Network State-Based Algorithm Selection for Power Flow Management Using Machine Learning

James E. King, *Student Member, IEEE*, Samuel C. E. Jupe, and Philip C. Taylor, *Senior Member, IEEE*

***Abstract*—This paper demonstrates that machine learning can be used to create effective *algorithm selectors* that select between power system control algorithms depending on the state of a network, achieving better performance than always using the same algorithm for every state. Also presented is a novel method for creating algorithm selectors that consider two objectives. The method is used to develop algorithm selectors for power flow management algorithms on versions of the IEEE 14- and 57-bus networks, and a network derived from a real distribution network. The selectors choose from within a diverse set of power flow management algorithms, including those based on constraint satisfaction, optimal power flow, power flow sensitivity factors, and linear programming. The network state-based algorithm selectors offer performance benefits over always using the same power flow management algorithm for every state, in terms of minimizing the number of overloads while also minimizing the curtailment applied to generators.**

***Index Terms*—Algorithms, machine learning, power system control, power systems, smart grids.**

I. Introduction

POWER flow management (PFM) is an active network management (ANM) approach to alleviating thermal overloading of network branches by curtailing the output of generators causing the overloads. PFM can be used instead of network reinforcement to enable the connection of generators, such as renewables, that may have otherwise been technically, economically, socially or environmentally unviable.

There are numerous algorithms for PFM, and these determine which generators to curtail and by how much each should be curtailed. In the authors' previous work [1], [2], a diverse set of PFM algorithms were tested, including those based on constraint satisfaction [3], optimal power flow (OPF) [4], power flow sensitivity factors (PFSFs) [5], and linear programming (LP) [6]. These algorithms were tested over a range of states on two different networks. It was found that which algorithm was most effective at minimizing the number of overloads, while also minimizing curtailment, depended on the state of the network. If algorithms were appropriately selected for each network state, there was a *potential* performance benefit over always using the same algorithm, even the algorithm that was effective over the most states.

This paper shows that some of this potential performance benefit can be realized by using machine learning with algorithm performance data to create systems (*algorithm selectors*) offline, which select online which algorithm to use for each network state. Algorithm selectors have been created for other applications, particularly within computer science, but this paper is the first work to create selectors for PFM algorithms. Also presented is a novel method for creating selectors that allows two objectives to be considered, in this case minimizing the number of overloads while also minimizing curtailment.

The paper is structured as follows: Section II provides background on algorithm selectors and machine learning; Section III describes the three case study networks used for testing PFM algorithms and selectors; Section IV describes the PFM algorithms tested and selected between in this paper; Section V presents a novel method for creating algorithm selectors for PFM that consider two objectives; Section VI presents the performance of the PFM algorithms and selectors when applied to the case study networks; Section VII discusses the results and the potential benefits of algorithm selectors in future power systems; and Section VIII concludes the paper.

II. Algorithm Selectors

A. Algorithm Selection Problem

The "*problem of selecting an effective, good or best algorithm*" was defined by Rice [7] as the *Algorithm Selection Problem*, posed within the following conceptual model:

- A Problem Space (P), containing all the inputs x that can be provided to the algorithms for a particular application. For PFM each x could represent a different network state.
- A Feature Space (F), containing features f, which are key characteristics extracted from each input $(f(x))$. The features for PFM could be measurements of network state derived from each x.
- An Algorithm Space (A), containing the algorithms a that can be applied to the inputs x.
- A Performance Space (Y), containing measures of performance p of each algorithm a applied to each input x. Each measure can pertain to an algorithm's execution (such as runtime) or output (such as, for PFM, the number of overloads or the amount of curtailment). For n real-valued performance measures, $p \in Y = \mathbb{R}^n$.

Manuscript received April 26, 2014; revised August 22, 2014; accepted September 25, 2014. Date of publication October 13, 2014; date of current version July 17, 2015. This work was supported in part by U.K.'s Engineering and Physical Science Research Council (EPSRC) under grant EP/I031650/1, "The Autonomic Power System". Paper no. TPWRS-00567-2014.

J. E. King is with Parsons Brinckerhoff, Godalming, Surrey, GU7 2AZ, U.K. (e-mail: james.king@pbworld.com).

S. C. E. Jupe is with Nortech Management Ltd., Eckington, Pershore, WR10 3DN, U.K. (e-mail: samuel.jupe@nortechonline.co.uk).

P. C. Taylor is with the School of Electrical and Electronic Engineering, Newcastle University, Newcastle upon Tyne, NE1 7RU, U.K. (e-mail: phil.taylor@ncl.ac.uk).

Digital Object Identifier 10.1109/TPWRS.2014.2361792

Solving a particular Algorithm Selection Problem involves deriving a selection mapping S (an *algorithm selector*), which maps from problem features to an algorithm choice ($S : f(x) \rightarrow a$). Selectors are typically designed to optimize an objective function of the performance measures.

B. Machine Learning for Algorithm Selection

Any decision making technique could be used as an algorithm selector; however, selectors are most often derived using machine learning. This is a branch of artificial intelligence concerned with creating systems that can use experience (data) to improve their performance on particular tasks without explicit programming. Various types of machine learning have been used to create algorithm selectors, as discussed below.

1) Classification: This is the task of predicting to which class—out of a discrete set of classes—an observation belongs. A system that performs classification is known as a *classifier*, and these are trained by *supervised learning* using a number of training examples consisting of observations labelled with target values, in this case the correct classes.

