</i>	<i>N_SW_C_1</i>	<i>NS_NSW_1</i>	<i>W_NS_C_1</i>	<i>NS_SW_1</i>	<i>NW_S_2</i>	<i>NS_SW_3</i>	<i>W_NS_C_2</i>	<i>NW_S_1</i>	<i>Other</i>
NW_NS_1	884	22	0	9529	31	37	2436	4	24	397	82
NS_SW_2	48	102	1	40298	0	114	630	24	0	51	13
N_SW_C_1	0	27	65	11008	0	4	0	2	0	0	55
NS_NSW_1	41	29	0	26509	0	28	557	9	0	49	15
W_NS_C_1	0	0	0	0	0	0	0	0	0	0	0
NS_SW_1	0	0	0	76	0	1	0	0	0	0	0
NW_S_2	4	0	0	0	0	0	2	0	0	1	1
NS_SW_3	6	7	0	5220	0	7	108	1	0	4	5
W_NS_C_2	28	0	0	0	136	0	0	0	93	0	0
NW_S_1	30	11	0	2880	0	12	315	2	0	30	6
Other	0	1	0	78	0	0	0	0	0	0	1

Table 3: Operation Mode Correlation Matrix between \hat{z}_1 and z_1^* .

we can see that the entire $N_{\{\theta_1, \theta_2\}}$ generalises to unseen real-world instances, despite some generalisation loss.

We now compare the operation modes \hat{z}_1 , which are generated by \mathbb{G}_{θ_1} , and the z_1^* , which are produced by our MIP solver. To do so we use the following naming convention: We name the three pairs of boundary nodes N (north), S (south), and W (west). Using W_NS_C_2 as an example, we know that flow comes from W, and goes to N and S. The C in the name stands for active compression, and the final index is to differentiate between duplicate names. As seen in Figure 12, which plots the frequency of specific z_1 if they occurred more than 50 times, a single choice dominates z_1^* . This is interesting, because we expected there to be a lot of symmetry between z_1 , with the MIP solver selecting symmetric solutions with equal probability. For instance, take W_NS_C_1 and take W_NS_C_2.

$N_{\{\theta_1, \theta_2\}}$ only ever predicts W_NS_C.2, however with half the frequency the MIP solver selects each of them. This indicates that from the MIP’s point of view they are symmetric, and either can be chosen, while $N_{\{\theta_1, \theta_2\}}$ has recognised this and converged to a single choice. We can support this by analysing the data, where the difference in W_NS_C.1 and W_NS_C.2 is which compressor machine is used, with both machines being identical. This duplicate choice apparently does not exist in bypass modes however, where the uniqueness of z_1 , determined by valve states, results in different $f(\mathbb{P}_\pi^{z_1})$ values. It is observable then that for the majority of instances NS_NSW_1 is the optimal choice, and that $N_{\{\theta_1, \theta_2\}}$ has failed to identify its central importance. We believe this is due to the training method, where over generalisation to a single choice is strongly punished. For a comprehensive overview of the selection of operation modes and the correlation between \hat{z}_1 and z_1^* , we refer interested readers to Table 3.

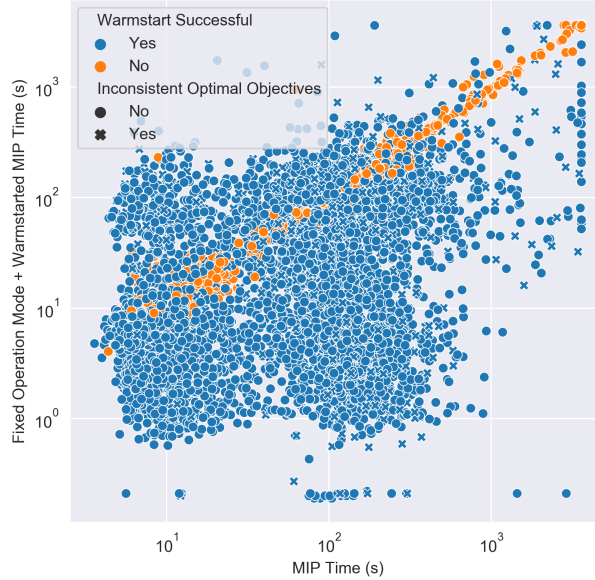


Figure 13: The combined running time of solving $\mathbb{P}_\pi^{\hat{z}_1}$, and solving a warm-started \mathbb{P}_π , compared to solving \mathbb{P}_π directly.

