Algorithm-Data Driven Optimization of Adaptive Communication Networks

Mu He, Patrick Kalmbach, Andreas Blenk, Wolfgang Kellerer
Technical University of Munich, Germany
Email: {mu.he, patrick.kalmbach, andreas.blenk, wolfgang.kellerer}@tum.de

Stefan Schmid
Aalborg University, Denmark
schmiste@cs.aau.dk

Abstract—**This paper is motivated by the emerging vision of an *automated* and *data-driven* optimization of communication networks, making it possible to fully exploit the flexibilities offered by modern network technologies and heralding an era of fast and *self-adjusting* networks. We build upon our recent study of machine-learning approaches to (statically) optimize resource allocations based on the data *produced by network algorithms* in the past. We take our study a crucial step further by considering *dynamic* scenarios: scenarios where communication patterns can change over time. In particular, we investigate network algorithms which learn from the traffic distribution (the *feature vector*), in order to predict global network allocations (a *multi-label* problem).**

As a case study, we consider a well-studied k-median problem arising in Software-Defined Networks, and aim to imitate and speedup existing heuristics as well as to predict good initial solutions for local search algorithms. We compare different machine learning algorithms by simulation and find that neural network can provide the best abstraction, saving up to two-thirds of the algorithm runtime.

I. Introduction

Communication networks are in constant flux: traffic patterns, resource demands, applications, and user locations change continuously. Such dynamic demands stand in stark contrast to the usually static resource supply in traditional communication networks, forcing operators to either over-provision their network or risk an unacceptable performance under peak demands.

The advent of Software-Defined Networks (SDNs) and Network Virtualization (NV) introduces great flexibilities [1]: such technologies, in principle, enable a more adaptive, automated, and programmatic resource allocation scheme, according to the current demands. For example, a logically centralized SDN controller may leverage collected traffic statistics to adjust, reactively and in a fine-grained manner, traffic engineering decisions or even bandwidth reservations. Moreover, by decoupling applications from the constraints of the underlying physical infrastructure, virtualization enables a flexible allocation of resources and network functions.

While increasingly flexible technologies to realize and manage adaptive communication networks are emerging, today, only little is known about how to algorithmically exploit them. In fact, most communication networks are either not adapted to the workload at all, or reconfigured infrequently and manually, e.g., when long-term shifts in the traffic matrix are observed. In other words, most existing communication networks are to a substantial extent oblivious to the current network demands.

Data-driven communication networks promise to overcome these limitations. According to this vision, communication networks leverage existing network data to self-optimize, in an *automated* fashion and at runtime, towards the demands. Thus, data-driven communication networks have the potential to fully exploit the flexibilities offered by recent networking technologies, and herald a new networking era of truly "intelligent" networks.

This paper is motivated by the vision of data-driven communication networks, and in particular by the possibility to *learn from past executions of network algorithms*. We observe that network algorithms such as traffic engineering algorithms, admission control algorithms, etc. naturally generate a wealth of data which is potentially very useful for future executions of the algorithms but which remained largely untapped so far. Indeed, one of the key challenges of data-driven communication networks is to obtain useful data for implementing the self-adjusting resource allocation and communication algorithms: the input-output pairs computed by algorithms in the past are likely to be one accessible information source.

We have recently conducted a first study of how to extend network algorithms by machine learning [2], showing the potential of such an approach for optimizing *static allocations*. Our results indicate that the "big data" produced by an algorithm during past executions can be leveraged effectively to improve and speed up future, similar solutions, by reducing the algorithm's search space.

With this paper, we take our endeavour a critical step further: we study the feasibility to learn from *dynamic algorithms*, subject to *changing traffic*, a relevant extension in practice [3], [4]. In machine learning terminology, this means that we investigate whether it is possible to learn from feature vectors describing traffic distributions over time. Besides moving from static to dynamic scenarios, we, for the first time, study the possibility of making predictions of global, network-wide allocations; prior work was limited to optimizing resources on a per node-by-node basis [2]. In other words, we consider a *multi-label* learning problem.

As a case study, we consider an archetypical k-median

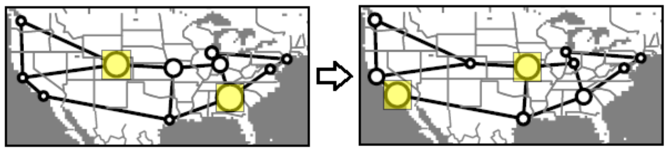


Fig. 1. An example showing dynamic traffic in the network Abilene. The size of each node represents the traffic intensity expected for a time period. Two k-means, e.g., SDN controllers, shown as yellow squares, are placed accordingly to minimize weighted average latency. Since different traffic distributions require different controller locations, the controller placement needs to adapt when the requests change.

problem, which we henceforth regard as a *Weighted Controller Placement Problem (WCPP)*: the k-median resp. controller locations need to be adapted over time to account for changing and different traffic distributions. Fig. 1 illustrates an example of such adaptation.