For a classifier used as an algorithm selector, the classes would be the algorithms a to be selected between and the observations would be the features f of the inputs x supplied to the algorithms. Training data for such a selector would consist of feature values labelled with algorithm selections.

Many of the numerous machine learning techniques for producing classifiers have been used to create algorithm selectors, including artificial neural networks [8], case-based reasoning [9], decision trees [10], and support vector machines [11].

2) Regression: This is similar to classification, but a real-valued output is predicted for each observation instead of a discrete class. For algorithm selection, regression models can be used to predict the performance of each algorithm (referred to as *Empirical Performance Models* [12]). The predicted performance $\hat{p}$ can then be used to make selection decisions. Supervised learning is used, with the training data (for each algorithm) consisting of features f labelled with the algorithm's performance p. Regression techniques such as artificial neural networks [8], random forests [12], and ridge regression [13] have been used to create algorithm selectors.

3) Reinforcement Learning (RL): RL techniques learn to make decisions over time by receiving rewards depending on the quality of their past decisions. This allows online learning to take place, which can be useful when it is difficult or impossible to generate training data, for example. RL has been used to create algorithm selectors, such as in [14].

C. Applications

Machine learning has been used in numerous successful algorithm selection applications, although no particular machine learning technique has become a predominant way of creating selectors. The applications have principally been within computer science, such as selecting algorithms for Boolean satisfiability [13], constraint satisfaction [9], sorting [14], machine learning [8], matrix kernels [11], and simulation systems [10]. among others. The authors have previously examined algorithm selection for voltage control [15], although the standard practice for algorithm selection for power system control would be manual studies conducted on network planning timescales by expert users [16].

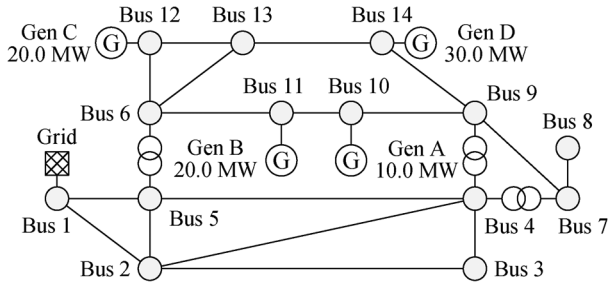

Fig. 1. Modified IEEE 14-bus network.

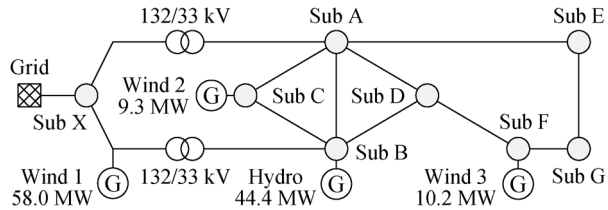

Fig. 2. 33-kV distribution network.

III. Case Study Networks

Three different case study networks are used to test the PFM algorithms and selectors over a range of network states.

A. IEEE 14-Bus Network

The first case study network is a modified version of the IEEE 14-bus system, shown in Fig. 1. This is based on the parameters in [17], with four generators of 10 to 30 MW capacity added. Ratings have been assigned to a number of the network branches so that the generators can cause overloads, thus creating conditions where the PFM algorithms can be applied to alleviate the overloads. To test the algorithms and selectors, different network states were randomly created covering a range of credible conditions, each representing one hour of operation. In each network state, the total load and each of the generators' uncurtailed output were scaled by random variables drawn from independent uniform distributions. Ten thousand network states were used for testing, which was sufficient to reveal any algorithm performance differences.

B. IEEE 57-Bus Network

The second case study network is a modified version of the IEEE 57-bus system, using parameters from [17] and containing three generators that the PFM algorithms can control. As with the IEEE 14-bus system, ratings have been assigned to a number of the network branches and the same process was used to randomly generate 10 000 states for testing.

C. 33-kV Distribution Network

The third case study network is derived from a real distribution network in the United Kingdom and was used in [3]. As shown in Fig. 2, the network is predominantly 33 kV and has four renewable generators of different sizes, which have the potential to cause overloads on some of the branches. A year of

half-hourly profile data [18] for demand, hydro and wind was used to create 17 520 network states for testing.

IV. PFM Algorithms for Case Study

Five PFM algorithms are investigated in this paper. Each uses measurements of network state to calculate real power output limits $P_g^{\lim}$ for each generator g within a network.

A. PFM-CSP

Based on [3], this algorithm formulates PFM as a constraint satisfaction problem (CSP) consisting of:

- A set of variables V; for PFM, each variable represents the output limits of a generator.
- A set of domains D, which are the values the variables are allowed to take; the PFM-CSP algorithm described here uses the discrete domain $\{0\%, 50\%, 100\%\}$ for each variable, where 0% represents a generator limited to zero output, 50% represents output limited to half of rated power, and 100% represents uncurtailed operation.
- A set of constraints C that apply to the variable values; for the PFM-CSP algorithm there is a single constraint which is that any assignment of output limits in V must not result in any branch overloads. This is checked by applying the output limits to generators within a network model internal to the algorithm and executing a load flow.

A solution of a CSP is when the variables V are assigned values from the domains D that also satisfy the constraints C. PFM-CSP uses backtracking search to check assignments of values to variables. As this may return multiple feasible solutions, the solution that minimizes the total generator curtailment is selected and applied to the network under control.

B. PFM-OPF

This algorithm is based on [4] and formulates PFM as an OPF problem, with an objective of maximizing a linear sum of the generator outputs (hence minimizing the curtailment applied) while satisfying standard OPF constraints relating to power balance, maximum generator output, and branch ratings.

