Anderson, Lovis; Turner, Mark; Koch, Thorsten

Generative deep learning for decision making in gas networks

Mathematical Methods of Operations Research

Mathematical Methods of Operations Research (2022) 95:503–532
https://doi.org/10.1007/s00186-022-00777-x

ORIGINAL ARTICLE

Generative deep learning for decision making in gas networks

Lovis Anderson[2] · **Mark Turner**[1,2] · **Thorsten Koch**[1,2]

Received: 28 January 2021 / Revised: 3 February 2022 / Accepted: 23 February 2022 /
Published online: 19 April 2022

Abstract

A decision support system relies on frequent re-solving of similar problem instances. While the general structure remains the same in corresponding applications, the input parameters are updated on a regular basis. We propose a generative neural network design for learning integer decision variables of mixed-integer linear programming (MILP) formulations of these problems. We utilise a deep neural network discriminator and a MILP solver as our oracle to train our generative neural network. In this article, we present the results of our design applied to the transient gas optimisation problem. The trained generative neural network produces a feasible solution in 2.5s, and when used as a warm start solution, decreases global optimal solution time by 60.5%.

Keywords Mixed-integer programming · Deep learning · Primal heuristic · Gas networks · Generative modelling

The work for this article has been conducted in the Research Campus MODAL funded by the German Federal Ministry of Education and Research (BMBF) (fund numbers 05M14ZAM, 05M20ZBM), and was supported by the German Federal Ministry of Economic Affairs and Energy (BMWi) through the project UNSEEN (fund no 03EI1004D): Bewertung der Unsicherheit in linear optimierten Energiesystem-Modellen unter Zuhilfenahme Neuronaler Netze.

✉ Mark Turner
turner@zib.de

Lovis Anderson
anderson@zib.de

Thorsten Koch
koch@zib.de

[1] Chair of Software and Algorithms for Discrete Optimization, Institute of Mathematics, Technische Universität Berlin, Straße des 17. Juni 135, 10623 Berlin, Germany

[2] Applied Algorithmic Intelligence Methods Department, Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany

1 Introduction

Mixed-Integer Linear Programming (MILP) is concerned with the modelling and solving of problems from discrete optimisation. These problems can represent real-world scenarios, where discrete decisions can be appropriately captured and modelled by integer variables. In real-world scenarios a MILP model is rarely solved only once. More frequently, the same model is used with varying data to describe different instances of the same problem, which are solved on a regular basis. This holds true in particular for decision support systems, which can utilise MILP to provide real time optimal decisions on a continual basis, see (Beliën et al. 2009) and (Ruiz et al. 2004) for examples in nurse scheduling and vehicle routing. The MILPs that these decision support systems solve have identical structure due to both their underlying application and 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 be used in a primal heuristic by solving the induced subproblem following variable fixations. Additionally, the result of the primal heuristic can be used in a warm start context to help improve solver performance in achieving global optimality. We demonstrate the performance of our neural network (NN) design on the transient gas optimisation problem (Ríos-Mercado and Borraz-Sánchez 2015), specifically on real-world instances embedded in day ahead decision support systems.

The design of our framework is inspired by the recent development of Generative Adversarial Networks (GANs) (Goodfellow 2016). Our design consists of two NNs, a generator and a discriminator. The generator is responsible for generating binary decision values, while the discriminator is tasked with predicting the optimal objective function value of the reduced MILP that arises after fixing these binary variables to their generated values. Our NN design and its application to transient gas network MILP formulations is an attempt to integrate Machine Learning (ML) into the MILP solving process. This integration has recently received an increased focus (Tang et al. 2019; Bertsimas and Stellato 2019; Gasse et al. 2019), which has been encouraged by the success of ML integration into other aspects of combinatorial optimisation, see (Bengio et al. 2018) for a thorough overview.

The paper is structured as follows: Sect. 2 contains an overview of the literature with comparisons to our work. In Sect. 3, we introduce our main contribution, a new generative NN design for learning binary variables of parametric MILPs. Afterward, we outline a novel data generation approach for generating synthetic gas transport instances in Sect. 4. Section 5 outlines the exact training scheme for our new NN and how our framework can be used to warm start MILPs. Finally, in Sect. 6, we show and discuss the results of our NN design on real-world gas transport instances. This represents a major contribution, as the trained NN generates a primal solution in 2.5s and via warm start reduces solution time to achieve global optimality by 60.5%.

2 Background and related work

As mentioned in the introduction, the intersection of MILP 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 Goodfellow et al. (2016), and for MILP to Achterberg (2007). 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 MILP applied to ML instead of the orientation we consider, see (Wong and Kolter 2017) for an interesting example.

Firstly, we summarise applications of ML to adjacent areas of the MILP solving process. Gasse et al. (2019) creates a method for encoding MILP 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. (2020) take a different approach, and use a NN design that incorporates the branch-and-bound tree state directly. In doing so, they show that information contained in the global branch-and-bound tree state is an important factor in variable selection. Furthermore, their publication is one of the few to present results on heterogeneous instances. Etheve et al. (2020) show a successful implementation of reinforcement learning for variable selection. Tang et al. (2019) 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 fidelity and complexity in the field of transient gas optimisation, and there is no standard model for transient gas transport problems. Moritz (2007) presents a piecewise linear MILP approach to the transient gas transport problem, (Burlacu et al. 2019) a nonlinear approach with a novel discretisation scheme, and (Hennings et al. 2020) and (Hoppmann et al. 2019) a linearised approach. For the purpose of our experiments, we use the model of Hennings et al. (2020), which uses linearised equations and focuses on gas subnetworks with many controllable elements. The current research of ML in gas transport is still in the early stages. Pourfard et al. (2019) use a dual NN design to perform online calculations of a compressors operating point to avoid re-solving the underlying model. The approach constrains itself to continuous variables and experimental results are presented for a gunbarrel gas network. MohamadiBaghmolaei et al. (2014) 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. More often, ML has been used in fields closely related to gas transport, as in Hanachi et al. (2018), with ML used to track the degradation of compressor performance, and in Petkovic et al. (2019) to forecast demand values at the boundaries of the gas network. For a more

complete overview of the transient gas literature, we refer readers to Ríos-Mercado and Borraz-Sánchez (2015).

Our framework, which predicts the optimal objective value of an induced sub-MILP, can be considered similar to Baltean-Lugojan et al. (2019) in what it predicts and similar to Ferber et al. (2019), in how it works. In the first paper (Baltean-Lugojan et al. 2019), a NN 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 MILP formulation. In the second paper (Ferber et al. 2019), a technique is developed that performs backward passes directly through a MILP. It does this by solving MILPs exclusively with cutting planes, and then receiving gradient information from the KKT conditions of the final linear program. This application of a NN, which produces input to the MILP, 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 MILP directly, and that we generate variable values instead of parameter values with our generator.

While our framework is heavily inspired from GANs (Goodfellow 2016), it is also similar to actor-critic algorithms, see (Pfau and Vinyals 2016). These algorithms have shown success for variable generation in MILP, and are notably different in that they sample from a generated distribution for downstream decisions instead of always taking the decision with highest probability. Recently, (Chen et al. 2020) generated a series of coordinates for a set of UAVs using an actor-critic based algorithm, where these coordinates were continuous variables in a Mixed-Integer Non-Linear Program (MINLP) formulation. The independence of separable subproblems 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 and Vinyals (2016).

Finally, we summarise existing research that also deals with the generation of decision variable values for Mixed-Integer Programs. Bertsimas and Stellato (2018, 2019) attempt to learn optimal solutions of parametric MILPs and Mixed-Integer Quadratic Programs (MIQPs), which involves both outputting all integer decision variable values and the active set of constraints. They mainly use optimal classification trees in Bertsimas and Stellato (2018) and NNs in Bertsimas and Stellato (2019). Their aim is tailored towards smaller problems classes, where speed is an absolute priority and parameter value changes are limited. Masti and Bemporad (2019) 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 goal of obtaining a primal heuristic is similar to ours, and while their design is much simpler, it has been shown to work effectively on very small problems. Our improvement over this design is our non-reliance on labelled optimal solutions, which are needed for binary cross entropy. Ding et al. (2019) present a GCNN design which is an extension of Gasse et al. (2019), and use it to generate binary decision variable values. Their contributions are a tripartite graph encoding of MILP instances, and the inclusion of their aggregated generated values as branching decisions in the branch-and-bound tree, both in an exact approach and in an approximate approach with local branching (Fischetti and Lodi 2003). Very recently, (Nair et al. 2020) combined the branching approach of Gasse et al. (2019) with a novel neural diving approach, in which integer variable values are generated. They use a GCNN for generating both branching decisions and integer variables values.

Different to our generator-discriminator based approach, they generate values directly from a learned distribution, which is based on an energy function that incorporates resulting objective values.

3 The solution framework

We begin by formally defining both a MILP and a NN. Our definition of a MILP is an extension of more traditional formulations, see (Achterberg 2007), but still encapsulates general instances.

Definition 1 Let $\pi \in \mathbb{R}^p$ be a vector of problem-defining parameters, where $p \in \mathbb{Z}_{\geq 0}$. We use π as a subscript to indicate that a variable is parameterised by π. We call $\mathbb{P}_\pi$ a MILP parameterised by π.

$$\mathbb{P}_\pi := \begin{cases} \min & c_1^\mathsf{T} x_1 + c_2^\mathsf{T} x_2 + c_3^\mathsf{T} z_1 + c_4^\mathsf{T} z_2 \\ \text{s.t.} & A_\pi \begin{bmatrix} x_1 \\ x_2 \\ z_1 \\ z_2 \end{bmatrix} \leq b_\pi \\ & c_k \in \mathbb{R}^{n_k}, k \in \{1,2,3,4\}, A_\pi \in \mathbb{R}^{m \times n}, b_\pi \in \mathbb{R}^m \\ & x_1 \in \mathbb{R}^{n_1}, x_2 \in \mathbb{R}^{n_2}, z_1 \in \{0,1\}^{n_3}, z_2 \in \{0,1\}^{n_4}, n_k \in \mathbb{Z}_{\geq 0} \end{cases}$$

Furthermore let $\Sigma \subset \mathbb{R}^p$ be a set of valid problem-defining parameters. We then call $\{\mathbb{P}_\pi \mid \pi \in \Sigma\}$ a problem class for Σ. Lastly, we denote the optimal objective function value of $\mathbb{P}_\pi$ by $f(\mathbb{P}_\pi)$.

