Comparing Machine Learning and Optimization Approaches for the $N-k$ Interdiction Problem Considering Load Variability

Alejandro D. Owen Aquino
Georgia Institute of Technology
aaquino30@gatech.edu

Rachel J. Harris
Georgia Institute of Technology
rharris94@gatech.edu

Alyssa Kody
Argonne National Laboratory
akody@anl.gov

Daniel K. Molzahn
Georgia Institute of Technology
molzahn@gatech.edu

Abstract

Power grids must be operated, designed, and maintained such that a small number of line failures will not result in significant load shedding. To identify problematic combinations of failures, we consider an $N-k$ interdiction problem that seeks the set of k failed lines (out of N total lines) that result in the largest load shed. This is formulated as a bilevel optimization problem with an upper level representing the attacker that selects line failures and a lower level modeling the defender's generator redispatch. Compounding the difficulties inherent to the bilevel nature of interdiction problems, we consider a nonlinear AC power flow model that makes this problem intractable with traditional approaches. Furthermore, since the solutions found at a particular load condition may not generalize to other loading conditions, operators may need to quickly recompute these worst-case failures online to protect against them during operations. To address these challenges, we formulate and compare the performance of three simplified methods for solving the $N-k$ interdiction problem: a state-of-the-art optimization approach based on a network-flow relaxation of the power flow equations and two newly developed machine learning (ML) algorithms that predict load sheds given the state of the network.

Keywords: Bilevel Optimization, Interdiction, Neural Networks, $N-k$.

1. Introduction

Today, power grids are designed and operated such that networks are $N-1$ secure. However, the threat of multiple lost lines from natural disasters, cyber-physical attacks, or aging components is increasing and therefore operators need to prepare for $N-k$ contingency scenarios [1]–[3]. Such $N-k$ contingency scenarios are useful for long-term design and planning. System operators may also need to identify these worst-case contingencies quickly during real-time operations. For example, they must identify components to prioritize for short-term maintenance and disallow shutdown of critical devices during repair, or identify the most important devices on which to deploy additional security measures during war time scenarios. Overall, there has been increasing interest in real-time situational awareness for grid resilience, exemplified by the US Department of Energy's North American Energy Resilience Model (NAERM) program [4]. This paper explores methods to identify critical sets of line failures which are computationally tractable and can be solved online to enable fast corrective and protective actions.

The optimization problem developed to identify critical sets of line failures is typically formulated as an adversarial bilevel problem called the $N-k$ interdiction problem. Modeling the attacker, the upper level disconnects up to k lines out of the N lines in the system in order to maximize load shed. The lower level, representing the defender, optimally redispatches the system to maximize load delivery given the disconnected lines.

When using the AC power flow equations to model the network physics, this problem is a nonconvex bilevel mixed-integer nonlinear program (MINLP) that is strongly NP-hard. Nevertheless, some previous works maintain the nonlinear AC power flow equations in the lower-level problem and solve with tailored algorithms [1], [5]–[7] to make the problem more computationally tractable. While these methods can provide high-fidelity solutions, they are not suited for being solved repeatedly in real-time. Most formulations therefore rely on linearizing the lower-level problem, which can lead to "false positives" or "false negatives" when identifying

dangerous contingencies [3], [8]–[11]. In essence, solving this problem involves a tradeoff between solution correctness and computational tractability.

Machine learning (ML) approaches are promising in that they can produce higher-quality solutions compared to traditional linearized approximations and, after offline training, can be used to make quick, efficient predictions during real-time operations. ML has been used for contingency analysis in works such as [2], which used Q-learning to identify the worst-impacted zones of power grids during extreme events. Other works such as [12]–[14] use neural networks (NNs) to rank the severity of contingencies based on predicting performance indices or line flow approximations.

In this paper, we propose and benchmark two new ML approaches for quickly and accurately identifying worst-case failures in power systems. In the first approach, we train a NN to predict load shed for any given set of line failures and power demands. We then embed the NN's mapping of contingency to load shed inside a mixed-integer linear program (MILP) and optimize for highest load shed. In the second approach, we train a NN to predict the impact of individual line failures on load shed and step through failures one at a time to estimate a contingency's total load shed. Unlike previous work, we directly predict load shed at a given network state during the process of ranking contingencies. We also consider a range of load variation, i.e., the factor by which loads may randomly vary from nominal values during training and run-time testing, in both of these approaches to make the predictions flexible to different load patterns and/or inclusion of distributed solar. We compare the performance of these new ML approaches to the state-of-the-art linearized bilevel optimization approach in [11]. The results demonstrate that the ML methods outperform this linearized approach at finding the most dangerous contingencies on average over several test cases and contingency sizes.

The rest of the paper is organized as follows. First, we formulate the $N-k$ interdiction problem in Section 2. Second, we present the two new ML approaches proposed in this paper as well as the state-of-the-art linearized bilevel problem which provides a baseline for comparison in Section 3. Third, we report the results of those approaches when applied to several test cases in Section 4. We conclude the paper and discuss directions for future research in Section 5.

2. The $N-k$ Interdiction Problem

This section describes the bilevel optimization formulation of the $N-k$ interdiction problem. Modeling the attacker, the upper-level problem maximizes system disruption by choosing lines to de-energize. Given these line failures, the lower-level problem mitigates disruption with an AC maximum load delivery (AC-MLD) redispatch. With a nonconvex feasible region due to the AC power flow equations, the lower-level problem itself is NP-Hard [15], compounding the challenges associated with the bilevel and discrete nature of the overall problem. Thus, approximations, relaxations, and/or decompositions are needed for tractability, as described in Section 3.