In **summary**, this paper initiates the study of data-driven network optimizations in dynamic networks (based on algorithm data) subject to time-varying traffic patterns and demands. In particular, we demonstrate that machine learning can indeed be used to learn from past solutions, in a case study revolving around k-median resp. controller placement. Our approach is to phrase the problem as a multi-label classification problem, where the input is traffic intensity at each node in the substrate, and the output is a set of node labels representing the controller locations. We consider different real-world networks with varying traffic distributions. The predicted placement can be used for different purposes: we can use the prediction either (1) directly as a solution, or (2) as an initial solution for a heuristic algorithm like local search. Our proposed system architecture is illustrated in Fig. 2.

II. A Case Study

We consider a k-median problem in SDN, namely the weighted controller placement problem (WCPP). This problem has been studied intensively by the networking community in the past [5], [6], [7], [8]. We first introduce the representation of both network topology and placement request, i.e. traffic distribution. Then the mathematical formulation of WCPP is outlined, followed by the introduction of greedy and heuristic algorithm.

WCPP: Problem Formulation. The underlying network topology is an undirected graph $G := (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ describes the set of nodes in the topology and $\mathcal{E}$ describes the set of edges that inter-connect the nodes, i.e. $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$. $\mathcal{C}$ denotes the set of all possible controller locations. We assume that each node can serve either one or no controller, therefore set $\mathcal{C}$ and set $\mathcal{N}$ are identical. The total number of controllers that need to be placed is $\boldsymbol{k}$. The forwarding shortest-path latency between two nodes n_1 and n_2 is represented as a mapping $\mathcal{L}(n_1, n_2) \rightarrow \mathbb{R}^+, n_1, n_2 \in \mathcal{N}$ [1].

[1]In this paper, we use $\mathbb{R}^+$ to denote all positive real numbers and zero, i.e. $\mathbb{R}^+ := [0, +\infty)$.

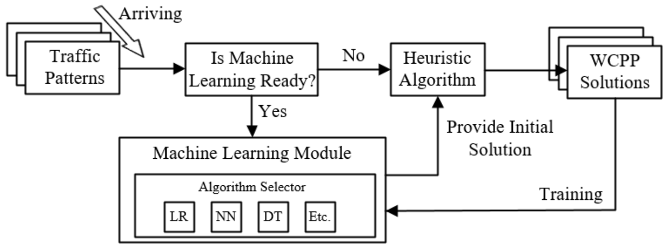


Fig. 2. Traffic patterns change over time. The heuristic algorithm module first generates a number of solutions, which will be used to train the machine learning module. For different available machine learning algorithms, the one which best fits the current setup will be selected. After training, the machine learning module processes new traffic patterns and creates initial solutions, which could be further improved by the heuristic algorithm module.

Traffic intensities of all switches constitute a traffic distribution/pattern, which is defined as a mapping $\mathcal{R}(n) \rightarrow \mathbb{R}^+, n \in \mathcal{N}$. Because of the dynamics in the network, e.g. changes of user locations, traffic distribution alters over time. Given a distribution $\mathcal{R}(\cdot)$, the controllers should be placed in order to minimize the weighted average control latency of all switches. We formulate WCPP as an integer programming problem:

$$\min \sum_{n \in \mathcal{N}, c \in \mathcal{C}} \mathcal{R}(n) \cdot \mathcal{L}(n, c) \cdot a_{n,c} \tag{1}$$

subject to:

$$\sum_{c \in \mathcal{C}} p_c = \boldsymbol{k} \tag{2a}$$

$$\sum_{c \in \mathcal{C}} a_{n,c} = 1, \forall n \in \mathcal{N} \tag{2b}$$

$$\sum_{n \in \mathcal{N}} a_{n,c} \leq |\mathcal{N}| \cdot p_c, \forall c \in \mathcal{C} \tag{2c}$$

$$p_c, a_{n,c} \in \{0, 1\}, \forall c \in \mathcal{C}, n \in \mathcal{N} \tag{2d}$$

Binary variable p_c indicates whether a controller is placed on node $c \in \mathcal{C}$ and binary variable $a_{n,c}$ indicates whether node $n \in \mathcal{N}$ is assigned to controller which locates on node $c \in \mathcal{C}$. Constraint (2a) forces $\boldsymbol{k}$ controllers to be placed. Constraint (2b) assigns a switch to only one controller and (2c) ensures that a switch is assigned to a controller, only if the controller has been placed on a certain node.