When executed, network measurements are used to update the algorithm's OPF model, which is then solved using an Interior Point solver (the OPF solver within PyPower [19], a Python version of MATPOWER [20]). The generator output levels from the solved OPF are then applied as output limits to the generators in the network under control.

C. PFSF-Egal

This algorithm is based on [5] and uses PFSFs to calculate a "egalitarian" proportional signal Φ that all generators are curtailed by. The PFSFs used by this algorithm are $(\partial P_l/\partial P_g)$, which represent the change in real power flow along a branch l due to a change in real power injection by a generator g. These PFSFs are calculated offline from a network model and are only an approximation of the actual power flow changes.

The PFSF-Egal algorithm follows the process of the flowchart Fig. 3, in which $\triangle P_l$ (the change of real power flow required to remove an overload on branch l) is calculated by

$$\triangle P_l = \begin{cases} P_l^{\max} - P_l, & \text{if } P_l \geq 0 \\ -P_l^{\max} - P_l, & \text{otherwise} \end{cases} \tag{1}$$

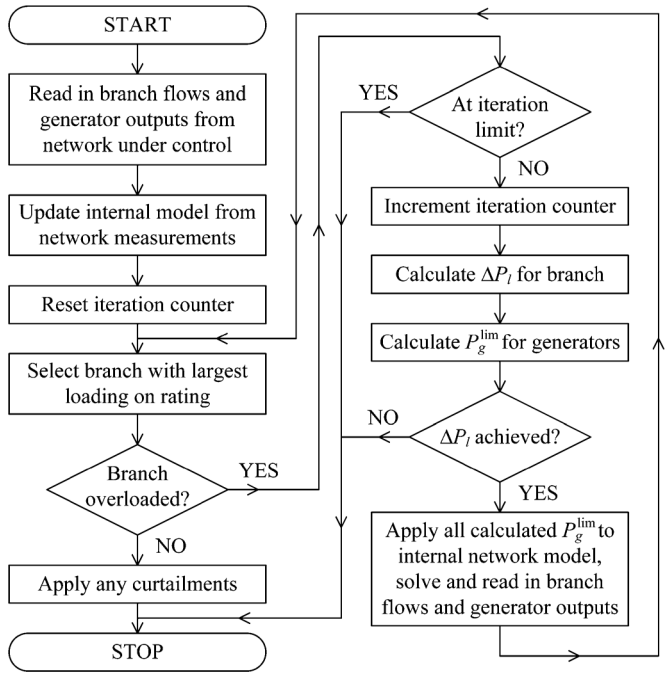

Fig. 3. Flowchart for PFSF-Egal and PFSF-TMA algorithms.

where $P_l^{\max} = \sqrt{(S_l^{\max})^2 - Q_l^2}$, P_l and Q_l are the real and reactive power flows, respectively, along the branch and $S_l^{\max}$ is the branch rating (the branch ratings are adjusted to 99% for all the PFSF-based algorithms in this paper, accounting for PFSFs being an approximation of the actual branch power flow changes). If $Q_l > S_l^{\max}$, the algorithm terminates early as only changing real power cannot remove the overload.

The generator output limits, $P_g^{\lim}$, are derived from $\triangle P_l$ and from the "egalitarian" proportional reduction signal Φ:

$$\Phi = \frac{\triangle P_l}{\sum_{g \in G} \left(\frac{\partial P_l}{\partial P_g}\right) P_g} \tag{2}$$

where P_g is the real power output of generator g from the set of generators G. $P_g^{\lim}$ can be found for each generator:

$$P_g^{\lim} = \begin{cases} (1-\Phi)P_g, & \text{if } 0 \leq \Phi \leq 1 \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

The effect of the generator output changes can be estimated for comparison with the required $\triangle P_l$:

$$\triangle P_l^{\star} = \sum_{g \in G} \left(\frac{\partial P_l}{\partial P_g}\right) \left(P_g - P_g^{\lim}\right). \tag{4}$$

The calculated output limits are then applied to a network model (internal to the algorithm) to check for overloads. If overloads remain in the internal model, the output limits are recalculated using measurements from the internal model, rather than the external network. This process is repeated until there are no overloads in the internal model or an iteration limit is reached (set to 10 in this implementation).

D. PFSF-TMA

This algorithm uses the same process as PFSF-Egal (Fig. 3), except for how $P_g^{\lim}$ and $\triangle P_l^{\star}$ are calculated. Rather than all

generators being curtailed by the same proportion, PFSF-TMA only curtails the generators that are "technically most appropriate" for removing an overload according to their PFSF relating to the overloaded branch.

First, $\triangle P_l^\star$ is set to 0, then the generators are sorted by the value of their PFSF relating to the overloaded branch l. This sort is ascending if $\triangle P_l$ is positive and descending otherwise. For each generator g in turn, $P_g^{\lim}$ is calculated by

$$P_g^{\lim} = P_g - \frac{\triangle P_l - \triangle P_l^\star}{\left(\frac{\partial P_l}{\partial P_g}\right)}. \tag{5}$$

As importing real power is not allowed, negative values of $P_g^{\lim}$ are set to 0. $P_g^{\lim}$ is then used to update $\triangle P_l^\star$:

$$\triangle P_l^\star := \triangle P_l^\star + \left(\frac{\partial P_l}{\partial P_g}\right)\left(P_g - P_g^{\lim}\right). \tag{6}$$

Equations (5) and (6) are repeated for all remaining (sorted) generators in G or until $\triangle P_l^\star$ equals $\triangle P_l$. PFSF-TMA then proceeds in the same way as PFSF-Egal.