Note that the explicit parameter space Σ is usually unknown, but we assume in the following to have access to a random variable Π that samples from Σ. In addition, note that c_i and n_i are not parameterised by π, and as such the objective function and variable dimensions do not change between scenarios. In Definition 3 we define an oracle NN $\mathbb{G}_{\theta_1}$, which predicts a subset of the binary variables of $\mathbb{P}_\pi$, namely z_1. Additionally, the continuous variables x_2 are separated in order to differentiate the slack variables in our example, which we will introduce in Sect. 4.

We now provide a simple definition for feed forward NNs. For a larger variety of definitions, see (Goodfellow et al. 2016).

Definition 2 Let N_θ be defined by:

$$\begin{aligned} N_\theta : \mathbb{R}^{|a_1|} \to \mathbb{R}^{|a_{k+1}|}; \quad a_1 \to N_\theta(a_1) = a_{k+1} \\ h_i : \mathbb{R}^{|a_i|} \to \mathbb{R}^{|a_i|} \quad \forall i \in \{2, ..., k+1\} \\ a_{i+1} := h_{i+1}(W_i a_i + b_i) \quad \forall i \in \{1, ..., k\} \\ W_i \in \mathbb{R}^{|a_{i+1}| \times |a_i|}, \quad b_i \in \mathbb{R}^{|a_{i+1}|} \quad \forall i \in \{1, ..., k\} \end{aligned} \tag{1}$$

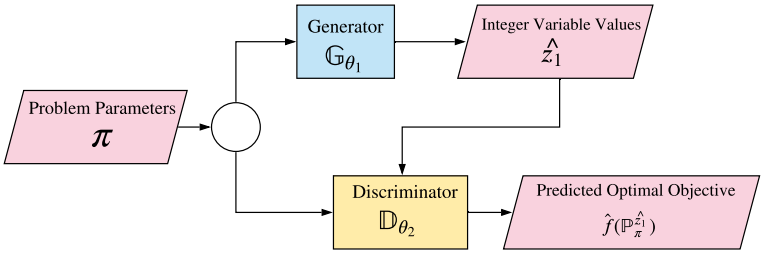


Fig. 1 The general design of $\mathbb{N}_{\{\theta_1,\theta_2\}}$

We then call N_θ a *k-layer feed forward NN*. Here, θ is the vector of all weights W_i and biases b_i of the NN. The functions h_i are non-linear element-wise functions, called activation functions. Additionally, a_i, b_i, and W_i are tensors for all $i \in \{1, \ldots, k\}$.

Definition 3 For a problem class $\{\mathbb{P}_\pi \mid \pi \in \Sigma\}$, let the *generator* $\mathbb{G}_{\theta_1}$ be a NN predicting z_1, and the *discriminator* $\mathbb{D}_{\theta_2}$ be a NN predicting $f(\mathbb{P}_\pi)$ for $\pi \in \Sigma$, i.e.

$$\begin{aligned} \mathbb{G}_{\theta_1} &: \mathbb{R}^p \to (0,1)^{n_3} \\ \mathbb{D}_{\theta_2} &: (0,1)^{n_3} \times \mathbb{R}^p \to \mathbb{R} \end{aligned} \tag{2}$$

Furthermore, a forward pass of both $\mathbb{G}_{\theta_1}$ and $\mathbb{D}_{\theta_2}$ is defined as follows:

$$\hat{z_1} := \mathbb{G}_{\theta_1}(\pi) \tag{3}$$

$$\hat{f}(\mathbb{P}_\pi^{\hat{z_1}}) := \mathbb{D}_{\theta_2}(\hat{z_1}, \pi) \tag{4}$$

The hat notation is used to denote quantities that were approximated by a NN. We use superscript notation to create the following instances:

$$\mathbb{P}_\pi^{\hat{z_1}} := \mathbb{P}_\pi \quad \text{s.t.} \quad z_1 = [\hat{z_1}] \tag{5}$$

The additional notation of the square brackets around $\hat{z_1}$, refers to the rounding of values from the range (0, 1) to {0, 1}, which is required as the variable values must be binary.

The goal of this framework is to generate good initial solution values $\hat{z_1}$, which lead to an induced sub-MILP, $\mathbb{P}_\pi^{\hat{z_1}}$, whose optimal solution is a good feasible solution to the original problem $\mathbb{P}_\pi$. Further, the idea is to use this feasible solution as a first incumbent for warm starting the solution process of $\mathbb{P}_\pi$. To ensure feasibility for all choices of z_1, we divide the continuous variables into two sets, x_1 and x_2, as seen in Definition 1. The variables x_2 are potential slack variables that ensure all generated decisions result in feasible $\mathbb{P}_\pi^{\hat{z_1}}$ instances, and are penalised in the objective. We now describe the design of $\mathbb{G}_{\theta_1}$ and $\mathbb{D}_{\theta_2}$.

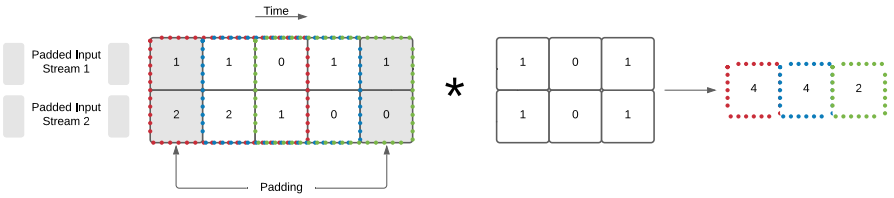


Fig. 2 Method of merging two 1-D input streams

3.1 Generator and discriminator design

$\mathbb{G}_{\theta_1}$ and $\mathbb{D}_{\theta_2}$ are NNs whose structure is inspired by Goodfellow (2016), as well as both inception blocks and residual NNs, which have greatly increased large-scale model performance (Szegedy et al. 2017). We use the block design Resnet-v2 from Szegedy et al. (2017), see Fig. 3, albeit with a modification that primarily uses 1-D convolutions, with that dimension being time. Additionally, we separate initial input streams by their characteristics, and when joining two streams use 2-D convolutions. These 2-D convolutions reduce the data back to 1-D, see Fig. 2 for an visualisation of this process. The final layer of $\mathbb{G}_{\theta_1}$ contains a softmax activation function with temperature. As the softmax temperature parameter increases, this activation function's output approaches a one-hot vector encoding. The final layer of $\mathbb{D}_{\theta_2}$ contains a softplus activation function. All other intermediate layers use the ReLU activation function. We refer readers to Goodfellow et al. (2016) for a thorough overview of deep learning, and to Fig. 14 in Appendix 1 for our complete design.

For a vector $x = (x_1, \cdots, x_n)$, the softmax function with temperature $T \in \mathbb{R}$, σ_1, the ReLU function, σ_2, and the softplus function with parameter $\beta \in \mathbb{R}$, σ_3, are defined as:

$$\sigma_1(x_i, T) := \frac{\exp(T x_i)}{\sum_{j=1}^{n} \exp(T x_j)} \tag{6}$$

$$\sigma_2(x_i) := \max(0, x_i) \tag{7}$$

$$\sigma_3(x_i, \beta) := \frac{1}{\beta} \log(1 + \exp(\beta x_i)) \tag{8}$$

We compose $\mathbb{G}_{\theta_1}$ and $\mathbb{D}_{\theta_2}$ to make $\mathbb{N}_{\{\theta_1,\theta_2\}}$. The definition of this composition is given in (9), and a visualisation in Fig. 1.

$$\mathbb{N}_{\{\theta_1,\theta_2\}}(\pi) := \mathbb{D}_{\theta_2}(\mathbb{G}_{\theta_1}(\pi), \pi) \tag{9}$$

3.2 Interpretations

In a similar manner to GANs and actor-critic algorithms, see (Pfau and Vinyals 2016), the design of $\mathbb{N}_{\{\theta_1,\theta_2\}}$ has a bi-level optimisation interpretation, see (Dempe 2002) for an overview of bi-level optimisation. Here we list the explicit objectives of both $\mathbb{G}_{\theta_1}$ and $\mathbb{D}_{\theta_2}$, and how their loss functions represent these objectives.