As discussed above, $N_{\{\theta_1, \theta_2\}}$ cannot reliably produce z_1^* . Nevertheless, it produces near-optimal \hat{z}_1 suggestions, which are still useful in a warm-start context, see Algorithm 6. The results of our warm-start algorithm are displayed in Figure 13. Our warm-start suggestion was successful 72% of the time, and the algorithm resulted in an average speed up of 60.5%. We use the shifted geometric mean with a shift of 1 for this measurement to avoid distortion by relative variations of the smaller valued instances. Especially surprising is that some instances that were previously unsolvable within the time-limit were easily solv-

able given the warm-start suggestion. In addition, many of the solvable but complicated instances are also solved near instantly with the warm-start suggestion. As such, we have created an effective primal heuristic that is both quick to run and beneficial in the context of locating a globally optimal solution.

7 Conclusion

In this paper, we presented a dual neural network design for generating decisions in a MIP, where the network is trained without ever solving the MIP with unfixed decision variables. The neural network is used as a primal heuristic, and then used to warm-start the MIP solver for the original problem. We proved the usefulness of our design on the transient gas transportation problem. While doing so we created methods for generating synthetic transient gas data for training purposes, reserving an unseen 9291 real-world instances for validation purposes. Despite some generalisation loss, our trained neural network results in a primal heuristic that takes on average 2.5s to run, and results in a 60.5% decrease in global optimal solution time when used in a warm-start context.

While we feel like our approach is an important step forward in neural network design and ML’s application to gas transport, we believe that there exists three primary directions for additional research. The first of which is to convert our approach into more traditional reinforcement learning, and then utilise policy gradient approaches [41]. The major hurdle to this approach is that much of the computation would be shifted online, requiring many more calls to solve the induced MIPs. This could be offset however, by using our technique to initialise the weights for such an approach, thereby avoiding early stage training difficulties with policy gradient approaches. The second is focused on the recent improvements in Graph Neural Networks [15]. Their ability to generalise to different input sizes would permit the creation of a single NN over multiple network stations or gas network topologies. Thirdly, there exists a large gap in the literature w.r.t data generation for transient gas networks. Improved methods are needed, which function at scale and result in real-world like data.

Finally, although we focused on the transient gas transportation problem, our approach can be generalised to arbitrary problem classes.

Acknowledgements

The work for this article was funded by UNSEEN (fund number 03EI1004D) and has been conducted in the Research Campus MODAL funded by the German Federal Ministry of Education and Research (BMBF) (fund numbers 05M14ZAM, 05M20ZBM)).

References

- [1] T. Achterberg. *Constraint integer programming*. PhD thesis, Technische Universität Berlin, 2007.
- [2] E. Balas. The Convex Hull of a Disjunctive Set. In *Disjunctive Programming*, pages 17–39. Springer International Publishing, Cham, 2018.
- [3] R. Baltean-Lugojan, P. Bonami, R. Misener, and A. Tramontani. Scoring positive semidefinite cutting planes for quadratic optimization via trained neural networks. *optimization-online preprint 2018/11/6943*, 2019.
- [4] J. Beliën, E. Demeulemeester, and B. Cardoen. A decision support system for cyclic master surgery scheduling with multiple objectives. *Journal of scheduling*, 12(2):147, 2009.
- [5] Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: a methodological tour d’horizon. *arXiv preprint arXiv:1811.06128*, 2018.
- [6] D. Bertsimas and B. Stellato. The voice of optimization. *arXiv preprint arXiv:1812.09991*, 2018.
- [7] D. Bertsimas and B. Stellato. Online mixed-integer optimization in milliseconds. *arXiv preprint arXiv:1907.02206*, 2019.
- [8] R. Burlacu, H. Egger, M. Groß, A. Martin, M. E. Pfetsch, L. Schewe, M. Sirvent, and M. Skutella. Maximizing the storage capacity of gas networks: a global minlp approach. *Optimization and Engineering*, 20(2):543–573, 2019.
- [9] Z. Chen, Y. Zhong, X. Ge, and Y. Ma. An actor-critic-based uav-bss deployment method for dynamic environments. *arXiv preprint arXiv:2002.00831*, 2020.
- [10] J.-Y. Ding, C. Zhang, L. Shen, S. Li, B. Wang, Y. Xu, and L. Song. Optimal solution predictions for mixed integer programs. *arXiv preprint arXiv:1906.09575*, 2019.
- [11] M. Etheve, Z. Alès, C. Bissuel, O. Juan, and S. Kedad-Sidhoum. Reinforcement learning for variable selection in a branch and bound algorithm. *arXiv preprint arXiv:2005.10026*, 2020.
- [12] J. Fang, Q. Zeng, X. Ai, Z. Chen, and J. Wen. Dynamic optimal energy flow in the integrated natural gas and electrical power systems. *IEEE Transactions on Sustainable Energy*, 9(1):188–198, 2017.
- [13] A. Ferber, B. Wilder, B. Dilina, and M. Tambe. Mipaal: Mixed integer program as a layer. *arXiv preprint arXiv:1907.05912*, 2019.
- [14] M. Fischetti and A. Lodi. Local branching. *Mathematical programming*, 98(1-3):23–47, 2003.