E. PFSF-LP

This algorithm uses PFSFs and formulates PFM as an LP, based on [6]. The LP considers the dummy variables $\triangle P_g = P_g - P_g^{\lim}$, representing the amount of curtailment (with respect to the current output P_g) applied to generator g.

The objective is to minimize the total curtailment:

$$\min \sum_{g \in G} \triangle P_g. \tag{7}$$

This objective is subject to constraints relating to:

- Non-negativity of curtailment (generators are not allowed to increase their power output):

$$\triangle P_g \geq 0 \quad \forall g \in G \tag{8}$$

- Maximum available curtailment (generators are not allowed to import real power):

$$\triangle P_g \leq P_g \quad \forall g \in G \tag{9}$$

- Real power flows for each branch l being within limits:

$$-P_l^{\max} \leq P_l - \sum_{g \in G}\left(\frac{\partial P_l}{\partial P_g}\right)(\triangle P_g) \leq P_l^{\max} \tag{10}$$

A publicly-available Python package [21] is used to formulate and solve the LP, with the resultant $\triangle P_g$ values used to derive new generator output limits ($P_g^{\lim} = P_g - \triangle P_g$).

V. Method for Creating an Algorithm Selector for Power Flow Management

This section presents a novel method for creating network state-based algorithm selectors for PFM. These selectors are in the form of classifiers that take measurements of network state as input and provide an algorithm selection as output. The method presented is agnostic of the machine learning technique used for classifier training, and pre-processes training data so that the two objectives of minimizing the number of overloads, while minimizing generator curtailment, can be considered.

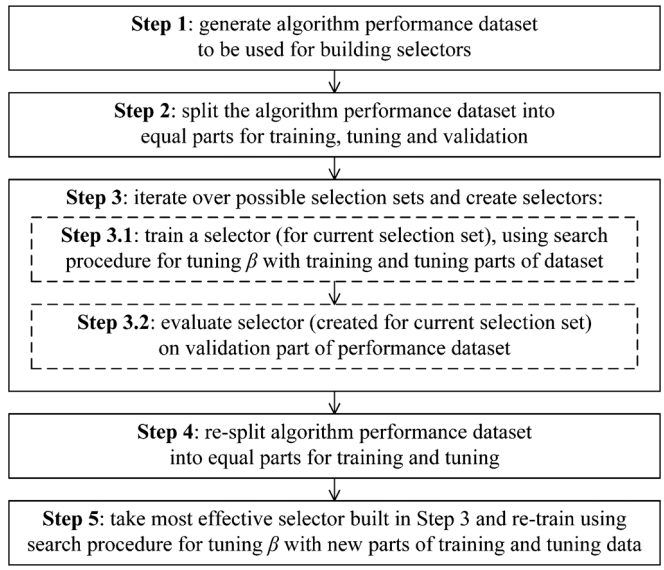

Fig. 4. Flowchart of overall method used to create a selector.

An overview of the method is provided in Fig. 4, with a detailed explanation provided below. The method begins with performance data being generated for each of the PFM algorithms (Steps 1 and 2). Using this data, selectors are created and evaluated that select between different sets of PFM algorithms (Step 3). Finally, the most effective of these selectors is re-trained using the complete performance dataset (Steps 4 and 5).

A. Generating an Algorithm Performance Dataset

In Step 1, a performance dataset is created by testing each of the PFM algorithms on a number of states (30 000 in this paper) generated for each network. The performance dataset is used to create the selectors, and the states within it are separate from the states described in Section III that are used for testing the performance of the selectors and algorithms. For the IEEE 14- and 57-bus networks the same process as described in Section III-A is used to generate the random states, whereas for the 33-kV distribution network the states are generated by replacing the profile data inputs described in Section III-C with independent uniform random variables.

Following testing, the performance of each PFM algorithm a on each state i is calculated for two performance measures o: $p_{o,i,a}$. The two performance measures considered are 1) the number of overloads remaining after each algorithm is applied, and 2) the amount of curtailment each algorithm applies. Next, the features are extracted, which are provided as inputs to the selectors and are all real-valued continuous variables. For the IEEE 14- and 57-bus networks, the features extracted are the total demand in the networks and the outputs of each of the generators. For the 33-kV distribution network, the features extracted are the profile values for demand, hydro and wind.

In Step 2, the performance dataset (including features) is split in to 3 equal parts, each of 10 000 states, ready for use during Step 3. Two parts (*training* and *tuning*) are used to create selectors in Step 3.1, while the remainder (the *validation* part) is used when evaluating selectors in Step 3.2.

B. Selectors Considering Different Sets of Algorithms

Depending on nuances of the network, states, algorithms and machine learning technique used, selectors created to select between particular sets of algorithms (a *selection set* A_s) may be more effective than selectors using other selection sets. Which selection sets will lead to the most effective selectors is not known *a priori*, so in Step 3 a number of selectors are created (Step 3.1) using different selection sets. In this work, the selection sets considered are the complete set of PFM algorithms along with pair-wise combinations of the most effective algorithm and each of the top 3 other algorithms.

The effectiveness of each of the selectors, created with the different selection sets, is evaluated by observing which algorithms they select for the states in the validation part of the performance data. From the performance of the algorithms selected, the most effective selector is that which minimizes the total number of overloads ($\sum p_{\text{overloads}}$) while also minimizing curtailment. The validation data used to evaluate the selectors is separate from the data used when creating the selectors (the training and tuning parts) to help avoid bias.