We adapt the lower-level formulation and associated notation from the AC-MLD problem in [16]. Let $\mathcal{N}$ be the set of buses, $\mathcal{G}$ the set of generators, and $\mathcal{E}$ the set of lines in the system. Each bus $k \in \mathcal{N}$ has a complex voltage phasor V_k, a shunt admittance Y_k^S, a complex power demand S_k^D (with non-negative real power), and at most one generator with a complex power output of $S_{g_k}^G$, where g_k denotes the generator at bus k. Each line $(i,j) \in \mathcal{E}$ has a series admittance Y_{ij} and a shunt admittance Y_{ij}^c as well as a (possibly complex) turns ratio T_{ij} to permit models of transformers. Each line also has a maximum apparent power flow limit of S_{ij}^u.

Binary variables x_{ij}, z_k^V, and z_g^G denote the statuses of the lines $(i,j) \in \mathcal{E}$, the buses $k \in \mathcal{N}$, and the generators $g \in \mathcal{G}$, respectively. For each of these binary variables, a value of 1 indicates that the component is connected to the system and a value of 0 indicates that the component is disconnected. For each bus $k \in \mathcal{N}$, the continuous variables z_k^D and z_k^S denote the fractions of the satisfied load demand and the shunt admittance connected to the system, respectively, considering a constant power factor.

We model the load delivery and component status with a weighted sum of the satisfied demand and the connection statuses of the buses, generators, and shunts:

$$\sum_{k\in\mathcal{N}} \left(\Re(S_k^D) z_k^D + C^S z_k^S + C^V z_k^V\right) + \sum_{g\in\mathcal{G}} C^G z_k^G \quad (1)$$

where C^V, C^S, and C^G are weighting parameters and $\Re(\cdot)$ takes the real part of a complex number. We select weighting parameters as described in [16] so that maximizing (1) minimizes load shed while attempting to keep components at their pre-contingency statuses. The use of the binary variables for generator and bus statuses and continuous variables for shunt and load statuses prevents infeasibility from islanding.

The lower-level problem redispatches the system for maximum load delivery subject to AC power flow equations and operational limits. Thus the lower-level objective maximizes (1). The AC power flow equations ensure power balance at each bus $k \in \mathcal{N}$:

$$S_{g_k}^G - z_k^D S_k^D - z_k^S \left(Y_k^S\right)^* |V_k|^2 = \sum_{(i,j)\in\mathcal{E}_k} x_{ij} S_{ij} \quad (2)$$

where $\mathcal{E}_k$ is the set of lines connected to bus k and $(\cdot)^*$

denotes the complex conjugate. Constraints modeling the power flowing on each line $(i,j) \in \mathcal{E}$, S_{ij}, are:

$$-M(1-x_{ij}) \leq S_{ij} - (Y_{ij} - Y_{ij}^c)^* \frac{|V_i|^2}{|T_{ij}|^2} + Y_{ij}^* \frac{V_i V_j^*}{T_{ij}} \leq (1-x_{ij})M \quad (3a)$$

$$-M(1-x_{ij}) \leq S_{ij} - (Y_{ij} - Y_{ji}^c)^* |V_j|^2 + Y_{ij}^* \frac{V_j V_i^*}{T_{ij}} \leq (1-x_{ij})M \quad (3b)$$

where M is a "Big-M" constant used in modeling the line status and complex inequalities are interpreted as separate constraints on the real and imaginary parts. Notice that (3) is non-binding when $x_{ij} = 0$ and is effectively an equality constraint modeling Ohm's law when $x_{ij} = 1$. For each line $(i,j) \in \mathcal{E}$, bus $k \in \mathcal{N}$, and generator $g \in \mathcal{G}$, limits on voltage magnitudes, generator outputs, and line flows are:

$$|S_{ij}| \leq S_{ij}^u x_{ij} \quad (4a)$$

$$z_k^V v_k^l \leq |V_k| \leq z_k^V v_k^u \quad (4b)$$

$$z_g^G S_g^{Gl} \leq S_g^G \leq z_g^G S_g^{Gu} \quad (4c)$$

where (4a) is enforced for the apparent power flowing into both terminals of the line, v_k^l and v_k^u are lower and upper bounds on the voltage magnitudes, and S_k^{Gl} and S_k^{Gu} are lower and upper bounds on the generator's complex power outputs.

The upper-level problem selects line statuses x_{ij} to maximize the disruption to the system by minimizing the lower-level objective subject to a cardinality constraint on the number of disconnected lines:

$$\sum_{(i,j)\in\mathcal{E}} x_{ij} \geq N - \kappa \quad (5)$$

where the parameter κ indicates the maximum number of line failures and N is the total number of lines.

The complete $N-k$ interdiction problem is:

$$\min_{x_{ij}\in\mathcal{E}} \quad \zeta \quad (6a)$$

$$\text{s.t. } (\forall (i,j) \in \mathcal{E})$$

$$(5),\ x_{ij} \in \{1,0\} \quad (6b)$$

$$\zeta = \max_{z_k^V, z_g^G, z_k^D, z_k^S} (1) \quad (6c)$$

$$\text{s.t. } (\forall k \in \mathcal{N},\ \forall g \in \mathcal{G},\ \forall (i,j) \in \mathcal{E})$$

$$(2)\text{–}(4),\ z_k^V, z_g^G, \in \{1,0\},\ z_k^D, z_k^S \in (1,0) \quad (6d)$$

The maximizer to this optimization problem yields a single contingency that leads to the worst-case load shedding after allowing for redispatching of the network. However, to make short-term maintenance decisions and to protect against possible threats across the entire network, we may want to find many other failure combinations that lead to significant load shedding. This would require solving the problem multiple times and adding "no-good" cuts after each iteration to avoid repeated solutions. Additionally, the severity of contingencies is load-profile dependent, which means that solutions found at one load condition do not necessarily generalize to the full operating range.