- [15] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 15554–15566, 2019.
- [16] I. Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.
- [17] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.
- [18] L. Gurobi Optimization. Gurobi optimizer reference manual, 2020.
- [19] H. Hanachi, C. Mechefske, J. Liu, A. Banerjee, and Y. Chen. Performance-based gas turbine health monitoring, diagnostics, and prognostics: A survey. *IEEE Transactions on Reliability*, 67(3):1340–1363, 2018.
- [20] W. E. Hart, C. D. Laird, J.-P. Watson, D. L. Woodruff, G. A. Hackebeil, B. L. Nicholson, and J. D. Siirola. *Pyomo—optimization modeling in python*, volume 67. Springer Science & Business Media, second edition, 2017.
- [21] W. E. Hart, J.-P. Watson, and D. L. Woodruff. Pyomo: modeling and solving mathematical programs in python. *Mathematical Programming Computation*, 3(3):219–260, 2011.
- [22] F. Hennings. Benefits and limitations of simplified transient gas flow formulations. In *Operations Research Proceedings 2017*, pages 231–237. Springer, 2018.
- [23] F. Hennings, L. Anderson, K. Hoppmann-Baum, M. Turner, and T. Koch. Controlling transient gas flow in real-world pipeline intersection areas. *Optimization and Engineering*, pages 1–48, 2020.
- [24] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [25] K. Hoppmann, F. Hennings, R. Lenz, U. Gotzes, N. Heinecke, K. Spreckelsen, and T. Koch. Optimal operation of transient gas transport networks. Technical report, Technical Report 19-23, ZIB, Takustr. 7, 14195 Berlin, 2019.
- [26] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [27] F. Kunz, M. Kendzioriski, W.-P. Schill, J. Weibezahn, J. Zepter, C. R. von Hirschhausen, P. Hauser, M. Zech, D. Möst, S. Heidari, et al. Electricity, heat, and gas sector data for modeling the german system. Technical report, DIW Data Documentation, 2017.
- [28] D. Masti and A. Bemporad. Learning binary warm starts for multiparametric mixed-integer quadratic programming. In *2019 18th European Control Conference (ECC)*, pages 1494–1499. IEEE, 2019.

- [29] M. MohamadiBaghmolaei, M. Mahmoudy, D. Jafari, R. MohamadiBaghmolaei, and F. Tabkhi. Assessing and optimization of pipeline system performance using intelligent systems. *Journal of Natural Gas Science and Engineering*, 18:64–76, 2014.
- [30] S. Moritz. *A mixed integer approach for the transient case of gas network optimization*. PhD thesis, Technische Universität, 2007.
- [31] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.
- [32] M. Petkovic, Y. Chen, I. Gamrath, U. Gotzes, N. S. Hadjidimitriou, J. Zittel, and T. Koch. A hybrid approach for high precision prediction of gas flows. Technical Report 19-26, ZIB, Takustr. 7, 14195 Berlin, 2019.
- [33] D. Pfau and O. Vinyals. Connecting generative adversarial networks and actor-critic methods. *arXiv preprint arXiv:1610.01945*, 2016.
- [34] A. Pourfard, H. Moetamedzadeh, R. Madoliat, and E. Khanmirza. Design of a neural network based predictive controller for natural gas pipelines in transient state. *Journal of Natural Gas Science and Engineering*, 62:275–293, 2019.
- [35] R. Z. Ríos-Mercado and C. Borraz-Sánchez. Optimization problems in natural gas transportation systems: A state-of-the-art review. *Applied Energy*, 147:536–555, 2015.
- [36] D. B. Rubin. The bayesian bootstrap. *The annals of statistics*, pages 130–134, 1981.
- [37] R. Ruiz, C. Maroto, and J. Alcaraz. A decision support system for a real vehicle routing problem. *European Journal of Operational Research*, 153(3):593–606, 2004.
- [38] L. N. Smith. Cyclical learning rates for training neural networks, 2017.
- [39] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.
- [40] Y. Tang, S. Agrawal, and Y. Faenza. Reinforcement learning for integer programming: Learning to cut. *arXiv preprint arXiv:1906.04859*, 2019.
- [41] P. S. Thomas and E. Brunskill. Policy gradient methods for reinforcement learning with function approximation and action-dependent baselines. *arXiv preprint arXiv:1706.06643*, 2017.
- [42] E. Wong and J. Z. Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. *arXiv preprint arXiv:1711.00851*, 2017.