C. Creating Selectors Using Multi-Label Weighted Learning

An identical process is used in Steps 3.1 and 5 to create a selector for a given selection set and training and tuning data. The selectors are in the form of classifiers, which are trained using a number of established machine learning techniques. These techniques are implemented in the Weka machine learning software [22], with default parameters used. Furthermore, a *multi-label*, *weighted* variant of supervised learning is used during classifier training.

The *multi-label* aspect of the learning process refers to the fact that each network state used as a training example can be labelled with more than one algorithm. This is implemented by creating duplicate training examples for each network state, each labelled with a different algorithm. Each example is then *weighted* according to the performance of the algorithm used as a label. During classifier training, maximizing the sum of weights of classified examples is sought, thus the weights direct the classifier to favour better-performing algorithms.

The weight $W_{i,a}$ of each training example (for state i and algorithm a) considers both the overload and curtailment performance of each algorithm, and is given by

$$W_{i,a} = \alpha \cdot w_{\text{overloads},i,a} + \beta \cdot w_{\text{curtailment},i,a} \qquad (11)$$

where α and β are coefficients and the component $w_{o,i,a}$ for each performance measure o is

$$w_{o,i,a} = \left| p_{o,i,a} - \max_{a \in A_s} p_{o,i,a} \right|. \qquad (12)$$

The coefficient for the component of the weight reflecting overloads (α) is fixed at 1.0. The value of the coefficient for curtailment performance (β) is not fixed and is tuned using the procedure listed in Fig. 5, with the aim of finding the largest value of β for which the selector's overload performance does not suffer. This creates a selector that considers both minimizing the number of overloads and minimizing the amount of curtailment. The procedure iterates between training and evaluating a selector based on a particular β value (Lines 3–7) and determining the next β value to try (Lines 10–16).

Require: A_s, α, $\triangle\beta_{\max}$, $\triangle\beta_{\min}$, $l_{\max}$, training/tuning data
1: Initialize $\beta := 0.0$, $\beta_{\text{best}} := 0.0$, $\triangle\beta := \triangle\beta_{\min}$
2: **For** $l := 1, l_{\max}$
3: Calculate $W_{i,a}\ \forall\ a \in A_s, i \in$ training data
4: Create weighted training dataset using $W_{i,a}$
5: Train selector using weighted training dataset
6: Evaluate trained selector on tuning dataset
7: Calculate $\sum p_{\text{overloads}}$ of selector on tuning dataset
8: **If** $\beta = 0$ **then**
9: $\sum p^{\text{target}}_{\text{overloads}} := \sum p_{\text{overloads}}$
10: **If** $\sum p^{\text{target}}_{\text{overloads}} = \sum p_{\text{overloads}}$ **then**
11: $\beta_{\text{best}} := \beta$
12: $\triangle\beta := \min(2\triangle\beta, \triangle\beta_{\max})$
13: $\beta := \beta + \triangle\beta$
14: **If** $\triangle\beta > \triangle\beta_{\min}$ **then**
15: $\triangle\beta := \triangle\beta_{\min}$
16: $\beta := \beta_{\text{best}} + \triangle\beta$
17: **Else**
18: **Break**
19: **Return** $\beta := \beta_{best}$

Fig. 5. Procedure used to tune β.

During the training part of the procedure, a multi-label weighted training dataset is created based on supplied training data (Lines 3–4) with weights calculated from the current value of β. This training dataset is then used to train the selector (a classifier) by calling Weka (Line 5). The trained selector is then applied to the tuning data (Line 6), making selections based on the feature values of each state. The total number of overloads $(\sum p_{\text{overloads}})$ of the algorithms selected for each of the states is then calculated (Line 7).

A form of incremental search is used to determine the next value of β to try. Starting at zero, β is iteratively incremented by a step $\triangle\beta$. The step size is not fixed, but starts at $\triangle\beta_{\min}$ and is doubled at every iteration up to a ceiling of $\triangle\beta_{\max}$. If, for some value of β, the performance $\sum p_{\text{overloads}}$ of a selector differs from the selector with $\beta = 0.0$ $(\sum p^{\text{target}}_{\text{overloads}})$, the search returns to the last β value (β_{best}) with the step $\triangle\beta$ reset to $\triangle\beta_{\min}$. The search continues until a step of $\triangle\beta_{\min}$ results in a selector with $\sum p_{\text{overloads}} \neq \sum p^{\text{target}}_{\text{overloads}}$ or the number of iterations l reaches the limit $l_{\max}$.