Due to the challenges inherent to this formulation, many solution approaches simplify the problem to yield a structure suited for single-level reformulation. The network flow model used in [11], for example, linearly relaxes the power flow equations in the lower-level problem. Conversely, the two new approaches presented in the next section of this paper train NNs with sampled solutions to the nonlinear lower-level problem (6c)–(6d) to preserve some of the nonlinear behavior and allow for fast online detection of multiple dangerous contingencies across a range of operation.

3. Solution Approaches

This section presents three approaches for solving the $N-k$ interdiction problem (6): a state-of-the-art bilevel programming approach that uses a network flow relaxation of the power flow equations [11], a new approach that uses a NN reformulated as a MILP, and a new multi-step NN regression approach.

3.1. Simplified Bilevel Network Flow Formulation

As a benchmark, we consider the recently proposed approach in [11] that solves the $N-k$ interdiction problem using a network flow relaxation of the power flow equations [17] in the lower-level problem. While the upper-level problem is similar to that of model (6), the lower-level problem is much simpler in that it (i) is linear, (ii) cannot remove other network components, and (iii) focuses only on minimizing the load shed. The formulation in [11] is:

$$\min_{x_{ij}\in\mathcal{E}} \ \zeta \quad (7a)$$

$$\text{s.t. } (\forall (i,j) \in \mathcal{E})$$

$$(5),\ x_{ij} \in \{1,0\} \quad (7b)$$

$$\zeta = \max_{z^D} \sum_{k\in\mathcal{N}} P_k^D z_k^D \quad (7c)$$

$$\text{s.t. } (\forall k \in \mathcal{N},\ \forall g \in \mathcal{G},\ \forall (i,j) \in \mathcal{E})$$

$$P_{g_k}^G - P_k^D z_k^D = \sum_{(i,j)\in\mathcal{E}_k} P_{ij} \quad (7d)$$

$$0 \leq P_g^G \leq P_g^{Gu} \tag{7e}$$

$$-P_{ij}^u x_{ij} \leq P_{ij} \leq P_{ij}^u x_{ij} \tag{7f}$$

$$z_k^D \in (0,1) \tag{7g}$$

The lower-level objective (7c) maximizes the active power demand P_k^D served at each bus $k \in \mathcal{N}$ where the continuous variable z_k^D denotes the fraction of the satisfied load demand connected to the system. Equation (7d) enforces power balance at each bus $k \in \mathcal{N}$ where $P_{g_k}^G$ denotes the active power generated at that bus and P_{ij} denotes the active power flow in each line $(i,j) \in \mathcal{E}_k$. Operational constraints (7e) and (7f) enforce limits on generators $g \in \mathcal{G}$ and thermal limits on lines $(i,j) \in \mathcal{E}$, respectively, where P_g^{Gu} is the upper bound on the generator's active power output and P_{ij}^u is the upper bound on the line's active power flow. Notice that there are no line flow equations in this problem, i.e., no equivalent of (3a) and (3b), and thus this problem is a relaxation of model (6) [17].

To solve (7), the lower-level problem is replaced with its Karush–Kuhn–Tucker (KKT) optimality conditions [18], which are added as constraints to the upper-level problem [19]. This yields a single-level MILP that can be solved with commercial solvers. The work in [11], specifically, gives an explicit reformulation of model (7) that will be used for the rest of this paper for solving the bilevel network flow approach. Along with other techniques for improving the tractability of the formulation, the authors of [11] show that known bounds on the dual variables are key for achieving a tractable implementation.

Once the bilevel network flow formulation problem is solved, its maximizer represents the contingency that produces the maximum load shed. Furthermore, we can find multiple contingencies that lead to significant load shedding by solving the problem in a loop and iteratively adding constraints which prevent the solver from finding previously identified failure combinations.

3.2. Neural Network Formulated with a Mixed-Integer Linear Program (NN-MILP)

We next propose propose a "NN-MILP" approach for solving the $N-k$ interdiction problem. This approach uses a NN ω that maps a network state to the corresponding amount of load shed.

The algorithm first samples τ random failure combinations. For each failure combination, we solve the AC-MLD problem (6c)–(6d) multiple times with different, randomly selected demand profiles within a load variability range. For example, with 50% load variability, a nominal load of 100 MW would be assigned a random value between 50 MW and 150 MW with a uniform distribution. This information is stored in a replay buffer Ω and used for the initial training of the NN. In our numerical experiments, this NN ω consists of an input layer that takes in a vector X with the load profile and the status of each line in the system, two fully connected "hidden" layers with ten neurons and saturating linear activation functions, and a final layer with a one node outputting the estimated load shed.

By choosing piecewise linear activation functions for ω we can perform initial training and then use the weights and biases of the NN to formulate it as a part of the MILP shown in model (8). Fig. 1 shows the components of the NN that are modeled in this MILP. This method was proposed in [20]–[22] and previously applied in the context of power systems in [23], [24]. Here, H denotes the total number of hidden layers, and $\mathcal{L}^{[m]}$ denotes the set of neurons in a particular layer $m \in (1,\ldots,H+1)$. The objective function (8a) maximizes the output of the single neuron of the last layer $O_1^{[H+2]}$ (i.e., the load shed) which is defined in (8l). The output of first layer $O^{[1]}$, defined in (8b), is the same as the state vector X, which consists of the active and reactive demands P_{d_k} and Q_{d_k} at each bus $k \in \mathcal{N}$ and the binary variables x_{ij} that represent the status of each line $(i,j) \in \mathcal{E}$ in the system. The input of each hidden layer neuron $I_l^{[m]}$ is formulated in (8d) using a set of weights $W_{nl}^{[m]}$ and biases $b_{nl}^{[m]}$, where the superscript denotes the layer index m and the subscript denotes the flow of information from a particular neuron n in layer $m-1$ to a particular neuron l in layer m. The output of each hidden layer neuron $O_l^{[m]}$ is formulated using equations (8e)–(8g). These equations use the binary variables $\phi_{l1}^{[m]}$, $\phi_{l2}^{[m]}$, and $\phi_{l3}^{[m]}$ for each neuron l to formulate the linear saturation functions that bound the minimum and maximum outputs of the neurons to be between 0 and 1. Each of these binary