Greedy and Heuristic Algorithm. Existing solvers can solve the WCPP optimally, however, they will suffer from exponentially growing runtime in the worst case. Greedy algorithm provides a simple alternative to tackle the optimization problem. Alg. 2 incrementally puts controllers into a set until the number $\boldsymbol{k}$ is reached. Each added controller always gives rise to a minimized objective function. Alg. 1 describes the process of calculating the objective function, i.e. weighted average control latency in (1), which serves as a subroutine for other algorithms.

The local search algorithm has been proved to be a promising heuristic for k-median [9], which outperforms the greedy

Algorithm 1 Objective function Calculation $\mathcal{F}()$

```
Require: Latencies between nodes $\mathcal{L}$, traffic load $\mathcal{R}$, set of switches $\mathcal{N}$, controller set $\mathcal{C}$
1: Get switch to controller mapping $\mathcal{A}$
2: obj $\leftarrow 0$
3: for $n \in \mathcal{N}$ do
4:     obj $\leftarrow$ obj $+ \mathcal{R}(n) \cdot \mathcal{L}(n, \mathcal{A}(n))$
5: end for
6: obj $\leftarrow$ obj$/|\mathcal{N}|$
7: return obj
```

Algorithm 2 Greedy Algorithm

```
Require: Number of controllers $\boldsymbol{k}$
1: Controller Set $\mathcal{C} \leftarrow \varnothing$
2: while $|\mathcal{C}| < \boldsymbol{k}$ do
3:     $n \leftarrow \text{argmin}_n(\mathcal{F}(\mathcal{C} \cup \{n\}))$
4:     $\mathcal{C} \leftarrow \mathcal{C} \cup \{n\}$
5: end while
6: return $\mathcal{C}$
```

TABLE I
MATHEMATICAL NOTATIONS

Symbol	Meaning
$\mathbf{x}^{(i)}$	$\mathbf{x}^{(i)} \in \mathbb{R}^n$, ith feature vector, in our work $n = \mid \mathcal{N} \mid$.
$\mathcal{I}$	$\mathcal{I} := \{l_1, l_2, \ldots, l_{\mid\mathcal{N}\mid}\}$, set of possible labels.
$\widetilde{\mathcal{I}}$	$\widetilde{\mathcal{I}} \subseteq \mathcal{I}$, one specific labeling.
$\widetilde{\mathcal{I}}^{(i)}$	$\widetilde{\mathcal{I}}^{(i)} \subseteq \mathcal{I}$, set of labels associated with vector $\mathbf{x}^{(i)}$.
$\mathbf{z}^{(i)}$	$\mathbf{z}^{(i)} \in \{0,1\}^{\mid\mathcal{I}\mid}$, binary indicator vector associated with $\mathbf{x}^{(i)}$. Component $\mathbf{z}_j^{(i)}$ is one, if and only if label $l_j \in \widetilde{\mathcal{I}}^{(i)}$. Vector $\mathbf{z}^{(i)}$ represents the "true" placement.
h	$h : \mathbb{R}^n \to \{0,1\}^{\mid\mathcal{I}\mid}$, hypothesis of a classifier.
$\mathbf{y}^{(i)}$	$\mathbf{y}^{(i)} \in \{0,1\}^{\mid\mathcal{I}\mid}$, prediction of a classifier, i.e., $\mathbf{y}^{(i)} = h\left(\mathbf{x}^{(i)}\right)$.

algorithm. We introduce Alg. 3 based on local search as a heuristic for WCPP. During initialization, it randomly generates a controller set and calculates the corresponding objective. Line 5 to Line 18 is repeated until no further dominant solutions can be found. Each time we perform $|\mathcal{C}| \cdot (|\mathcal{N}| - |\mathcal{C}|)$ local searches, from which the best one will be compared with the incumbent. A local search move represents swapping a node in set $\mathcal{C}$ with a node in set $\mathcal{N} \setminus \mathcal{C}$.