D. Machine Learning Techniques

To show that machine learning can be used to create effective algorithm selectors for PFM, the method described in Fig. 4 was followed for each network using three established machine learning techniques that can be used to produce classifiers and that can have real-valued features as inputs:

1) Artificial Neural Network (ANN): Inspired by biological neural networks, an ANN consists of layers of artificial neurons, with connections between each layer [23]. One layer of neurons is the "input" layer with one neuron per input feature. There is then a number of "hidden" layers where the outputs of the neurons in one layer are connected to the inputs of the neurons in the next layer. Connected to the final hidden layer is an "output" layer of neurons, with a neuron for each class output. The output of a neuron depends on the outputs of the neurons that it is connected to, and during training the strength of the connections are varied with the aim of obtaining the correct class outputs

TABLE I
ALGORITHM AND SELECTOR PERFORMANCE OVER THE 10 000 NETWORK STATES TESTED ON THE IEEE 14-BUS NETWORK

Algorithm	Overloads (count)	Curtailment (MWh)
No algorithm applied	5,345	0.00
PFM-CSP	45	58,855.10
PFM-OPF	**24**	**19,544.04**
PFSF-Egal	66	33,900.57
PFSF-TMA	60	21,797.45
PFSF-LP	1,172	19,503.16
ANN-based selector	9	20,093.96
DT-based selector	12	21,118.02
RF-based selector	**8**	**20,073.74**
Oracle selections	0	19,311.40

TABLE II
ALGORITHM AND SELECTOR PERFORMANCE OVER THE 10 000 NETWORK STATES TESTED ON THE IEEE 57-BUS NETWORK

Algorithm	Overloads (count)	Curtailment (MWh)
No algorithm applied	25,377	0,000.00
PFM-CSP	3,660	982,967.03
PFM-OPF	**1,367**	**749,899.56**
PFSF-Egal	24,823	30,780.23
PFSF-TMA	24,247	43,312.05
PFSF-LP	12,863	405,575.30
ANN-based selector	**787**	**859,837.09**
DT-based selector	808	866,763.03
RF-based selector	788	870,839.62
Oracle selections	768	821,087.30

from the ANN. The "MultilayerPerceptron" ANN implementation from the Weka machine learning software is used in this paper.

2) Decision Tree (DT): These consist of a number of branches, which are split based on the values of the input features, leading to leaves that determine the output class. Different techniques for training DTs exist, which determine which features and values are used for splitting branches. Weka's "J48" implementation of the C4.5 tree learning technique [24] is used in this paper.

3) Random Forest (RF): These comprise a number of "random trees", which are DTs where the features used to split the branches are decided randomly during training [25]. The outputs of the trees are combined to make a class prediction. Weka's "RandomForest" implementation is used in this paper.

VI. RESULTS

Tables I–III show, for the IEEE 14-bus, IEEE 57-bus, and 33-kV distribution networks, respectively, the performance of the PFM algorithms from Section IV and the network state-based algorithm selectors created using the method from Section V. The network states described in Section III were used for testing the algorithms and selectors, and these are separate from the states used when creating the selectors. The algorithms and selectors that are most effective overall at minimizing the number of overloads, while minimizing curtailment, are indicated in bold. Note our previous work has reported the PFM algorithm performance results for the IEEE 14-bus network [1] and the 33-kV distribution network [2].

TABLE III
ALGORITHM AND SELECTOR PERFORMANCE OVER THE 17 520 NETWORK STATES TESTED ON THE 33-KV DISTRIBUTION NETWORK

Algorithm	Overloads (count)	Curtailment (MWh)
No algorithm applied	14,779	0.00
PFM-CSP	194	76,910.75
PFM-OPF	**0**	**40,246.45**
PFSF-Egal	3,594	59,905.56
PFSF-TMA	1,903	39,694.49
PFSF-LP	1,819	39,743.66
ANN-based selector	**0**	**39,980.87**
DT-based selector	0	40,246.45
RF-based selector	0	40,195.28
Oracle selections	0	39,879.06

The top row of each table shows the performance achieved without an algorithm being applied, while the last row of the tables gives the performance achieved if selection decisions for each state are made based on an "oracle" that has perfect *a priori* knowledge of which algorithms will be most effective. In other words, this is the performance achieved with optimal selection decisions made for each state.

For the IEEE 14-bus network (Table I), PFM-OPF is the most effective PFM algorithm as it removes the most overloads and does so without excessive curtailment. If the PFM algorithms are correctly selected between for each state, all overloads can be removed and the total amount of curtailment applied is less than that of each of the PFM algorithms. The method from Section V is able to produce effective selectors with all of the three machine learning techniques tried. Each selector is able to remove more overloads than PFM-OPF, although the selectors apply slightly more generator curtailment. Although none of the selectors matches the performance of the "oracle" and a few overloads remain, the overloads that remain are small in magnitude. For example, the remaining overloads for the best performing selector (RF) have an average magnitude of 3.86% of branch rating.

For the IEEE 57-bus network (Table II), PFM-OPF is again the best performing PFM algorithm, although it leaves a larger number of overloads than on the IEEE 14-bus network. The relative performance of the other PFM algorithms is considerably worse than on the IEEE 14-bus network, particularly PFSF-Egal and PFSF-TMA. Although none of the PFM algorithms can remove all overloads, selecting between them depending on the network states allows for the number of overloads to be reduced significantly. The selectors created using the method from Section V are effective at outperforming the individual PFM algorithms and get close to the performance of the "oracle". While there are a large number of overloads remaining, these have an average magnitude of 3.89% of branch rating for the best performing selector (ANN).

For the 33-kV distribution network (Table III), PFM-OPF is again the most effective PFM algorithm and can remove all overloads, unlike the other PFM algorithms. The performance benefit of state-based algorithm selection is a reduction in curtailment. The ANN- and RF-based selectors are able to realize some of this performance benefit, outperforming PFM-OPF, while the DT-based selector does not deliver any benefit as it

just selects PFM-OPF for every state. The ANN-based selector is the most effective and is closer to the performance of the "oracle" than to the performance of PFM-OPF.