- [43] I. Yueksel Erguen, J. Zittel, Y. Wang, F. Hennings, and T. Koch. Lessons learned from gas network data preprocessing. Technical report, Technical Report 20-13, ZIB, Takustr. 7, 14195 Berlin, 2020.
- [44] G. Zarpellon, J. Jo, A. Lodi, and Y. Bengio. Parameterizing branch-and-bound search trees to learn branching policies. *arXiv preprint arXiv:2002.05120*, 2020.

A Appendix

Algorithm 8: Prepare Discriminator Training Data

Input: Neural network $N_{\{\theta_1, \theta_2\}}$, labelled_data
Result: Data for training \mathbb{D}_{θ_2}
new_labelled_data = [] ;
for $i = 0$; $i < num_data_new$ **do**
 initial_state = Uniformly select from generated offline data ;
 flow_forecast, pressure_forecast = Boundary Prognosis Generator()^a;
 $\pi = (\text{flow_forecast}, \text{pressure_forecast}, \text{initial_state})$;
 $\hat{z}_1, \hat{f}(\mathbb{P}_{\pi}^{\hat{z}_1}) = N_{\{\theta_1, \theta_2\}}(\pi)$;
 $f(\mathbb{P}_{\pi}^{\hat{z}_1}) = \text{solve } \mathbb{P}_{\pi}^{\hat{z}_1}$;
 new_labelled_data.append($\pi, \hat{z}_1, f(\mathbb{P}_{\pi}^{\hat{z}_1})$) ;
end
return concatenate(labelled_data[-num_data_old:], new_labelled_data)

^aSee Algorithm 1

Algorithm 9: Discriminator Pretraining

Input: Discriminator \mathbb{D}_{θ_2} , data
optimizer = Adam(learning_rate, weight_decay);
dataloader = DataLoader(data, batch_size, shuffle=True);
lr_scheduler=ReduceLROnPlateau();
for $i = 0$; $i < num_epochs$ **do**
 Discriminator Training Loop(\mathbb{D}_{θ_2} , dataloader, optimizer)¹;
 lr_scheduler.step();
end

Algorithm 10: Generator Training

Input: Neural network $\mathbb{N}_{\{\theta_1, \theta_2\}}$
Result: Average loss in training
optimizer = Adam();
lr_scheduler=cyclicLR^a(max_lr, base_lr, step_size_up);
data = [] ;
for $i = 0; i < num_scenarios$ **do**
 initial_state = Uniformly select from generated offline data ;
 flow_forecast, pressure_forecast = Boundary Prognosis Generator()^b;
 $\pi = (\text{flow_forecast}, \text{pressure_forecast}, \text{initial_state})$;
 data.append(π) ;
end
dataloader = DataLoader(data, batch_size, shuffle=True);
for $batch$ in dataloader **do**
 optimizer.zero_grad();
 losses = [];
 $\hat{z}_1, \hat{f}(\mathbb{P}_{\pi}^{\hat{z}_1}) = \mathbb{N}_{\{\theta_1, \theta_2\}}$ (batch);
 loss = L1Loss($\hat{f}(\mathbb{P}_{\pi}^{\hat{z}_1})$, 0);
 loss.backward();
 optimizer.step();
 lr_scheduler.step();
 losses.append(loss);
end
Result: mean(losses)

^aIntroduced by Smith in [38], see also
pytorch.org/docs/stable/optim.html#torch.optim.lr_scheduler.CyclicLR.

^bSee Algorithm 1

Algorithm 11: Discriminator Training Loop

Input: Discriminator \mathbb{D}_{θ_2} , dataloader, optimizer
for $batch$ in dataloader **do**
 optimizer.zero_grad();
 $\hat{f}(\mathbb{P}_{\pi}^{z_1}) = \mathbb{D}_{\theta_2}$ (batch);
 loss = L1Loss($\hat{f}(\mathbb{P}_{\pi}^{z_1})$, $f(\mathbb{P}_{\pi}^{z_1})$);
 loss.backward();
 optimizer.step();
end