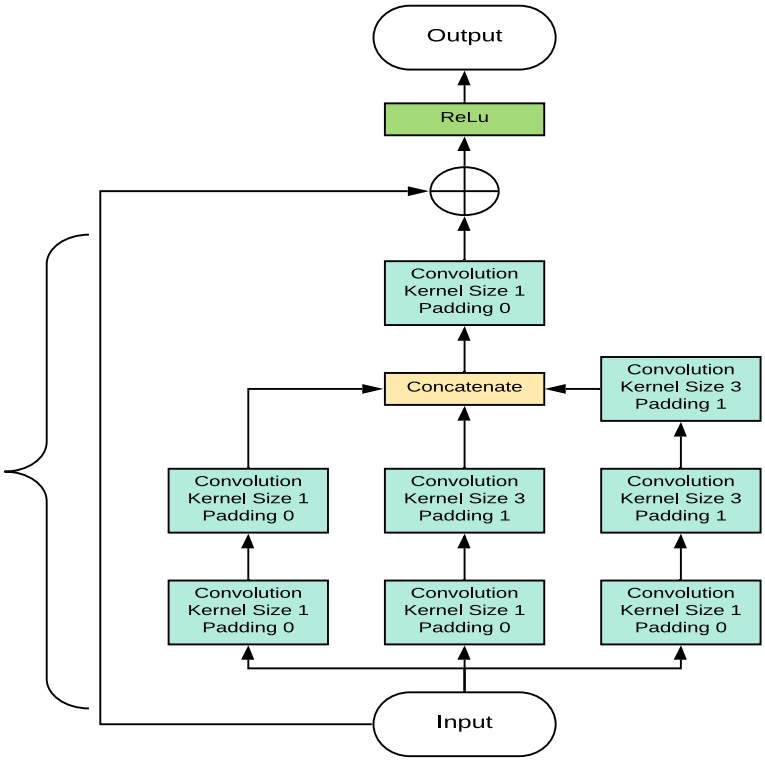


Fig. 3 1-D Resnet-v2 Block Design

The objective of $\mathbb{D}_{\theta_2}$ is to predict $f(\mathbb{P}_{\pi}^{\hat{z_1}})$, the optimal induced objective value of $\mathbb{P}_{\pi}^{\hat{z_1}}$. Its loss function is thus:

$$L(\theta_2, \pi) := \left|\mathbb{D}_{\theta_2}(\mathbb{G}_{\theta_1}(\pi), \pi) - f(\mathbb{P}_{\pi}^{\mathbb{G}_{\theta_1}(\pi)})\right| \tag{10}$$

The objective of $\mathbb{G}_{\theta_1}$ is to minimise the induced prediction of $\mathbb{D}_{\theta_2}$. Its loss function is thus:

$$L'(\theta_1, \pi) := \mathbb{D}_{\theta_2}(\mathbb{G}_{\theta_1}(\pi), \pi) \tag{11}$$

The corresponding bi-level optimisation problem can then be viewed as:

$$\begin{aligned} \min_{\theta_1} \quad & \mathbb{E}_{\pi\sim\Pi}[\mathbb{D}_{\theta_2}(\mathbb{G}_{\theta_1}(\pi),\pi)] \\ \text{s.t.} \quad \min_{\theta_2} \quad & \mathbb{E}_{\pi\sim\Pi}[|\mathbb{D}_{\theta_2}(\mathbb{G}_{\theta_1}(\pi),\pi) - f(\mathbb{P}_\pi^{\mathbb{G}_{\theta_1}(\pi)})|] \end{aligned} \tag{12}$$

3.3 Training method

For effective training of $\mathbb{G}_{\theta_1}$, a capable $\mathbb{D}_{\theta_2}$ is needed. We therefore pre-train $\mathbb{D}_{\theta_2}$. The following loss function, which replaces $\mathbb{G}_{\theta_1}(\pi)$ with synthetic z_1 values in (10), is used for this pre-training:

$$L''(\theta_2,\pi) := \left|\mathbb{D}_{\theta_2}(z_1,\pi) - f(\mathbb{P}_\pi^{z_1})\right| \tag{13}$$

However, performing this initial training requires generating instances of $\mathbb{P}_\pi^{z_1}$, and is therefore done in a supervised manner offline manner on synthetic data.

After the initial training of $\mathbb{D}_{\theta_2}$, we train $\mathbb{G}_{\theta_1}$ as a part of $\mathbb{N}_{\{\theta_1,\theta_2\}}$, using samples $\pi \sim \Pi$, the loss function (11), and fixed θ_2. The issue of $\mathbb{G}_{\theta_1}$ outputting continuous values for $\hat{z_1}$ is overcome by the choice of the final activation function of $\mathbb{G}_{\theta_1}$. The softmax with temperature (6) 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 the completion of both initial trainings, we alternatingly train both NNs using updated loss functions in the following way:

- $\mathbb{D}_{\theta_2}$ training:
  - As in the initial training, using loss function (13).
  - In an online fashion, using predictions from $\mathbb{G}_{\theta_1}$ and loss function (10).
- $\mathbb{G}_{\theta_1}$ training:
  - As explained above with loss function (11).

Our design allows the loss to be back-propagated through $\mathbb{D}_{\theta_2}$ and distributed to the individual nodes of the final layer of $\mathbb{G}_{\theta_1}$ that correspond to z_1. This is largely different to other methods, many of which rely on using loss against optimal solutions of $\mathbb{P}_\pi$, see (Masti and Bemporad 2019; Ding et al. 2019). Our advantage over these is that the contribution to $\hat{f}(\mathbb{P}_\pi^{\hat{z_1}})$ of each predicted decision $\hat{z_1}$ can be calculated. Additionally, we believe that it makes our generated suboptimal solutions more likely to be near optimal. We believe this because the NN is trained to minimise a predicted objective rather than copy previously observed optimal solutions.

4 The gas transport model and data generation

To evaluate the performance of our approach, we test our framework on the transient gas optimisation problem, see (Ríos-Mercado and Borraz-Sánchez 2015) for an overview of the problem and associated literature. This problem is difficult to solve as it combines a transient flow problem with complex combinatorics representing switching decisions. The natural modelling of transient gas networks as time-expanded networks lends itself well to our framework, however, due to the static underlying gas network and repeated constraints at each time step. In this section we summarise important aspects of our MILP formulation, and outline our methods for generating artificial gas transport data.

4.1 The gas transport model

We use the description of transient gas networks by Hennings et al. (2020). This model contains *operation modes*, which are binary decisions corresponding to the z_1 of Definition 1. Exactly one operation mode is selected each time step, and this decision decides on the discrete states of all controllable elements in the gas network for that time step. We note that we deviate slightly from the model by Hennings et al. (2020), and do not allow the inflow over a set of entry-exits to be freely distributed according to which group they belong. This is an important distinction as each single exit-entry in our model has a complete forecast.

The model by Hennings et al. (2020) contains slack variables that change the pressure and flow demand scenarios at entry-exits. These slack variables are represented by x_2 in Definition 1, and because of their existence we have yet to find an infeasible instance $\mathbb{P}_\pi^{z_1}$ for any choice of z_1. We believe that infeasible scenarios can be induced with sufficiently small time steps, but this is not the case in our experiments. The slack variables x_2 are penalised in the objective.

4.2 Data generation

In this subsection we outline our methods for generating synthetic transient gas instances for training purposes, i.e. generating $\pi \sim \Pi$ and artificial z_1 values. Section 4.2.1 introduces a novel method for generating balanced demand scenarios, followed by Sect. 4.2.2 that outlines how to generate operation mode sequences. Afterward, Sect. 4.2.3 presents an algorithm, which generates initial states of a gas network. These methods are motivated by the lack of available gas network data, see (Yuksel Erguen et al. 2020; Kunz et al. 2017), and the need for large amounts of data to train our NN.

4.2.1 Boundary forecast generation

Let $d_{v,t} \in \mathbb{R}$, be the flow demand of entry-exit $v \in \mathcal{V}^{\mathrm{b}}$ at a time $t \in \mathcal{T}$, where $\mathcal{V}^{\mathrm{b}}$ are the set of entry-exit nodes and $\mathcal{T}$ is our discrete time horizon. Note that in Hennings et al. (2020), these variables are written with hat notation, but we have omitted them

to avoid confusion with predicted values. We generate a balanced demand scenario, where the demands are bounded by the largest historically observed values, and the demand between time steps has a maximal change. Additionally, two entry or exits from the same *fence group*, $g \in \mathcal{G}$, see (Hennings et al. 2020), have maximal demand differences within the same time step. Let $\mathcal{I}$ denote the set of real-world instances, where the superscript $i \in \mathcal{I}$ indicates that the value is an observed value in real-world instance i, and the superscript 'sample' indicates the value is sampled. A complete flow forecast consists of $d_{v,t}^{\text{sample}}$ values for all $v \in \mathcal{V}^{\text{b}}$ and $t \in \mathcal{T}$ that satisfy the following constraints:

$$\sum_{v \in \mathcal{V}^{\text{b}}} d_{v,t}^{\text{sample}} = 0 \quad \forall t \in \mathcal{T} \tag{14}$$

$$\begin{aligned} M_{\text{q}} &= \max_{v \in \mathcal{V}^{\text{b}}, t \in \mathcal{T}, i \in \mathcal{I}} |d_{v,t}^{i}| \\ d_{v,t}^{\text{sample}} &\in \left[-\frac{21}{20} M_{\text{q}}, \frac{21}{20} M_{\text{q}} \right] \\ |d_{v,t}^{\text{sample}} - d_{v,t-1}^{\text{sample}}| &\leq 200 \quad \forall t \in \mathcal{T}, \quad v \in \mathcal{V}^{\text{b}} \end{aligned} \tag{15}$$

$$\text{sign}(d_{v,t}^{\text{sample}}) = \begin{cases} 1 & \text{if v is an entry} \\ -1 & \text{if v is an exit} \end{cases} \quad \forall t \in \mathcal{T}, \quad v \in \mathcal{V}^{\text{b}} \tag{16}$$

$$|d_{v_1,t}^{\text{sample}} - d_{v_2,t}^{\text{sample}}| \leq 200 \quad \forall t \in \mathcal{T}, \quad v_1, v_2 \in g, \ g \in \mathcal{G}, \ v_1, v_2 \in \mathcal{V}^{\text{b}}$$

To generate demand scenarios that satisfy constraints (14) and (15), we use the method proposed in Rubin (1981). Its original purpose was to generate samples from the Dirichlet distribution, but it 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 (14) and (15) for each time step. Hence we can apply the sampling method for all time steps and reject all samples that do not satisfy constraints (16). We note that 3 dimensions are sufficient for our gas network, and that the rejection method would scale poorly to higher dimensions.