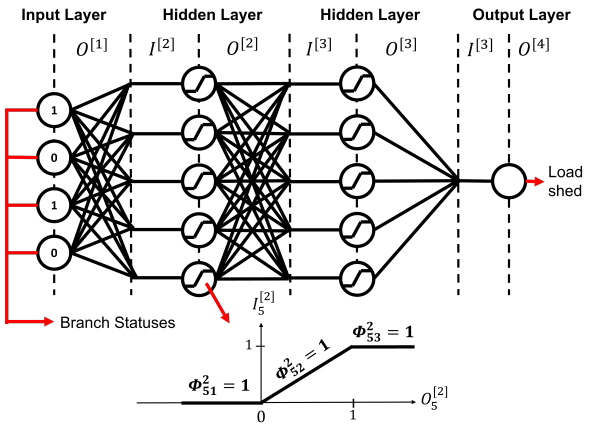

Figure 1. Illustration of a NN with all the components that are modeled as a MILP in model (8).

variables denotes one section of the piecewise linear activation function whose logic is given in (8h)–(8j). Equations (8f) and (8h)–(8i) use "Big-M" constants M_1 and M_2, respectively, to model the activation functions.

$$\max_{X} \quad O_1^{[H+2]} \tag{8a}$$

$$\text{s.t } (\forall l \in \mathcal{L}^{[m]},\ \forall m \in 2,..,H+1, \forall (i,j) \in \mathcal{E})$$

$$O^{[1]} = X \tag{8b}$$

$$x_{ij} \in \{0,1\} \tag{8c}$$

$$I_l^{[m]} = \sum_{n=1}^{L^{[m-1]}} O_n^{[m-1]} W_{nl}^{[m]} + b_{nl}^{[m]} \tag{8d}$$

$$-(1-\phi_{l1}^{[m]}) \leq O_l^{[m]} \leq 1 - \phi_{l1}^{[m]} \tag{8e}$$

$$I_l^{[m]} - M_1(1-\phi_{l2}^{m}) \leq O_l^{m} \leq I_l^{[m]} + M_1(1-\phi_{l2}^{m}) \tag{8f}$$

$$\phi_{l3}^{[m]} \leq O_l^{[m]} \leq 2 - \phi_{l3}^{[m]} \tag{8g}$$

$$-I_l^{[m]} \leq M_2 \phi_{l1}^{[m]} \tag{8h}$$

$$I_l^{[m]} - 1 \leq M_2 \phi_{l3}^{[m]} \tag{8i}$$

$$\phi_{l1}^{[m]} + \phi_{l2}^{[m]} + \phi_{l3}^{[m]} = 1 \tag{8j}$$

$$\phi_{l1}^{[m]}, \phi_{l2}^{[m]}, \phi_{l3}^{[m]} \in \{0,1\} \tag{8k}$$

$$O_1^{[H+2]} = \sum_{n=1}^{L^{[H+1]}} O_1^{[H+1]} W_{n1}^{[H+2]} + b_{n1}^{[H+2]} \tag{8l}$$

As described in Algorithm 1, each maximizer obtained from solving model (8) gives a new failure combination that will be stored in Ω and used in a next round of NN training. This process is repeated in a loop where ω is retrained with the initial random samples and the new failure combinations sampled from the MILP while giving high load shedding samples extra weight during training. Every time the MILP is built, however, new constraints are added to prevent the optimization problem from finding the top 5% contingencies obtained from previous steps, which helps the NN find and learn from different high-load-shed combinations. It is important to note the model (6c)–(6d) does not optimize over the load. Each time the model is solved the model is given a random but constant load profile, and the only decision variable is the vector of binary line statuses. If this vector of line statuses is not part of any sample in Ω, it is added to the collection of samples along with the load profile that was used when finding it. The size and structure of the NN, the choice of activation functions, and the number of loop iterations M, J, and C are all design choices selected based on empirical experiments.

Algorithm 1 NN-MILP training phase

```
Sample τ random failure combinations with multiple random load profiles each. Add to Ω
Calculate the load shed of each sample by solving AC-MLD (6c)–(6d)
for m = 1, 2, ..., M do
    for j = 1, 2, ..., J do        ▷ Train for J epochs
        Train ω with samples from Ω.
    end for
    Build model (8) with a random load profile
    Exclude top 5% of samples from solution X* via no-good cuts
    for c = 1, 2, ... C do         ▷ Get new samples
        Solve model (8). Get the solution X*
        Exclude X* from being a solution again
        if X* is a new contingency then
            Add X* to Ω with its current load profile
            Calculate the load shed of X* using AC-MLD (6c)–(6d).
        end if
    end for
end for
Return ω
```

After training is complete, we can solve model (8) online using the fully trained NN and any load profile within the load variability range to obtain the predicted worst-case contingency from the maximizer. Furthermore, we can find multiple contingencies that lead to large load sheds in real-time by solving the problem in a loop and adding constraints to prevent the solver from finding repeated solutions.