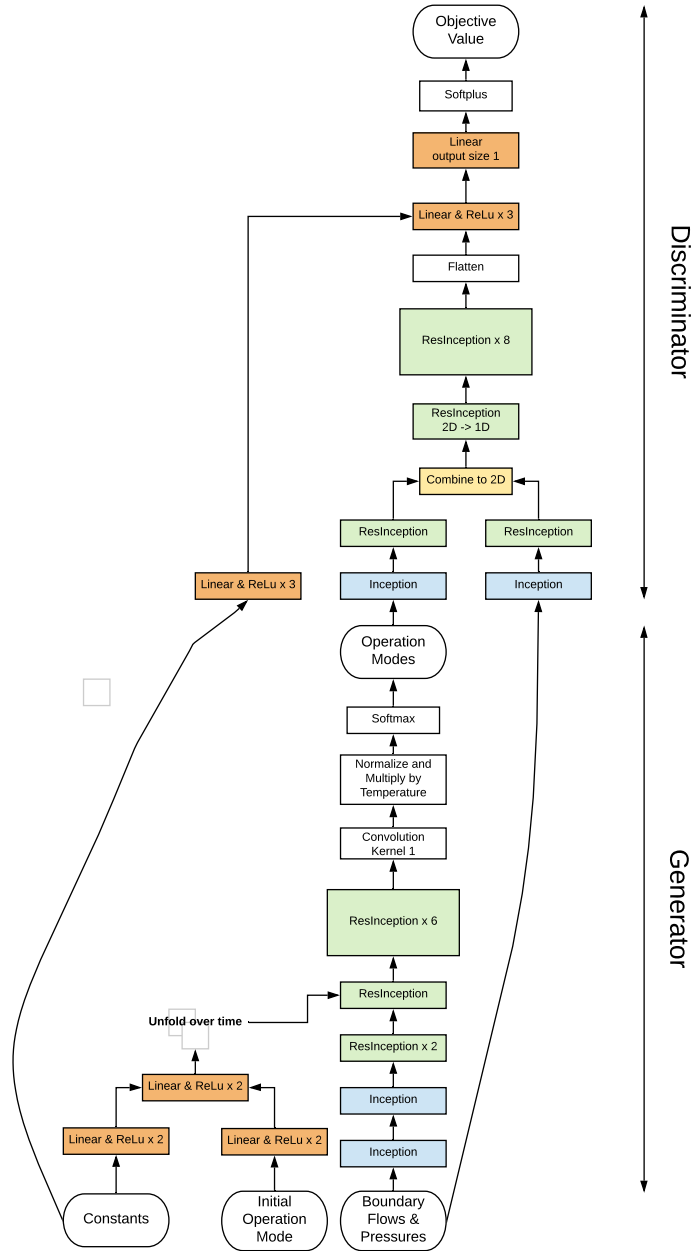


Figure 14: Neural Network Architecture

Parameter	Method	Value
batch_size	Algorithm 9	2048
num_epochs	Algorithm 9	500
learning_rate	Algorithm 9 / Adam	0.005
weight_decay	Algorithm 9 / Adam	5e-06
batch_size	Algorithm 10	2048
max_lr	Algorithm 10 / CyclicLR	0.0005
base_lr	Algorithm 10 / CyclicLR	5e-06
step_size_up	Algorithm 10 / CyclicLR	10000
num_scenarios	Algorithm 10	3200000
num_data_new	Algorithm 8	2048
num_data_old	Algorithm 8	8192
num_epochs	Algorithm 7	10
num_generator_epochs	Algorithm 7	25
num_discriminator_epochs	Algorithm 7	25
stopping_loss_discriminator	Algorithm 7	$3 * 1022.5^1$
stopping_loss_generator	Algorithm 7	$0.9 * 121848.27^2$
num_prelabelled	Algorithm 7 / mix_in_prelabelled_data	8192
ratio_test	Algorithm 7 / split_data	0.1
learning_rate	Algorithm 7 / Adam	0.001
weight_decay	Algorithm 7 / Adam	5e-06
patience	Algorithm 7 / ReduceLROnPlateau	2
factor	Algorithm 7 / ReduceLROnPlateau	0.5

¹ 1022.5 was the test loss after initial discriminator training.

² 121848.27 represents the average $\hat{f}(\mathbb{P}_\pi^{\mathcal{Z}_1})$ value over our artificial data.

Table 4: Parameters for training.

Parameter	Value
TimeLimit	3600 (s)
FeasibilityTol	1e-6
MIPGap	1e-4
MIPGapAbs	1e-2
NumericFocus	3

Table 5: Parameters for MIP solving.

	Parameters	Inception Blocks	Small Inception Blocks
Neural Network	1,701,505	13	12
Generator	1,165,576	13	0
Discriminator	535,929	0	12
Inception Block	87,296	-	-
Small Inception Block	27,936	-	-

Table 6: Number of parameters in the neural network and submodules.