VII. Discussion

Machine learning has been shown to be capable of creating effective network state-based algorithm selectors for power flow management. As can be seen in Tables I–III, the choice of machine learning technique affects the performance of the selector and there may be other techniques that are able to produce more effective selectors, getting closer to the "oracle" and thus realizing more of the potential benefit of network state-based algorithm selection. Furthermore, performance may be improved by using different sets of features or by using different amounts of data when creating the selectors.

Network state-based algorithm selection offered a performance benefit on the three case study networks used in this paper. Although the networks tested were diverse in their scale and character, selection may not offer a performance benefit on all networks. For example, the same PFM algorithms have been applied to 10 000 random states of a radial distribution network featuring just two generators. The PFM-OPF algorithm was found to be the most effective algorithm for almost every state, and the performance benefit from algorithm selection was negligible. This suggests that networks need to have a certain level of complexity before network state-based algorithm selection can offer a performance benefit for PFM.

Increased complexity is an expected characteristic of future power systems, along with increased uncertainty [26]. These characteristics will affect the performance of algorithms for PFM and for other power system applications, and thus influence the potential benefits from algorithm selection.

Complexity, such as that engendered by a large increase in the number and variety of devices that can be actively managed, will require algorithms that are both scalable and able to handle discrete and continuous control actions. Uncertainty, such as that arising from widespread intermittent renewable generation or noisy measurements, will require algorithms that can both handle uncertainty and be able to respond quickly to rapidly changing network conditions. There may also be multiple objectives being optimized, which may conflict with each other and could be constantly changing. Furthermore, algorithms may be assigned different parameter values—such as different OPF cost functions—that need to be selected carefully for particular conditions.

It is therefore reasonable to expect that no algorithm will always be the most effective for multiple, varying objectives, across all conditions encountered on all networks, and with computational constraints considered. This intuitive notion of no algorithm always being the most effective has some theoretical support, at least in the case of search and optimization algorithms, from the "no free lunch" theorem [27]. With no algorithm being the most effective it is therefore useful to select between the algorithms to obtain improved performance.

The characteristics of future power systems will require algorithm selection decisions to be made frequently and quickly in response to changes in network conditions and objectives and for the decisions to be made with uncertainty of the network state and algorithms' performance. Therefore, *robust*, *automated* systems to select algorithms online will be required. This paper has demonstrated that machine learning can be used to create effective algorithm selectors, which are ideal for such automated selection. Although uncertainty was not considered, particular machine learning techniques can be made robust to uncertainty. A machine learning-based approach to create selectors only requires data on the behavior of the algorithms (their performance), without needing any knowledge about their internal processes. Furthermore, such selectors can deliver selection decisions rapidly—for example, the selectors in this paper made selection decisions in less than 1 ms—and can scale well for larger or more complex networks.

One disadvantage of a machine learning-based approach is the time taken to create selectors. For example, to follow the method described in Section V for one network took over a day on a multi-core Intel Core i7-based laptop, with the majority of that time dedicated to generating the algorithm performance data. Additional time may be required for larger or more complex networks, or by increasing the number or complexity of the algorithms being tested. However, increased instrumentation within power systems will allow algorithm performance data to be obtained from the power system itself, reducing the time required, as well as providing data that can be used as the inputs to selectors. Furthermore, ever increasing computational power will help speed up the process of creating selectors using the performance data.

VIII. Conclusions

This paper has demonstrated that machine learning can be used to create effective network state-based algorithm selectors for power flow management. A method has been presented for creating algorithm selectors that can consider two objectives, in this case minimizing the number of overloaded branches within a network while also minimizing the curtailment applied to generators.

The selectors presented in this paper were able to outperform each of the individual PFM algorithms tested; however, they did not always make the optimal selection decisions. Future work will explore ways to further improve the performance of algorithm selectors for PFM, including trialling different machine learning techniques and different forms of selector—such as those based on Empirical Performance Models—as well as investigating creating selectors that are robust to uncertainty in input data and algorithm performance. The PFM algorithms will be tested on other networks and selectors created to realize any potential performance benefits from network state-based algorithm selection.

Power flow management is just one power system application where there is a choice between two or more algorithms that could be applied. Therefore, other power systems applications, such as voltage control, should be investigated to see if they would benefit from network state-based algorithm selectors created using machine learning.

REFERENCES

[1] J. E. King, S. C. E. Jupe, and P. C. Taylor, "The potential of network state-based algorithm selection to improve power flow management," in *Proc. IEEE Power & Energy Society General Meeting*, Washington, DC, USA, 2014.

[2] "Performance evaluation of control algorithms for active distribution networks—The potential for algorithm selection," in *Proc. CIGRÉ Session*, Paris, France, 2014.

[3] M. J. Dolan, E. M. Davidson, G. W. Ault, K. R. W. Bell, and S. D. J. McArthur, "Distribution power flow management utilizing an online constraint programming method," *IEEE Trans. Smart Grid*, vol. 4, no. 2, pp. 798–805, 2013.

[4] M. J. Dolan, E. M. Davidson, I. Kockar, G. W. Ault, and S. D. J. McArthur, "Distribution power flow management utilizing an online optimal power flow technique," *IEEE Trans. Power Syst.*, vol. 27, no. 2, pp. 790–799, May 2012.

[5] S. Jupe, P. C. Taylor, and A. Michiorri, "Coordinated output control of multiple distributed generation schemes," *IET Renew. Power Gener.*, vol. 4, no. 3, pp. 283–297, 2010.

[6] I. Skokljev, V. Maksimovic, and H. Weber, "Symbolic analysis congestion management," in *Proc. IEEE Power Tech*, Bologna, Italy, 2003, vol. 2.