3.3. Multi-step Neural Network Regression (MNNR)

Our second ML approach is multi-step neural network regression (MNNR), in which a NN is trained to predict, from a given initial state, how much additional load shed will result from failing each line in the system. For any given combination of k line failures, the total load shed can be estimated by making k predictions with the NN, failing the lines in succession and recording the predicted load shed. The main differences between the MNNR and the NN-MILP are that (1) the MNNR NNs are not limited to ReLU activation functions since piecewise linearity is not required to formulate an optimization problem and (2) the MNNR NNs predict load shed caused by one line failure at a time, while the NN-MILP predicts load shed for a set of k line failures. The input to the NN is the current state of the network, which consists of the active and reactive power demand at each load and the status of each line. A line status of 1 represents an in-service line, while a

Algorithm 2 MNNR training phase

Initialize NN α, training input X, training output Y
Sample τ random failure combinations. Store in C.
for $i = 1, 2, \ldots, \tau$ **do**
    Initialize all line statuses to active
    Randomly perturb loads
    Initialize current load shed $S_0 = 0$
    for $j = 1, 2, \ldots, k$ **do**
        Save line statuses and loads as x
        for $n = 1, 2, \ldots, N$ **do**
            Fail line n and calculate the resulting load shed s with AC-MLD (6c)–(6d).
            Save the new load shed resulting from the failure of line n as $y_n = s - S_{j-1}$.
        end for
        Fail line $[C_i]_j$ and record the load shed S_j
        Add x to X and y to Y
    end for
end for
Train α on X and Y
Return α

line status of 0 represents a de-energized line. The loads and line statuses are collected into a single vector with length $2\cdot$(number of loads)$+$(number of lines). The NN output is a vector where entry i represents the additional load shed that results from failing line i.

To collect training data, we sample τ random failure combinations and generate corresponding loading conditions by perturbing each load by some randomly sampled multiplier from a specified range. We generate ten different loading conditions for every failure combination. For each failure combination, we begin with all lines in-service and then fail each line in the combination, one at a time. At each step, we use the AC-MLD model (6c)–(6d) to calculate how much additional load shed would result from failing each line in the network.

The NN contains ten hidden layers, of which the first eight use rectified linear unit (ReLU) activation functions and the last two use hyperbolic tangent (tanh) activation functions. The final output layer is linear. We selected this network structure after comparing numerical performance for networks using several common activation functions, including ReLU, tanh, exponential linear unit, and sigmoid functions. The loss function which is minimized during NN training is the mean-squared-error (MSE) between predicted and actual load sheds. Algorithm 2 outlines the MNNR algorithm data generation and training.

After the NN is trained offline, we use it in real time to find sets of line failures that result in high load shed. First, we generate the set of all failure combinations for the network. Next, we use the NN to predict the load shed resulting from each combination. To do so, we initialize the NN input vector with the current load profile and active line statuses. Next, we step through the failure combination, failing one line at a time and summing up the predicted load shed. We compare the predicted load shed for each combination to find the worst contingencies. In total, this process requires $\binom{N}{k} \cdot k$ feedforward passes through the NN.

4. Experiments and Results

This section benchmarks the approaches described in Section 3 for selected test cases from the PGLib-OPF archive [25]. Note that case 118 for $k = 3$ was not included because finding the true top-100 contingencies through brute force evaluation was too computationally intensive. MATLAB R2019b, YALMIP [26], and MATLAB's Machine Learning Toolbox were used to implement the bilevel network flow formulation and the NN-MILP approaches on a computer with a quad-core 1.8 GHz processor and 16 GB of RAM. Julia v1.6.1 with PowerModels.jl [27], PowerModelsRestoration.jl [28], and Flux.jl [29] were used to implement the MNNR algorithm and the AC-MLD formulation on Georgia Tech's PACE cluster, where each node had a quad-core 2.7 GHz processor and 9 GB of RAM. Nonlinear programs (NLP) were solved using Ipopt [30], while MILP problems were solved using Gurobi 9.0 with the default 0.01% relative MIP gap.

To ensure a fair comparison in all experiments, we give both the MNNR and NN-MILP algorithms a failure sample size equal to 10% of the total sample space, i.e:

$$\tau = 0.1 \left(\frac{N!}{(k!(N-k)!)} \right) \tag{9}$$

Each contingency sample is also given ten different load conditions within the specified load variation range.

The following subsections describe the results of experiments comparing the bilevel network flow optimization problem, the MNNR, and the NN-MILP algorithms. We compare the algorithms' performance in finding worst-case contingencies in Section 4.1, computation time in Section 4.2, load shed prediction accuracy for ML approaches in Section 4.3 and the relationship between accuracy and load variability for ML approaches in Section 4.4.

4.1. Identifying the Worst Contingencies

Table 1 shows with an asterisk which approaches were able to find the actual worst contingencies in a variety of test cases and with different numbers of failed components. The load settings for this experiment were the test cases' default nominal values. For $k = 2$, the bilevel network flow formulation and the

Table 1. Percentage of Correct Predictions in Top-100 Combinations

	k	Bilevel	NN-MILP	MNNR
case14_ieee	k=2	60*	52*	**61***
	k=3	65	56	**73***
case24_ieee_rts	k=2	15*	55	**92***
	k=3	47	46*	**94***
case30_ieee	k=2	81*	80*	**89***
	k=3	88	**89**	83
case39_epri	k=2	52*	60	**94***
	k=3	70*	94*	**96**
case57_ieee	k=2	11*	86	**87***
	k=3	82*	**86**	84
case73_ieee_rts	k=2	4	23	**36**
	k=3	**6**	0	**6**
case118_ieee	k=2	47	52	**59**
Average		48	60	**73**
Std. Dev.		30	28	**26**

MNNR performed best, but as we increased k all three approaches became less accurate, succeeding in two out of six experiments.

Another way to compare these approaches is by how many high-quality solutions they identify. Even if an approach finds the absolute worst contingency, it will not necessarily find other failure combinations that result in high load shedding. The opposite is also true: if an approach does not identify the worst contingency it might still be able to find several failure combinations that produce a similar load shedding.

To test this, we obtained the list of the top-100 failure combinations given by each approach and compared them to the actual 100 worst contingencies from model (6) obtained through brute force evaluation of all possibilities. If a predicted top-100 failure combination was also found in the list of the actual worst-case contingencies, we say that that particular combination was correctly identified, even if the exact rank did not match. Table 1 summarizes these results. The numbers in bold correspond to the best-performing method on each test case. While the bilevel network flow formulation was one of the best at finding the very worst contingencies, it did not always find many other high-quality solutions. Both ML approaches found more of the top 100 worst contingencies than the bilevel network flow, and the MNNR method performed best of all.