In addition to flow demands, we require a pressure forecast for all entry-exits. Let $p_{v,t} \in \mathbb{R}$ be the pressure demand of entry-exit $v \in \mathcal{V}^{\text{b}}$ at time $t \in \mathcal{T}$. We generate pressures that respect bounds derived from the largest historically observed values, and have a maximal change for the same entry-exit between time steps. These constraints are described below:

$$\begin{aligned} M_{\text{p}}^{+} &= \max_{v \in \mathcal{V}^{\text{b}}, t \in \mathcal{T}, i \in \mathcal{I}} p_{v,t}^{i} \qquad M_{\text{p}}^{-} = \min_{v \in \mathcal{V}^{\text{b}}, t \in \mathcal{T}, i \in \mathcal{I}} p_{v,t}^{i} \\ p_{v,t}^{\text{sample}} &\in \left[M_{\text{p}}^{-} - \frac{1}{20}(M_{\text{p}}^{+} - M_{\text{p}}^{-}), M_{\text{p}}^{+} + \frac{1}{20}(M_{\text{p}}^{+} - M_{\text{p}}^{-}) \right] \end{aligned} \tag{17}$$

$$|p_{v,t}^{\text{sample}} - p_{v,t-1}^{\text{sample}}| \leq 5 \quad \forall t \in \mathcal{T}, \quad v \in \mathcal{V}^{\text{b}} \tag{18}$$

We now have the tools required to generate artificial forecast data, with the process described in Algorithm 1.

Algorithm 1: Boundary Value Forecast Generator

Input: Set of entry-exits $\mathcal{V}^{\mathrm{b}}$, discrete time horizon $\mathcal{T}$
Result: A forecast of pressure and flow values over the time horizon
flow_forecast ← Sample simplex (14), (15) uniformly until (16) holds for a sample
pressure_forecast ← Sample (17) uniformly until (18) holds for a sample
return (*flow_forecast*, *pressure_forecast*)

4.2.2 Operation mode sequence generation

During offline training, $\mathbb{D}_{\theta_2}$ 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

Input: Set of operation modes $\mathcal{O}$, discrete time horizon $\mathcal{T}$
Result: An operation mode per time step
for $t \leftarrow 1;\ t < |\mathcal{T}|;\ t \leftarrow t + 1$ **do**
  if $t == 1$ **then**
    Select random operation mode from $\mathcal{O}$ for time step t
  else if uniform random choice in range *[0,1]* $\geq$ *0.9* **then**
    Select random operation mode from $\mathcal{O}$ for time step t
  end
  else
    Select operation mode of previous time step for time step t
  end
end
return operation mode per time step

4.2.3 Initial state generation

In addition to the boundary forecast and operation mode sequence generators, we require a gas constants generator. As these values are assumed to be constant over the time horizon, we generate them only under the constraint that they are bounded by maximum historically observed values. Let gas_k represent the value for the gas constant $k \in$ {temperature, inflow norm density, molar mass, pseudo critical temperature, pseudo critical pressure}, then the following is the single constraint on sampling such values.

$$M^{+}_{gas_k} = \max_{i \in \mathcal{I}} gas^i_k \qquad M^{-}_{gas_k} = \min_{i \in \mathcal{I}} gas^i_k$$
$$gas^{\text{sample}}_k \in \left[M^{-}_{gas_k} - \frac{1}{20}(M^{+}_{gas_k} - M^{-}_{gas_k}),\ M^{+}_{gas_k} + \frac{1}{20}(M^{+}_{gas_k} - M^{-}_{gas_k}) \right] \tag{19}$$

We now have all the tools to generate synthetic initial states, which is the purpose Algorithm 3.

Algorithm 3: Initial State Generator

Input: *time_step* ($j \in [1, \cdots, |\mathcal{T}|]$), set of operation modes $\mathcal{O}$, set of entry-exits $\mathcal{V}^{\mathrm{b}}$, discrete time horizon $\mathcal{T}$
Result: An initial state to the transient gas optimisation problem
flow_forecast, *pressure_forecast* ← Boundary Value Forecast Generator($\mathcal{V}^{\mathrm{b}}$, $\mathcal{T}$) [a]
gas_constants ← Sample (19) uniformly for all gas constants
initial_state ← Select random state from real-world data
π ← (*flow_forecast*, *pressure_forecast*, *gas_constants*, *initial_state*) [b]
z_1 ← Operation Mode Sequence Generator($\mathcal{O}$, $\mathcal{T}$) [c]
$\mathbb{P}_\pi^{z_1}$ ← Generate from π and z_1
(*state_1*, $\cdots$, *state_k*) ← Optimal solution states from solving $\mathbb{P}_\pi^{z_1}$
return *state_j* `// time_step many steps from the initial state`

[a] See Algorithm 1
[b] Note that π explicitly includes *gas_constants* here. In all other cases *initial_state* contains this information.
[c] See Algorithm 2

5 Training scheme and warm start algorithm

In this section we introduce how our framework can be used to warm start MILPs and help achieve global optimality with lower solution times. Additionally, we outline the training scheme used for our NN design, as well as the real-world data used as a final validation set. For our application of transient gas instances, π is fully described by the flow and pressure forecast, combined with the initial state.

5.1 Primal heuristic and warm start

We consider the solution of $\mathbb{P}_\pi^{\hat{z_1}}$ as a primal heuristic for the original problem $\mathbb{P}_\pi$. We aim to incorporate $\mathbb{N}_{\{\theta_1,\theta_2\}}$ in a global MILP context and do this by using a partial solution of $\mathbb{P}_\pi^{\hat{z_1}}$ to warm start $\mathbb{P}_\pi$. The partial solution consists of $\hat{z_1}$, an additional set of binary variables called the *flow directions*, which are a subset of z_2 in Definition 1, and the realised pressure variables of the entry-exits, which are a subset of x_1. Note that partial solutions are used, since instances are numerically difficult. The primal heuristic and warm start algorithm are given in Algorithms 4 and 5 respectively.

Algorithm 4: Primal Heuristic

Input: Problem parameters π
Result: An optimal solution of $\mathbb{P}_\pi^{\hat{z_1}}$
$\hat{z_1} \leftarrow \mathbb{G}_{\theta_1}(\pi)$
$\mathbb{P}_\pi^{\hat{z_1}}$ ← Create MILP from π and $\hat{z_1}$
solution ← Solve $\mathbb{P}_\pi^{\hat{z_1}}$
return *solution*

Algorithm 5: Warm Start Algorithm

Input: Paramterised MILP $\mathbb{P}_\pi$
Result: An optimal solution of $\mathbb{P}_\pi$
primal_solution ← Primal Heuristic(π) [a]
optimal_solution ← Solve $\mathbb{P}_\pi$ with *primal_solution* as warm start
return *optimal_solution*

[a] See Algorithm 4

Algorithm 6: Neural Network Training

Input: Untrained $\mathbb{N}_{\{\theta_1,\theta_2\}}$, *prelabelled_data* (contains sampled: π, z_1, and $f(\mathbb{P}_\pi^{z_1})$), set of entry-exits $\mathcal{V}^{\mathrm{b}}$, discrete time horizon $\mathcal{T}$
Result: Trained $\mathbb{N}_{\{\theta_1,\theta_2\}}$
$\mathbb{D}_{\theta_2}$ ← Discriminator Pretraining($\mathbb{D}_{\theta_2}$, *prelabelled_data*)[a]
softmax_temp ← 0
for $i \leftarrow 0;\ i <$ *num_epochs*; $i \leftarrow i+1$ **do**
  for $j \leftarrow 0;\ j <$ *num_generator_epochs*; $j \leftarrow j+1$ **do**
    softmax_temp ← *softmax_temp* + 1
    loss ← Generator Training($\mathbb{N}_{\{\theta_1,\theta_2\}}$, *prelabelled_data*, $\mathcal{V}^{\mathrm{b}}$, $\mathcal{T}$)[b]
    if $loss \leq stopping\_loss\_generator$ **then**
      break
    end
    $\mathbb{G}_{\theta_1}$ ← Update $\mathbb{G}_{\theta_1}$ with Adam descent method[c]
  end
  data ← Prepare Discriminator Training Data($\mathbb{N}_{\{\theta_1,\theta_2\}}$, *prelabelled_data*)[d]
  train_data, *test_data* ← Split *data* with *ratio_test* proportionally in *test_data*
  for $j \leftarrow 0;\ j < num\_discriminator\_epochs;\ j \leftarrow j+1$ **do**
    $\mathbb{D}_{\theta_2}$ ← Discriminator Training Loop($\mathbb{D}_{\theta_2}$, *train_data*)[e]
    learning_rate ← Update *learning_rate* with *patience* and *factor*[f]
    test_loss ← Loss between $\hat{f}(\mathbb{P}_\pi^{z_1})$ and $f(\mathbb{P}_\pi^{z_1})$ from *test_data*[g]
    if *test_loss* ≤ *stopping_loss_discriminator* **then**
      break
    end
  end
end
return $\mathbb{N}_{\{\theta_1,\theta_2\}}$

[a] See Algorithm 8
[b] See Algorithm 10
[c] Introduced in Kingma and Ba (2014)
https://pytorch.org/docs/stable/generated/torch.optim.Adam.html#torch.optim.Adam
[d] See Algorithm 7
[e] See Algorithm 9
[f] See https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html
[g] See https://pytorch.org/docs/stable/generated/torch.nn.L1Loss.html

5.2 Training scheme

We generate our initial training and validation sets offline. This involves generating 10^4 initial states with parameter *time_step* set to 8 in Algorithm 3. Additionally, we

Table 1 MILP instance statistics over real-world data

	Mean
Number of constraints	6773.0
Number of continuous variables	2388.0
Number of binary variables	1285.0
Number of nonzeros	25436.3
Number of z_1 variables	672.0
Number of constraints (after presolve)	5183.4
Number of continuous variable (after presolve)	1222.9
Number of binary variables (after presolve)	1178.1
Number of nonzeros (after presolve)	37987.7

generate 4×10^6 coupled demand scenarios and operation mode sequences with Algorithms 1 and 2. All instances contain 12 time steps (excluding the initial state) with 30 minutes between each step. This training data is exclusively used by $\mathbb{D}_{\theta_2}$, and is 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 as described in Algorithm 6, alternating between $\mathbb{G}_{\theta_1}$ and $\mathbb{D}_{\theta_2}$. The complete list of parameters used are in Table 3, with default values being used otherwise. The exact block design of $\mathbb{N}_{\{\theta_1,\theta_2\}}$ can be seen in Fig. 3, and the general layout in Fig. 1. For the complete NN design we refer readers to Fig. 14 and Table 6 in the Appendix.