[7] J. Rice, "The algorithm selection problem," *Adv. Comput.*, vol. 15, pp. 65–118, 1976.

[8] K. A. Smith-Miles, "Towards insightful algorithm selection for optimisation using meta-learning concepts," in *Proc. IEEE Int. Joint Conf. Neural Networks (IJCNN 2008)*, 2008, pp. 4118–4124.

[9] L. Kotthoff, I. P. Gent, and I. Miguel, "A preliminary evaluation of machine learning in algorithm selection for search problems," in *Proc. 4th Int. Symp. Combinatorial Search (SoCS-2011)*, 2011, pp. 84–91.

[10] R. Ewald, J. Himmelspach, and A. M. Uhrmacher, "An algorithm selection approach for simulation systems," in *Proc. 22nd Workshop Principles of Advanced and Distributed Simulation*, 2008, pp. 91–98.

[11] R. Vuduc, J. W. Demmel, and J. A. Bilmes, "Statistical models for empirical search-based performance tuning," *Int. J. High Perform. Comput. Appl.*, vol. 18, no. 1, pp. 65–94, Feb. 2004.

[12] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, "Algorithm runtime prediction: Methods & evaluation," *Artif. Intell.*, vol. 206, pp. 79–111, Jan. 2014.

[13] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "SATzilla: Portfolio-based algorithm selection for SAT," *J. Artif. Intell. Res.*, vol. 32, pp. 565–606, Jun. 2008.

[14] M. G. Lagoudakis and M. L. Littman, "Algorithm selection using reinforcement learning," in *Proc. 17th Int. Conf. Machine Learning*, San Francisco, CA, USA, 2000.

[15] J. E. King, S. C. E. Jupe, and P. C. Taylor, "Autonomic control algorithm selection in decentralised power systems: A voltage control case study," in *Proc. 22nd Int. Conf. Exhib. Electricity Distribution (CIRED)*, Stockholm, Sweden, 2013.

[16] G. O. W. Grond, J. Morren, and J. G. Slootweg, "Requirements for advanced decision support tools in future distribution network planning," in *Proc. 22nd Int. Conf. Exhib. Electricity Distribution (CIRED)*, Stockholm, Sweden, 2013.

[17] University of Washington, Power Systems Test Case Archive, 2013 [Online]. Available: http://www.ee.washington.edu/research/pstca/

[18] United Kingdom Generic Distribution System [Online]. Available: http://monaco.eee.strath.ac.uk/ukgds

[19] PYPOWER 4.0.1: Python Package Index [Online]. Available: https://pypi.python.org/pypi/PYPOWER/4.0.1

[20] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Trans. Power Syst.*, vol. 26, no. 1, pp. 12–19, Feb. 2011.

[21] PuLP 1.1: Python Package Index [Online]. Available: https://pypi.python.org/pypi/PuLP/1.1

[22] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *SIGKDD Explor.*, vol. 11, no. 1, pp. 10–18, Nov. 2009.

[23] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms Information Theory, Inference, and Learning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, Dec. 2005.

[24] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann, 1993.

[25] L. Breiman, "Random Forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[26] S. D. J. McArthur, P. C. Taylor, G. W. Ault, J. E. King, D. Athanasiadis, V. D. Alimisis, and M. Czaplewski, "The autonomic power system network operation and control beyond smart grids," in *Proc. 3rd IEEE PES Int. Conf. Exhib. Innovative Smart Grid Technologies (ISGT Europe), 2012*, Berlin, Germany, 2012.

[27] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Computat.*, vol. 1, no. 1, pp. 67–82, 1997.

James E. King (S'12) received the M.Eng. (Hons) degree in new and renewable energy from Durham University, Durham, U.K., in 2010. He is pursuing the Ph.D. degree at Newcastle University, Newcastle upon Tyne, U.K.

He is currently an engineer at Parsons Brinckerhoff, U.K. His research interests include power systems control and applications of machine learning.

Mr. King is a member of the IET in the U.K. and is the Chairman of CIGRÉ-UK's Next Generation Network (NGN).

Samuel C. E. Jupe received the M.Eng. (Hons) and Ph.D. degrees from Durham University, Durham, U.K., in 2006 and 2010, respectively.

He is currently the Network Innovation Manager for Nortech Management Limited, U.K. He was previously a Senior Engineer at Parsons Brinckerhoff, Manchester, U.K., where he delivered a number of innovation projects on behalf of U.K. distribution network operators. His research interests include the innovative development of electricity networks, including the demonstration of advanced fault level management technologies, real-time thermal ratings and distributed generation output control algorithms.

Dr. Jupe is a member of the IET, a Chartered Engineer in the U.K., and is the U.K. Regular Member for CIGRÉ's Study Committee C6. He was part of consortium with Prof. Taylor that received an IET Innovation Award in 2010.

Philip C. Taylor (M'01–SM'08) received an Engineering Doctorate from the University of Manchester Institute of Science and Technology (UMIST), Manchester, U.K., in 2001.

He is the Professor of Electrical Power Systems and Director of the Institute for Sustainability at Newcastle University, Newcastle upon Tyne, U.K. His research focuses on the challenges associated with the widespread integration and control of distributed/renewable generation in electrical distribution networks.

Prof. Taylor is a Fellow of the IET and is a Chartered Engineer in the U.K. He is a member of the EPSRC Peer Review College, of the EPSRC Strategic Advisory Team on Energy, and of the Smart Grids European Technology Platform Working Group 1 (Network Operations and Assets).