One last consideration is that there are some cases where several different failure combinations produce nearly the same amount of load shed. Because this subtlety is not obvious from the top-100 rankings alone, we also compute compute running sums of the percent load shed for each top-100 list. The results for two representative cases are shown in Fig. 2, where the entry at failure index i represents the cumulative sum of load shed for contingencies 1 through i. In the figure, the blue line represents the cumulative sum of the true worst-case load sheds, which the best-performing algorithms follow

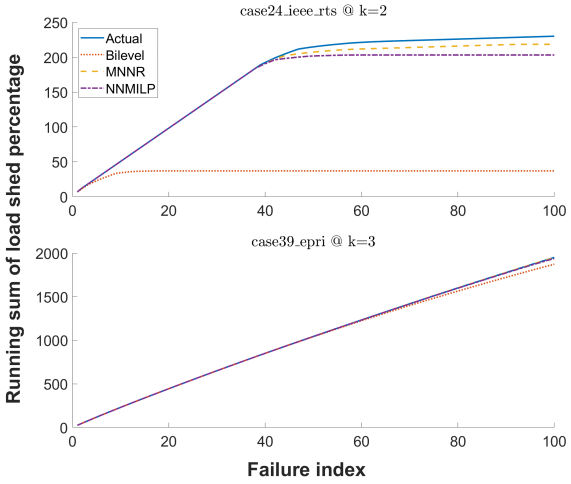

Figure 2. Cumulative sum of the top-100 contingencies' load sheds in descending order for two representative cases. The entry at failure index i is the sum of load shed from contingencies 1 through i. The best algorithms closely follow the true worst-case load shed in blue. Although the bilevel network flow formulation finds the very worst contingencies, it sometimes fails to find additional high-quality solutions, as shown in case 24 when it falls behind after the first few failures.

closely. For many test cases, all three methods successfully identify the top-100 contingencies, as in case39_epri at $k = 3$. However, there are several cases where the bilevel network flow formulation finds a few of the worst contingencies but fails to find additional high-quality solutions. This can be seen in case24_ieee as the bilevel network flow line quickly drops below the true worst-case cumulative sum. The two ML methods track the true worst-case sum better and more consistently.

4.2. Computation time comparison

Computation time is another important metric for comparison. Table 2 shows the computation time during

Table 2. Computation times for representative cases

case30_ieee @ $k = 2$		
	Training	Execution
Bilevel	None	34.2 seconds
NN-MILP	1.1 hours	59.6 seconds
MNNR	1.6 hours	1.6 seconds
case118_ieee @ $k = 2$		
	Training	Execution
Bilevel	None	98 seconds
NN-MILP	10 hours	110 seconds
MNNR	15 hours	692 seconds

training and execution for all three methods on case 30 and case 118 for $k = 2$. Execution time is defined as the amount of time required to predict the top 100 worst contingencies. Times were recorded for all three methods on the same computer with a quad-core 1.8 GHz processor and 16 GB of RAM. The bilevel network flow optimization required no training time, while the ML methods required hours to train the NNs.

During execution, the MNNR method is fast for case 30, but takes almost seven times longer than the bilevel and NN-MILP methods to find the top 100 contingencies for case 118. The MNNR execution time increases drastically with case size because it predicts and compares load shed for all possible failure combinations. This is faster than a brute force method where AC-MLD model (6c)–(6d) is solved for each combination, as the MNNR prediction requires only k feedforward passes through the NN for each combination. In addition, load shed prediction could be parallelized, which would reduce computation time. However, for large-scale networks with thousands of lines, testing every combination may not be feasible. Our future work includes developing heuristics that reduce the number of combinations tested. Similar heuristics have been developed for transmission-line switching problems [31]. We could adapt these strategies, using a line-ranking algorithm to narrow our search to a smaller set of high-impact lines. Alternatively, we could stack the NNs such that the output of one defines the input of the next, and formulate this stack (with all its non-linearities) as an optimization problem as in the NN-MILP approach. The validity of this heuristic would depend on how efficiently the NLP defined by the NN mapping can be solved.

4.3. Load shed prediction quality

Both ML algorithms predict the amount of load shed caused by particular states of the network, which can help operators quantify the true severity of contingencies. To test prediction accuracy, we sample 100 random load profiles and failures, compute their actual percent load shed using the AC-MLD model, and then compare these results to the ones predicted by the fully trained NNs of the NN-MILP and MNNR algorithms. Fig. 3 shows this comparison for three representative cases.

In the figures on the left, the best-algorithms will closely track the thick yellow line representing the true load sheds of the 100 samples. MNNR does a particularly good job with generally smaller deviations than NN-MILP. In the figures on the right, we show box-and-whiskers plots in which the vertical axes represent the absolute load shed prediction errors. While both algorithms yield small median errors in most test cases, MNNR again shows a particularly strong

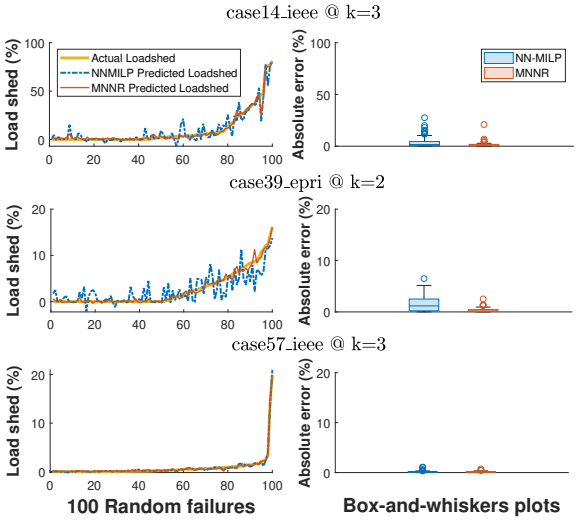

Figure 3. 100 random samples were collected and their true load sheds computed for three representative cases. Left column: comparison between the computed actual load sheds and the load sheds predicted by the NN-MILP and MMNR's fully trained NNs. Right column: Box-and-whisker plots of the ML algorithms' load shed absolute prediction errors as a percent of total system demand. While both NNs are able to approximate the true load sheds with small errors, MMNR does so with more consistency across test cases.

performance with smaller median errors and a smaller error spreads. These experiments show not only the ability of both NNs to predict the load shed of a given network setting, but also how they learn and preserve some of the nonlinearity of the optimization model (6).