5.3 Real world data

Real-world instances, similar to the artificial data, contain 12 time steps with 30 minutes between each step. We focus on Station D from Hennings et al. (2020), and present only results for this station. The topology for Station D can be seen in Fig. 13 in Appendix 1. Station D can be thought of as a T intersection, and is of average complexity compared to the stations presented in Hennings et al. (2020). 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. 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, where instances are on average 15 minutes apart and total 9291. Statistics on the these real-world MILP instances are provided in Table 1.

6 Computational results

We partition our results into three subsections. Section 6.1 focuses on the data generation methods, Sect. 6.2 on $\mathbb{N}_{\{\theta_1,\theta_2\}}$ during training and its performance on synthetic

data, and Sect. 6.3 on the performance of trained $\mathbb{N}_{\{\theta_1,\theta_2\}}$ on 15 weeks of real-world transient gas data.

For our experiments we use PyTorch 1.4.0 (Paszke et al. 2019) as our ML modelling framework, Pyomo v5.5.1 Hart et al. 2017, 2011 as our MILP modelling framework, and Gurobi v9.02 (Gurobi Optimization 2020) as our MILP solver. The MILP solver settings are available in Table 5 in Appendix 1. $\mathbb{N}_{\{\theta_1,\theta_2\}}$ is 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 evaluations are 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.

6.1 Data generation results

Figure 4 (left) shows how our generated flow prognosis compares to that of historic real-world data. We see that Nodes A, B, and C function both as entries and exits, but are dominated by a single orientation for each node over historical data. Specifically, Node C is the general entry, and Nodes A and B are the exits. In addition to the general orientation, we see that each node has significantly different real-world flow distributions, as opposed to the near identical distributions over our artificial data. Figure 4 (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 Fig. 11, as in most cases the gas network station is in bypass.

These differences between our synthetic data and real-world data are somewhat expected. The underlying distribution of the demand scenarios for both flow and pressure cannot be assumed to be uniform nor conditionally independent unlike in Algorithm 1. Moreover, the sampling range we use is significantly larger than that of the real-world observed values as we take a single maximum and minimum value over all entry-exits. We expect these differences to continue with our other data generation methods. Algorithm 3 was designed to output varied and valid initial states w.r.t. our MILP formulation; however, the choice of operation modes that occur in reality is unlikely to be uniform as generated in Algorithm 2. In reality, some operation modes occur with a much higher frequency than others. Additionally, we rely on a MILP solver to generate new initial states, and therefore cannot rule out the possibility of a bias. We believe these probable differences in distributions call for further research in realistic prognosis generation methods.

6.2 Training results

Figure 5 visualises the losses of $\mathbb{D}_{\theta_2}$ throughout the initial offline training. We observe that the loss decreases throughout training, highlighting the improvement in $\mathbb{D}_{\theta_2}$ for predicting $f(\mathbb{P}_\pi^{z_1})$. 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. The validation loss on synthetic data also converges to approximately 1000,

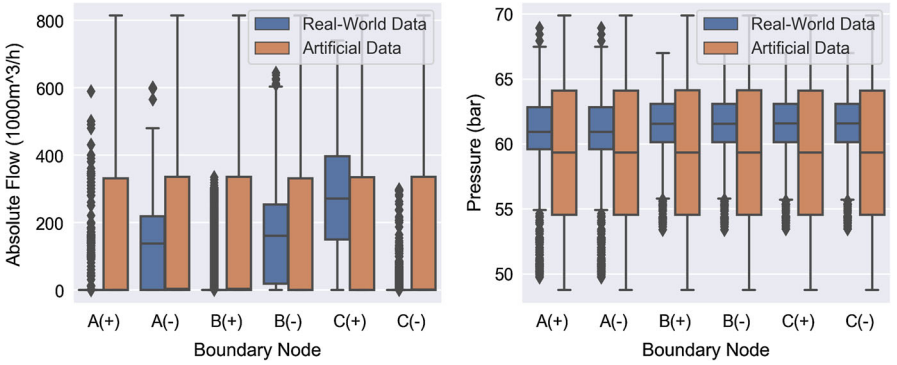


Fig. 4 Comparison of generated value distributions per node vs. the distribution seen in real-world data. For flow (Left), and pressure (Right)

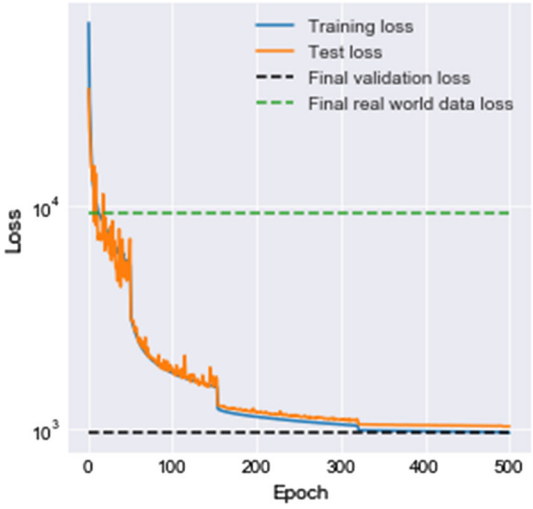


Fig. 5 The loss per epoch of $\mathbb{D}_{\theta_2}$ during the initial training of Algorithm 8. The dashed lines show the performance of $\mathbb{D}_{\theta_2}$ after $\mathbb{N}_{\{\theta_1,\theta_2\}}$ has been completely trained. A log scale is used for better visibility of later epochs

indicating that $\mathbb{D}_{\theta_2}$ generalises to unseen $\mathbb{P}_\pi^{z1}$ instances; however, we note that this generalisation doesn't translate perfectly to real-world data. Despite this we believe that an average distance between $\hat{f}(\mathbb{P}_\pi^{z1})$ and $f(\mathbb{P}_\pi^{z1})$, of 10000 is still very good. We discuss the issues of different underlying distributions of real-world data and our generated data distributions in Sect. 6.1.

The training loss during Algorithm 6 for $\mathbb{D}_{\theta_2}$ is shown in Fig. 6, and for $\mathbb{G}_{\theta_1}$ in Fig. 7. The observable cyclical increases in the training and test loss of $\mathbb{D}_{\theta_2}$ occur during the periodic retraining of $\mathbb{G}_{\theta_1}$. We believe that $\mathbb{G}_{\theta_1}$ learns how to induce suboptimal predictions during this periodic retraining. $\mathbb{D}_{\theta_2}$ in turn quickly relearns, but this highlights that learning how to predict $f(\mathbb{P}_\pi^{\hat{z1}})$ is unlikely without some error. Figure 7 (left) shows the loss over time of $\mathbb{G}_{\theta_1}$ as it is trained, with Fig. 7 (right) displaying magnified losses for the final epochs. We observe that $\mathbb{G}_{\theta_1}$ quickly learns important z_1 decision values. We hypothesise that this quick descent is helped by $\hat{z_1}$ that are unlikely

Fig. 6 The loss per epoch of $\mathbb{D}_{\theta_2}$ as it is trained using Algorithm 6

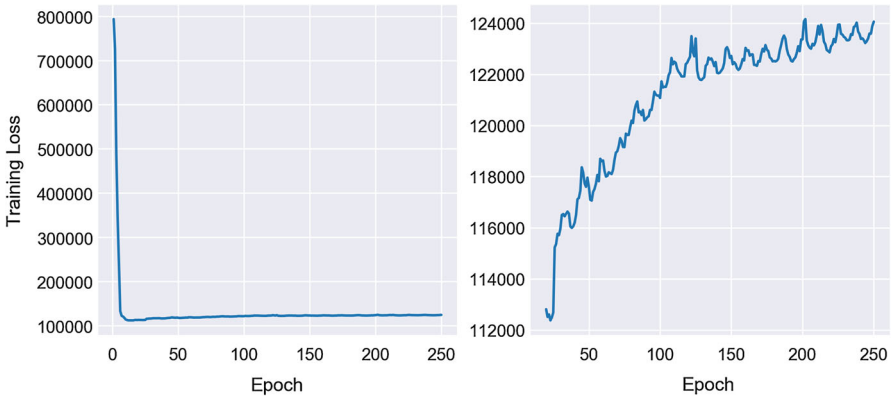


Fig. 7 (Left) The training loss per epoch of $\mathbb{G}_{\theta_1}$ as it is trained using Algorithm 6. On the left the loss over all epochs is shown. (Right) A magnified view of the loss starting from epoch 20

given our generation method in Algorithm 2. The loss increases following this initial decrease in the case of $\mathbb{G}_{\theta_1}$, showing the ability of $\mathbb{D}_{\theta_2}$ to further improve. It should also be noted that significant step-like decreases in loss are absent in both (left) and (right) of Fig. 7. We believe such steps would indicate $\mathbb{G}_{\theta_1}$ discovering new important z_1 values (operation modes). The diversity of produced operation modes, however, see Fig. 11, 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.

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 651 instances have been removed as warm starting resulted in an inconsistency with the set optimality tolerances. These instances have been kept in the

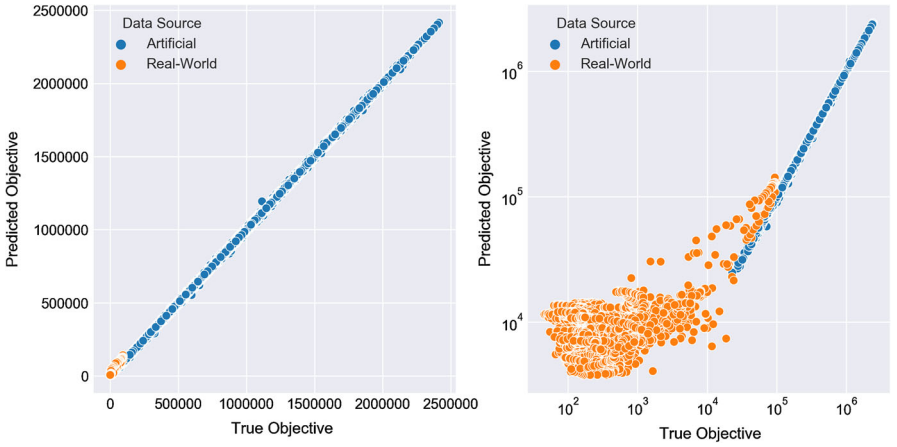


Fig. 8 $\hat{f}(\mathbb{P}_\pi^{\hat{z}1})$ for the validation set, and $\hat{f}(\mathbb{P}_\pi^{z_1^*})$ for real-world data, compared to $f(\mathbb{P}_\pi^{\hat{z}1})$ and $f(\mathbb{P}_\pi)$ respectively. Linear scale (Left) and log-scale (Right)

graphics, but are marked and conclusions will not be drawn from them. We also note that the linear programming relaxation of the MILP formulation from Hennings et al. (2020) is rather weak, largely due to the big-M constraints that model the controllable network elements. We believe that the weak relaxation is partly responsible for long run times, especially for scenarios that require a lot of slack and need to branch extensively to prove global optimality. This hypothesis is supported by Fig. 9, where the MILP instances that hit the time limit are predominantly those with large objective values.

Figure 8 compares predicted and true objectives for both artificial and real-world data. As expected, the distribution of objective values is visibly different for the artificial validation set compared to the real-world validation set. Our data generation method was intended to be as independent as possible from the historic data, and as a result, the average scenario has optimal solution larger than any real-world data point. The performance of $\mathbb{D}_{\theta_2}$ is again clearly visible here, however, 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, which we hypothesise is mainly due to the the lower values of $f(\mathbb{P}_\pi)$.

Figure 9 shows the comparison of $f(\mathbb{P}_\pi^{\hat{z}1})$ and $f(\mathbb{P}_\pi)$. In a similar manner to $\mathbb{D}_{\theta_2}$, we see that $\mathbb{G}_{\theta_1}$ 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 identical parameter values π. This comes as little surprise given the struggle of $\mathbb{D}_{\theta_2}$ 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. However, the value of our primal heuristic is clearly visible from those instances where the heuristic retrieves a better solution than the MILP solver does within an hour. Additionally, we see that no unsolved instance above the line $f(\mathbb{P}_\pi^{\hat{z}1}) = f(\mathbb{P}_\pi)$ is very far from the line, showing that our primal heuristic produces a comparable, sometimes equivalent solution, in

Fig. 9 A comparison of $f(\mathbb{P}_\pi^{\hat{z_1}})$ and $f(\mathbb{P}_\pi)$ for all real-world data instances

Table 2 Solution time statistics for different solving strategies

	Mean	Median	STD	Min	Max
$\mathbb{N}_{\{\theta_1,\theta_2\}}$ Inference Time (s)	0.009	0.008	0.001	0.008	0.017
Warm start $\mathbb{P}_\pi$ Time (s)	100.830	9.380	421.084	0.130	3600.770
$\mathbb{P}_\pi$ Time (s)	147.893	24.380	463.279	3.600	3601.280
$\mathbb{P}_\pi^{\hat{z_1}}$ + Warm start $\mathbb{P}_\pi$ 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

a much shorter time than the MILP solver's one hour. For a comparison of solution times, see Table 2.

Figure 10 shows the performance of the predictions $\hat{f}(\mathbb{P}_\pi^{\hat{z_1}})$ compared to $f(\mathbb{P}_\pi^{\hat{z_1}})$. Interestingly, $\mathbb{D}_{\theta_2}$ 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 $\mathbb{D}_{\theta_2}$ struggles with $f(\mathbb{P}_\pi^{\hat{z_1}})$ instances near 0, but the trend is evident for larger valued instances too. We observe that no data point is too far from the line $\hat{f}(\mathbb{P}_\pi^{\hat{z_1}}) = f(\mathbb{P}_\pi^{\hat{z_1}})$, and conclude, albeit with some generalisation loss, that $\mathbb{D}_{\theta_2}$ can adequately predict $\hat{z_1}$ solutions from $\mathbb{G}_{\theta_1}$ despite the change in data sets.

We now compare the operation modes $\hat{z_1}$ that are generated by $\mathbb{G}_{\theta_1}$, and the z_1^* that are produced by our MILP 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 Fig. 11, which plots the frequency of specific z_1

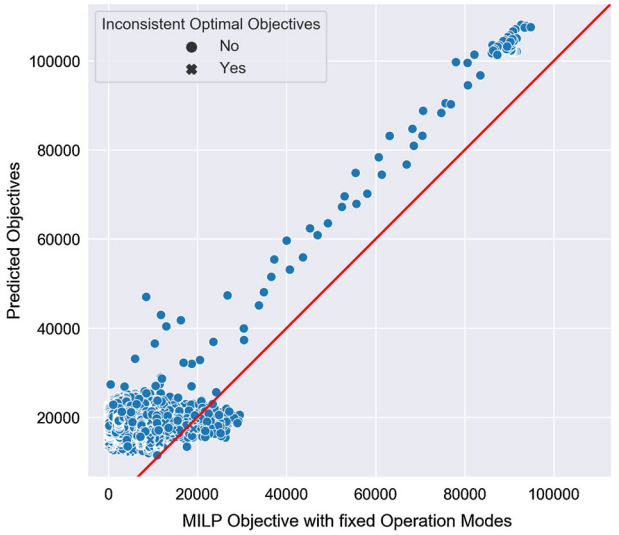


Fig. 10 A comparison of $\hat{f}(\mathbb{P}_\pi^{\hat{z_1}})$ and $f(\mathbb{P}_\pi^{\hat{z_1}})$ for all real-world data instances

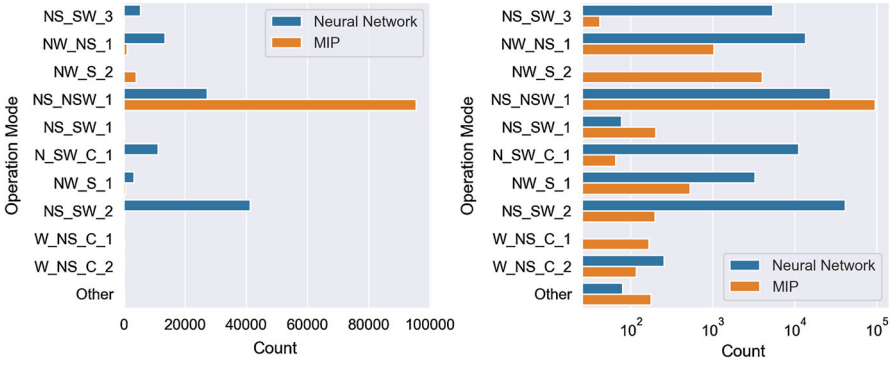


Fig. 11 Frequency of operation mode choice by $\mathbb{G}_{\theta_1}$ compared to MILP solver for all real-world instances. (Left) Linear scale, and (Right) log scale

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 MILP solver selecting symmetric solutions with equal probability. For instance, take operation modes W_NS_C_1 and W_NS_C_2, which differ by their usage of one of two identical compressor machines. $\mathbb{N}_{\{\theta_1,\theta_2\}}$ only ever predicts W_NS_C_2; however, with half the frequency the MILP solver selects each of them. We now suspect that these duplicate choices do not exist in bypass modes, and the uniqueness of z_1 determined by open flow paths, results in different $f(\mathbb{P}_\pi^{z_1})$ values. We believe that the central importance of NS_NSW_1 was not learnt by $\mathbb{N}_{\{\theta_1,\theta_2\}}$ as 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 4 in Appendix 1.

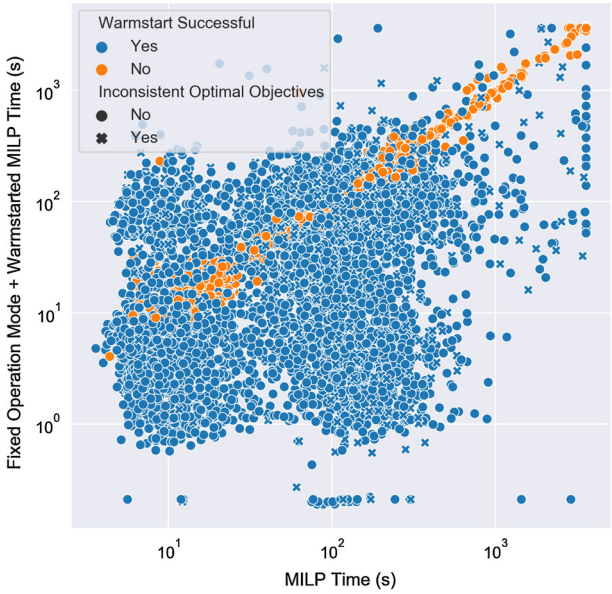


Fig. 12 The combined running time of solving $\mathbb{P}_\pi^{\hat{z_1}}$, and solving a warm started $\mathbb{P}_\pi$, compared to solving $\mathbb{P}_\pi$ directly

As discussed above, $\mathbb{N}_{\{\theta_1,\theta_2\}}$ 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 5. The results of our warm start algorithm are displayed in Fig. 12. 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(s) 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 solvable given the warm start suggestion. As such, we have created an effective primal heuristic that is both quick to run and beneficial in achieving global optimality.

7 Conclusion

In this paper, we have presented a dual NN design for generating decisions in a MILP. This design is trained without ever solving the MILP with unfixed decision variables. The NN is both used as a primal heuristic and used to warm-start the MILP solver for the original problem. We have shown the usefulness of our design on the transient gas transport problem. While doing so we have 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 NN 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 as a warm-start solution.