4.4. Sensitivity to load profiles and load variability

Our ML approaches are flexible under a range of loading conditions, as they are trained to use both power demand and line statuses to predict load shed. In order to test the ML algorithms' sensitivities to increasing load variability ranges, we modified the IEEE 30-bus case such that changes in load settings yield different worst contingencies. Next, we trained NNs for the MNNR and NN-MILP approaches five separate times, under 20%, 40%, 60%, 80% and 100% load variability. We generated test data by creating 100 different loading conditions for each load variability value and then using model (6) to find the true worst 100 contingencies for each loading condition. We assess solution quality by computing the percentage of the true worst 100 contingencies identified by the ML approaches for each load variability range. Fig. 4 shows the results. The MNNR approach outperforms

NN-MILP, identifying 67% of the worst contingencies even at 100% load variability. The NN-MILP method drops in accuracy significantly from 20 to 40% load variability, but consistently identifies about 50% of the worst contingencies from 40 to 100% load variability. These results demonstrate that the ML approaches, particularly MNNR, can identify most of the worst-case line failures even when load variability is high.

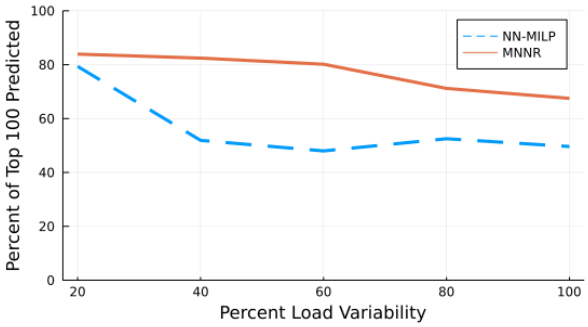

Figure 4. Percentage of correct predictions in top-100 combinations vs. load variability.

5. Conclusion and Future Work

The $N - k$ interdiction problem, which identifies small sets of component failures that result in large load shed, is an important problem for power system operation. The most natural bilevel optimization formulation of this problem is computationally intractable due to the nonconvexity of the AC power flow equations, and thus alternative approaches are needed. This paper compares the performance of three approaches: 1) a state-of-the-art bilevel optimization approach that uses a "network flow" relaxation of the AC power flow equations to obtain a tractable formulation, 2) a neural network trained on a set of line failure scenarios that is formulated as an MILP (NN-MILP), and 3) a multi-step neural network approach which predicts load shed one line at a time and allows for nonlinear activation functions (MNNR).

The results of this comparison on several PGLib-OPF test cases show that these approaches are capable of identifying the worst-case or near-worst-case sets of line failures. The ML approaches are trained on 10% of the failure combination sample space and display good performance on a range of loading conditions. Note that we considered up to $k = 3$ line failures, but the same approach could be extended to $k > 3$ by training on less than 10% of the failure combination sample space if that space is too large. The bilevel network flow and MNNR approaches were most successful at finding the single worst failure in most of the tests, while the NN-MILP and MNNR approaches were best at identifying large sets of high-quality solutions. The MNNR approach also outperformed NN-MILP in maintaining accuracy as load variability increases. The superior performance of MNNR may be due to the use of more general nonlinear activation functions. In addition, it may be easier for a network to map failure to load shed one line at a time in the MNNR approach rather than all k lines at once in the NN-MILP approach. However, the multi-step approach comes at a cost. MNNR takes by far the most time during execution, as our current method to find the worst contingencies requires comparing MNNR predictions on all failure combinations.

These results motivate items for future research:

1. Extend the results to include larger cases, stochastic renewable generation, and interdiction of other network components. Make the MNNR approach more scalable by developing heuristics to reduce the number of computations needed during execution.

2. Assess how well the ML methods perform when trained with fewer samples.

3. Evaluate the possible benefits of using more complex neural network structures and blending model-based and data-driven approaches.

References

[1] J. M. López-Lezama, J. Cortina-Gómez, and N. Muñoz-Galeano, "Assessment of the Electric Grid Interdiction Problem Using a Nonlinear Modeling Approach," *Electr. Power Syst. Res.*, vol. 144, no. 1, pp. 243–254, Mar. 2017.

[2] S. Paul and F. Ding, "Identification of Worst Impact Zones for Power Grids During Extreme Weather Events Using Q-Learning," in *IEEE PES Innovative Smart Grid Technologies Conf. (ISGT)*, 2020.

[3] J. Arroyo and F. Galiana, "On the Solution of the Bilevel Programming Formulation of the Terrorist Threat Problem," *IEEE Trans. Power Syst.*, vol. 20, no. 2, pp. 789–797, May 2005.

[4] "North American Energy Resilience Model," U.S. Department of Energy, Washington, DC, Tech. Rep., Jul. 2019.

[5] B. C. Dandurand, K. Kim, and S. Leyffer, "A Bilevel Approach for Identifying the Worst Contingencies for Nonconvex Alternating Current Power Systems," *SIAM J. Optimiz.*, vol. 31, no. 1, pp. 702–726, Jan. 2021.