While our approach is an important step forward in NN design and ML's application to gas transport, we believe that there exist three primary directions for future research. The first is to convert our approach into more traditional reinforcement learning, and then utilise policy gradient approaches, see (Thomas and Brunskill 2017). The major hurdle to this approach is that much of the computation would be shifted online, requiring many more calls to solve the induced MILPs. However, this could be offset by using our technique to initialise the NN for such an approach, thereby avoiding early stage training difficulties. The second is focused on the recent improvements in Graph NNs, see (Gasse et al. 2019). Their ability to generalise to different input sizes would permit the creation of a single NN over multiple gas network topologies. The final direction is to improve data generation techniques for transient gas networks. There exists a gap in the literature for improved methods that are scalable and result in real-world like data.

Funding Open Access funding enabled and organized by Projekt DEAL.

Appendix

Algorithm 7: Mix New Data

Input: $\mathbb{N}_{\{\theta_1,\theta_2\}}$, *prelabelled_data* (contains sampled: π, z_1, and $f(\mathbb{P}_\pi^{z_1})$), set of entry-exits $\mathcal{V}^{\mathrm{b}}$, discrete time horizon $\mathcal{T}$
Result: New data generated by $\mathbb{G}_{\theta_1}$ for training $\mathbb{D}_{\theta_2}$
new_data ← array of length *num_data_new*
for $i \leftarrow 0$; $i <$ num_data_new; $i \leftarrow i+1$ **do**
  initial_state ← Uniformly select from *prelabelled_data*
  flow_forecast, *pressure_forecast* ← Boundary Prognosis Generator($\mathcal{V}^{\mathrm{b}}$, $\mathcal{T}$)[a]
  $\pi \leftarrow$ (*flow_forecast*, *pressure_forecast*, *initial_state*)
  $\hat{z_1} \leftarrow \mathbb{G}_{\theta_1}(\pi)$
  $f(\mathbb{P}_\pi^{\hat{z_1}}) \leftarrow$ solve $\mathbb{P}_\pi^{\hat{z_1}}$
  new_data[i] ← Create new data point from $(\pi, z_1, f(\mathbb{P}_\pi^{\hat{z_1}}))$
end
old_data ← Uniformly sample *num_data_old* points uniquely from *prelabelled_data*
return Random ordering of *new_data* ∪ *old_data*

[a] See Algorithm 1

Algorithm 8: Discriminator Pretraining

Input: Untrained $\mathbb{D}_{\theta_2}$, *pre_labelled_data* (contains sampled: π, z_1, and $f(\mathbb{P}_{\pi}^{z_1})$)
Result: An initially trained $\mathbb{D}_{\theta_2}$
for $i \leftarrow 0$; $i < num\_epochs$; $i \leftarrow i + 1$ **do**
  $\mathbb{D}_{\theta_2} \leftarrow$ Discriminator Training Loop($\mathbb{D}_{\theta_2}$, *prelabelled_data*)[a] *learning_rate* $\leftarrow$ Update *learning_rate* with *patience* and *factor*[b]
end
return $\mathbb{D}_{\theta_2}$

[a] See Algorithm 9

[b] See https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html

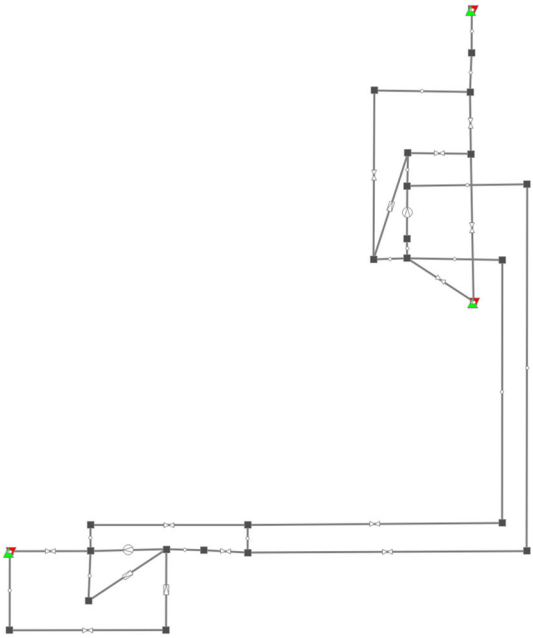
Algorithm 9: Discriminator Training Loop

Input: $\mathbb{D}_{\theta_2}$, *prelabelled_data* (contains: π, z_1/ $\hat{z_1}$, and $f(\mathbb{P}_{\pi}^{z_1})$/ $f(\mathbb{P}_{\pi}^{\hat{z_1}})$)
Result: An updated $\mathbb{D}_{\theta_2}$
batches $\leftarrow$ Split *prelabelled_data* into *num_batches* many batches
for *batch* in *batches* **do**
  $\hat{f}(\mathbb{P}_{\pi}^{z_1}) \leftarrow$ Forward pass of $\mathbb{D}_{\theta_2}$ over the *batch*
  loss $\leftarrow$ Loss between $\hat{f}(\mathbb{P}_{\pi}^{z_1})$ and true objectives from *prelablled_data*[a]
  $\mathbb{D}_{\theta_2} \leftarrow$ Update $\mathbb{D}_{\theta_2}$ with Adam descent method[b] `// Using weight_decay`
end
return $\mathbb{D}_{\theta_2}$

[a] See https://pytorch.org/docs/stable/generated/torch.nn.L1Loss.html

[b] See https://pytorch.org/docs/stable/generated/torch.optim.Adam.html#torch.optim.Adam

Fig. 13 Topology of Station D



Algorithm 10: Generator Training

Input: $\mathbb{N}_{\{\theta_1,\theta_2\}}$, *prelabelled_data* (contains sampled: π, z_1, and $f(\mathbb{P}_\pi^{z_1})$), set of entry-exits $\mathcal{V}^{\mathrm{b}}$, discrete time horizon $\mathcal{T}$
Result: Average loss in training
data ← array of length *num_scenarios*
for $i \leftarrow 0;\ i < num\_scenarios;\ i \leftarrow i+1$ **do**
  initial_state ← Uniformly select from *prelabelled_data*
  flow_forecast, *pressure_forecast* ← Boundary Prognosis Generator($\mathcal{V}^{\mathrm{b}}$, $\mathcal{T}$)[a]
  π ← (*flow_forecast*, *pressure_forecast*, *initial_state*)
  data[i] ← Create an unlabelled data point π
end
batches ← Create batches of size *batch_size*
losses ← array of length corresponding to length of *batches*
for $i \leftarrow 0;\ i < num\_batches;\ i \leftarrow i+1$ **do**
  $\hat{f}(\mathbb{P}_\pi^{\hat{z_1}})$ ← Forward pass of $\mathbb{N}_{\{\theta_1,\theta_2\}}$ over the *batch*
  losses[i] ← Loss between $\hat{f}(\mathbb{P}_\pi^{\hat{z_1}})$ and 0[b]
  $\mathbb{N}_{\{\theta_1,\theta_2\}}$ ← Update $\mathbb{N}_{\{\theta_1,\theta_2\}}$ with Adam descent method[c] `// θ2 is frozen`
  learning_rate ← Update using *max_lr*, *base_lr*, and *step_size_up*[d]
end
return mean value of *losses*

[a] See Algorithm 1
[b] See https://pytorch.org/docs/stable/generated/torch.nn.L1Loss.html
[c] See https://pytorch.org/docs/stable/generated/torch.optim.Adam.html#torch.optim.Adam
[d] Introduced in Smith (2017), see
https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.CyclicLR.html

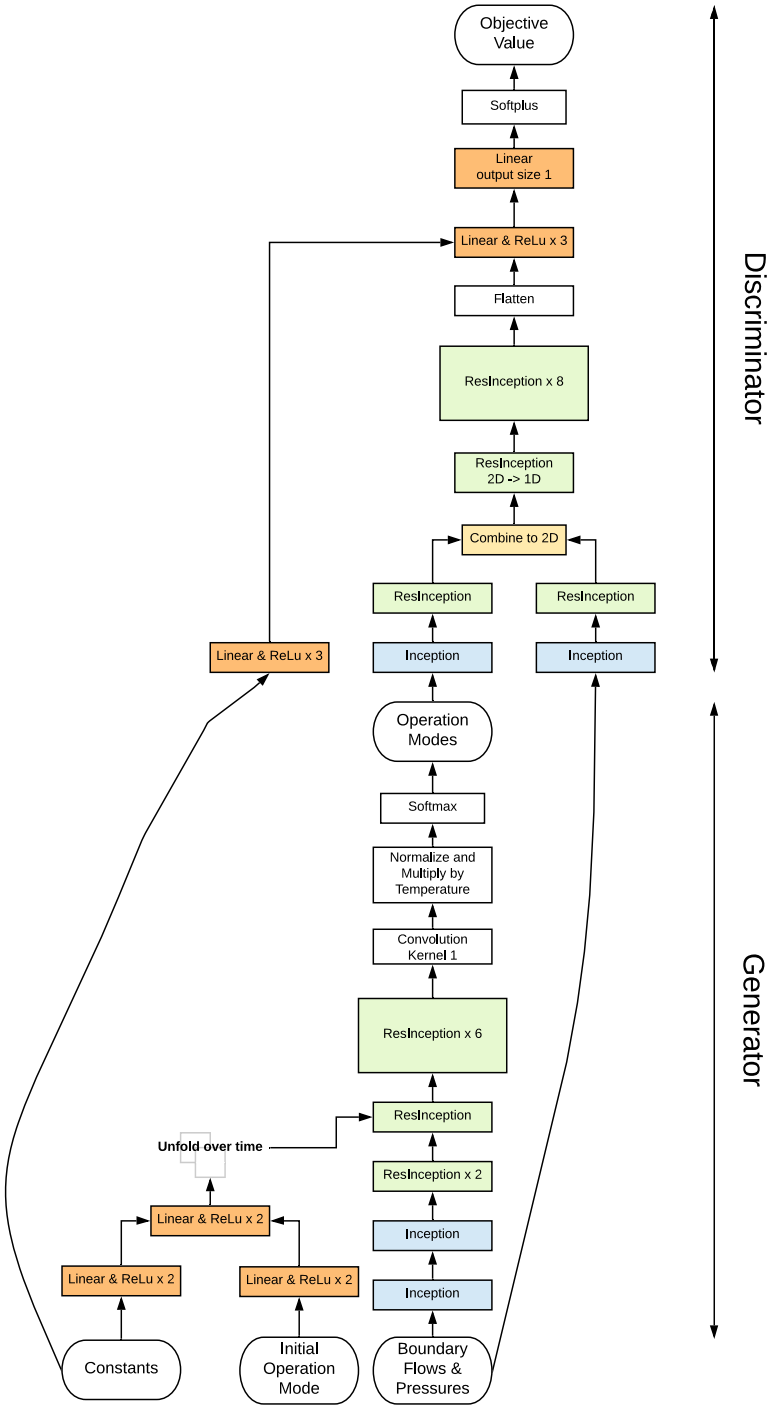


Fig. 14 Neural network architecture

Table 3 Parameters for training

Parameter	Method	Value
batch_size	Algorithm 8	2048
num_epochs	Algorithm 8	500
learning_rate	Algorithm 8 / 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 7	2048
num_data_old	Algorithm 7	8192
num_epochs	Algorithm 6	10
num_generator_epochs	Algorithm 6	25
num_discriminator_epochs	Algorithm 6	25
stopping_loss_discriminator	Algorithm 6	3 * 1022.5[1]
stopping_loss_generator	Algorithm 6	0.9 * 121848.27 [2]
ratio_test	Algorithm 6	0.1
learning_rate	Algorithm 6 / Adam	0.001
patience	Algorithm 6 / ReduceLROnPlateau	2
factor	Algorithm 6 / ReduceLROnPlateau	0.5

[1] 1022.5 was the test loss after initial discriminator training.

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

Table 4 Operation Mode Correlation Matrix between $\hat{z_1}$ and z_1^*

	NW_NS_1	NS_SW_2	N_SW_C_1	NS_NSW_1	W_NS_C_1	NS_SW_1	NW_S_2	NS_SW_3	W_NS_C_2	NW_S_1	Other
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 5 Parameters for MILP solving

Parameter	Value	Description
TimeLimit	3600 (s)	The maximum time the instance is allowed to run
FeasibilityTol	1e−6	All constraints must be satisfied within this tolerance
MIPGap	1e−4	The relative gap tolerance for declaring optimality
MIPGapAbs	1e−2	The absolute gap tolerance for declaring optimality
NumericFocus	3	Employ more expensive numerically safer techniques

Table 6 Number of parameters in the neural network and submodules

	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	–	–

References

Achterberg T (2007) Constraint integer programming. Ph.D. thesis, Technische Universität Berlin

Baltean-Lugojan R, Bonami P, Misener R, Tramontani A (2019) Scoring positive semidefinite cutting planes for quadratic optimization via trained neural networks. Optimization-online preprint 2018/11/6943

Beliën J, Demeulemeester E, Cardoen B (2009) A decision support system for cyclic master surgery scheduling with multiple objectives. J Sched 12(2):147

Bengio Y, Lodi A, Prouvost A (2018) Machine learning for combinatorial optimization: a methodological tour d'horizon. arXiv preprint arXiv:1811.06128

Bertsimas D, Stellato B (2018) The voice of optimization. arXiv preprint arXiv:1812.09991

Bertsimas D, Stellato B (2019) Online mixed-integer optimization in milliseconds. arXiv preprint arXiv:1907.02206

Burlacu R, Egger H, Groß M, Martin A, Pfetsch ME, Schewe L, Sirvent M, Skutella M (2019) Maximizing the storage capacity of gas networks: a global minlp approach. Optim Eng 20(2):543–573

Chen Z, Zhong Y, Ge X, Ma Y (2020) An actor-critic-based uav-bss deployment method for dynamic environments. arXiv preprint arXiv:2002.00831

Dempe S (2002) Foundations of bilevel programming. Springer, Berlin

Ding JY, Zhang C, Shen L, Li S, Wang B, Xu Y, Song L (2019) Optimal solution predictions for mixed integer programs. arXiv preprint arXiv:1906.09575

Etheve M, Alès Z, Bissuel C, Juan O, Kedad-Sidhoum S (2020) Reinforcement learning for variable selection in a branch and bound algorithm. arXiv preprint arXiv:2005.10026

Ferber A, Wilder B, Dilina B, Tambe M (2019) Mipaal: mixed integer program as a layer. arXiv preprint arXiv:1907.05912

Fischetti M, Lodi A (2003) Local branching. Math Program 98(1–3):23–47

Gasse M, Chételat D, Ferroni N, Charlin L, Lodi A (2019) Exact combinatorial optimization with graph convolutional neural networks. In: Advances in neural information processing systems, pp 15554–15566

Goodfellow I (2016) Nips 2016 tutorial: Generative adversarial networks. arXiv preprint arXiv:1701.00160

Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT press, London

Gurobi Optimization L (2020) Gurobi optimizer reference manual. http://www.gurobi.com

Hanachi H, Mechefske C, Liu J, Banerjee A, Chen Y (2018) Performance-based gas turbine health monitoring, diagnostics, and prognostics: a survey. IEEE Trans Reliab 67(3):1340–1363

Hart WE, Watson JP, Woodruff DL (2011) Pyomo: modeling and solving mathematical programs in python. Math Program Comput 3(3):219–260
Hart WE, Laird CD, Watson JP, Woodruff DL, Hackebeil GA, Nicholson BL, Siirola JD (2017) Pyomo-optimization modeling in python, vol 67, 2nd edn. Springer, Berlin
Hennings F, Anderson L, Hoppmann-Baum K, Turner M, Koch T (2020) Controlling transient gas flow in real-world pipeline intersection areas. Optim Eng 47:1–48
Hoppmann K, Hennings F, Lenz R, Gotzes U, Heinecke N, Spreckelsen K, Koch T (2019) Optimal operation of transient gas transport networks. Technical report, Technical Report 19–23, ZIB, Takustr. 7, 14195 Berlin
Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980
Kunz F, Kendziorski M, Schill WP, Weibezahn J, Zepter J, von Hirschhausen CR, Hauser P, Zech M, Möst D, Heidari S et al (2017) Electricity, heat, and gas sector data for modeling the german system. Technical report, DIW Data Documentation
Masti D, Bemporad A (2019) Learning binary warm starts for multiparametric mixed-integer quadratic programming. In: 2019 18th European control conference (ECC), IEEE, pp 1494–1499
MohamadiBaghmolaei M, Mahmoudy M, Jafari D, MohamadiBaghmolaei R, Tabkhi F (2014) Assessing and optimization of pipeline system performance using intelligent systems. J Nat Gas Sci Eng 18:64–76
Moritz S (2007) A mixed integer approach for the transient case of gas network optimization. Ph.D. thesis, Technische Universität Darmstadt
Nair V, Bartunov S, Gimeno F, von Glehn I, Lichocki P, Lobov I, O'Donoghue B, Sonnerat N, Tjandraatmadja C, Wang P, Addanki R, Hapuarachchi T, Keck T, Keeling J, Kohli P, Ktena I, Li Y, Vinyals O, Zwols Y (2020) Solving mixed integer programs using neural networks. arXiv:2012.13349
Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, et al. (2019) Pytorch: An imperative style, high-performance deep learning library. In: Advances in neural information processing systems, pp 8026–8037
Petkovic M, Chen Y, Gamrath I, Gotzes U, Hadjidimitriou NS, Zittel J, Koch T (2019) A hybrid approach for high precision prediction of gas flows. Technical Report , pp. 19–26, ZIB, Takustr. 7, 14195 Berlin
Pfau D, Vinyals O (2016) Connecting generative adversarial networks and actor-critic methods. arXiv preprint arXiv:1610.01945
Pourfard A, Moetamedzadeh H, Madoliat R, Khanmirza E (2019) Design of a neural network based predictive controller for natural gas pipelines in transient state. J Nat Gas Sci Eng 62:275–293
Ríos-Mercado RZ, Borraz-Sánchez C (2015) Optimization problems in natural gas transportation systems: a state-of-the-art review. Appl Energy 147:536–555
Rubin DB (1981) The bayesian bootstrap. Ann Stat 5:130–134
Ruiz R, Maroto C, Alcaraz J (2004) A decision support system for a real vehicle routing problem. Eur J Oper Res 153(3):593–606
Smith LN (2017) Cyclical learning rates for training neural networks. arXiv:1506.01186
Szegedy C, Ioffe S, Vanhoucke V, Alemi AA (2017) Inception-v4, inception-resnet and the impact of residual connections on learning. In: Thirty-first AAAI conference on artificial intelligence
Tang Y, Agrawal S, Faenza Y (2019) Reinforcement learning for integer programming: Learning to cut. arXiv preprint arXiv:1906.04859
Thomas PS, Brunskill E (2017) Policy gradient methods for reinforcement learning with function approximation and action-dependent baselines. arXiv preprint arXiv:1706.06643
Wong E, Kolter JZ (2017) Provable defenses against adversarial examples via the convex outer adversarial polytope. arXiv preprint arXiv:1711.00851
Yueksel Erguen I, Zittel J, Wang Y, Hennings F, Koch T (2020) Lessons learned from gas network data preprocessing. Tech. rep., Technical Report, pp 20–13, ZIB, Takustr. 7, 14195 Berlin
Zarpellon G, Jo J, Lodi A, Bengio Y (2020) Parameterizing branch-and-bound search trees to learn branching policies. arXiv preprint arXiv:2002.05120