[6] F. Capitanescu and L. Wehenkel, "Computation of Worst Operation Scenarios Under Uncertainty for Static Security Management," *IEEE Trans. Power Syst.*, vol. 28, no. 2, pp. 1697–1705, May 2013.

[7] J. M. Arroyo and F. J. Fernández, "Application of a Genetic Algorithm to N-K Power System Security Assessment," *Int. J. of Electr. Power Energy Syst.*, vol. 49, pp. 114–121, Jul. 2013.

[8] K. Sundar, S. Misra, R. Bent, and F. Pan, "Credible Interdiction for Transmission Systems," *IEEE Trans. Control Netw. Syst.*, vol. 8, no. 2, pp. 738–748, 2021.

[9] A. Motto, J. Arroyo, and F. Galiana, "A Mixed-Integer LP Procedure for the Analysis of Electric Grid Security Under Disruptive Threat," *IEEE Trans. Power Syst.*, vol. 20, no. 3, pp. 1357–1365, Aug. 2005.

[10] J. Salmeron, K. Wood, and R. Baldick, "Analysis of Electric Grid Security Under Terrorist Threat," *IEEE Trans. Power Syst.*, vol. 19, no. 2, pp. 905–912, May 2004.

[11] E. S. Johnson and S. S. Dey, "A Scalable Lower Bound for the Worst-Case Relay Attack Problem on the Transmission Grid," May 2021, arXiv: 2105.02801.

[12] S. Rajan, R. S. Kumar, and A. T. Mathew, "Online Static Security Assessment Module Using Artificial Neural Networks," *IEEE Trans. Power Syst.*, vol. 28, no. 4, pp. 4328–4335, Nov. 2013.

[13] B. Donnot, I. Guyon, M. Schoenauer, A. Marot, and P. Panciatici, "Anticipating Contingencies in Power Grids Using Fast Neural Net Screening," May 2018, arXiv: 1805.02608.

[14] K. Shanti Swarup and G. Sudhakar, "Neural Network Approach to Contingency Screening and Ranking in Power Systems," *Neurocomputing*, Neural Networks, vol. 70, no. 1, pp. 105–118, Dec. 2006.

[15] D. Bienstock and A. Verma, "Strong NP-hardness of AC power flows feasibility," *Oper. Res. Lett.*, vol. 47, no. 6, pp. 494–501, Nov. 2019.

[16] C. Coffrin, R. Bent, B. Tasseff, K. Sundar, and S. Backhaus, "Relaxations of AC Maximal Load Delivery for Severe Contingency Analysis," *IEEE Trans. Power Syst.*, vol. 34, no. 2, pp. 1450–1458, Mar. 2019.

[17] C. Coffrin, H. Hijazi, and P. Van Hentenryck, "Network Flow and Copper Plate Relaxations for AC Transmission Systems," in *19th Power Syst. Comput. Conf. (PSCC)*, Jun. 2016.

[18] H. W. Kuhn and A. W. Tucker, "Nonlinear Programming," *Proc. 2nd Berkeley Symp. Math. Stat. Probab.*, pp. 481–492, 1951.

[19] D. G. Luenberger and Y. Yinyu, *Linear and Nonlinear Programming*, 2nd edn. Wiley, 1989.

[20] V. Tjeng, K. Xiao, and R. Tedrake, "Evaluating Robustness of Neural Networks with Mixed Integer Programming," Nov. 2017, arXiv:1711.07356.

[21] M. Fischetti and J. Jo, "Deep Neural Networks and Mixed Integer Linear Optimization," *Constraints*, vol. 23, no. 3, pp. 296–309, Jul. 2018.

[22] B. Grimstad and H. Andersson, "ReLU Networks as Surrogate Models in Mixed-Integer Linear Programs," *Computers & Chemical Engineering*, vol. 131, no. 1, p. 106 580, Dec. 2019.

[23] I. Murzakhanov, A. Venzke, G. S. Misyris, and S. Chatzivasileiadis, "Neural Networks for Encoding Dynamic Security-Constrained Optimal Power Flow," Oct. 2021, arXiv:2003.07939.

[24] A. Venzke, G. Qu, S. Low, and S. Chatzivasileiadis, "Learning Optimal Power Flow: Worst-Case Guarantees for Neural Networks," arXiv:2006.11029, Nov. 2020.

[25] IEEE PES PGLib-OPF Task Force, "The Power Grid Library for Benchmarking AC Optimal Power Flow Algorithms," Aug. 2019, arXiv:1908.02788.

[26] J. Lofberg, "YALMIP: A Toolbox for Modeling and Optimization in MATLAB," in *IEEE Int. Conf. Robotics Automat.*, Sep. 2004, pp. 284–289.

[27] C. Coffrin, R. Bent, K. Sundar, Y. Ng, and M. Lubin, "Powermodels.jl: An Open-Source Framework for Exploring Power Flow Formulations," in *20th Power Systems Computation Conference (PSCC)*, Jun. 2018.

[28] N. Rhodes, D. M. Fobes, C. Coffrin, and L. Roald, "PowerModelsRestoration.jl: An Open-Source Framework for Exploring Power Network Restoration Algorithms," *Electr. Power Syst. Res.*, vol. 190, no. 1, p. 106 736, Jan. 2021, Presented at *21st Power Syst. Comput. Conf (PSCC)*.

[29] M. Innes, E. Saba, K. Fischer, *et al.*, "Fashionable Modelling with Flux," Nov. 2018, arXiv: 1811.01457.

[30] A. Wächter and L. T. Biegler, "On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming," *Math. Program.*, vol. 106, no. 1, pp. 25–57, Mar. 2006.

[31] D. J. Fuller, R. Ramasra, and A. Cha, "Fast Heuristics for Transmission-Line Switching," *IEEE Trans. Power Syst.*, vol. 27, no. 3, pp. 1377–1386, Mar. 2012.