III. Our Approach

We use multi-label classification [10] to predict controller locations. In multi-label classification, one observation represented by the feature vector $\mathbf{x}^{(i)}$ is associated with a *set* $\widetilde{\mathcal{I}}^{(i)} \subseteq \mathcal{I}$ of labels. Different observations can be associated with different sets of labels. See Tbl. I for an interpretation of the symbols used in this paper. Multi-label classification differs from multi-class classification, as in multi-class classification, exactly *one* label $l^{(i)} \in \mathcal{I}$ is associated with one observation

Algorithm 3 Local Search Algorithm

```
Require: Number of controllers $\boldsymbol{k}$
1: Random generate controller set $\mathcal{C}$, s.t. $|\mathcal{C}| = \boldsymbol{k}$
2: Non-controller node set $\overline{\mathcal{C}} \leftarrow \mathcal{N} \setminus \mathcal{C}$
3: obj$^* \leftarrow \mathcal{F}(\mathcal{C})$, stopflag $\leftarrow$ False
4: repeat
5:     obj$^{**} \leftarrow \infty$, $\mathcal{C}^{**} \leftarrow \varnothing$
6:     for $c \in \mathcal{C}$ do
7:         for $\overline{c} \in \overline{\mathcal{C}}$ do
8:             $\mathcal{C} \leftarrow (\mathcal{C} - \{c\}) \cup \{\overline{c}\}$
9:             if $\mathcal{F}(\mathcal{C}) <$ obj$^{**}$ then
10:                obj$^{**} \leftarrow \mathcal{F}(\mathcal{C})$, $\mathcal{C}^{**} \leftarrow \mathcal{C}$
11:            end if
12:        end for
13:    end for
14:    if obj$^* >$ obj$^{**}$ then
15:        $\mathcal{C} \leftarrow \mathcal{C}^{**}$
16:    else
17:        stopflag $\leftarrow$ True
18:    end if
19: until stopflag
20: return $\mathcal{C}$
```

$\mathbf{x}^{(i)}$. Multi-label classification is used, for example, in computer vision for sentiment analysis, where different images are associated with possibly different sentiments [11].

WCPP as Multi-label classification task. We use each traffic distribution $\mathcal{R}^{(i)}$ as a feature vector $\mathbf{x}^{(i)}$, i.e., $\mathbf{x}_j^{(i)} = \mathcal{R}^{(i)}(n_j)$. Similarly, we use the substrate node identifiers as labels; the ith label $l_i \in \mathcal{I}$ identifies the ith substrate node $n_i \in \mathcal{N}$. The controller placement for the traffic distribution $\mathcal{R}^{(i)}$ is represented by the labeling $\widetilde{\mathcal{I}}^{(i)}$, identifying the substrate nodes on which controllers are placed. Labels for different traffic distributions might change, yet classifiers generally do not have the capabilities to output varying sets. Therefore, we represent the labeling by a binary vector $\mathbf{z}^{(i)}$, where the jth-component $\mathbf{z}_j^{(i)}$ is set to one, if and only if $l_j \in \widetilde{\mathcal{I}}^{(i)}$, i.e., a controller is placed on node n_j. Similarly, the prediction of a classifier is represented by a binary vector $\mathbf{y}^{(i)}$, which is calculated based on $\mathbf{x}^{(i)}$, i.e., $\mathbf{y}^{(i)} = h\left(\mathbf{x}^{(i)}\right)$, where $h(\cdot)$ represents a classifier.

Multi-label Classifiers. Algorithms for multi-label classification can be grouped into two categories: problem transformation methods and algorithm adaptation methods [12], [13]. Problem transformation methods convert multi-label classification tasks into other well-studied learning tasks such as multi-class classification. Algorithm adaptation methods are directly applicable to multi-label classification tasks [13]. We consider three different classifiers: decision tree (DT) CART, neural network (NN) and logistic regression (LR).

Decision Tree. DT can be used for multi-label classification by employing the *binary relevance* approach [13]. DT in conjunction with binary relevance belong to the class of

TABLE II
SETTINGS FOR MACHINE LEARNING ALG.

Model	Parameter	Values
DT	All	Default Parameters
NN	Loss function	Bernoulli Cross Entropy
	# hidden layers	1
	hidden layer size	$\|\mathcal{N}\|$
	output layer size	$\|\mathcal{N}\|$
	Activation function (of hidden layer)	Sigmoid
	Activation function (of output layer)	Sigmoid
	Optimizer	ADAM [14] with step rate of 0.01 and regularization factor of 10^{-5}
LR	Loss function	Bernoulli Cross Entropy

problem transformation methods. One tree for each possible label $l \in \mathcal{I}$ is built and trained independently from all others. A prediction $\mathbf{y}$ is then a binary vector, where the i^{th} component is one if the respective tree detects the corresponding label. In theory, it is possible that more than $\boldsymbol{k}$ components in $\mathbf{y}$ are set to one. In this case, we randomly select $\boldsymbol{k}$ of the entries.

Logistic Regression. Similar to DT, LR predicts whether a controller is placed on each node. Contrary to DT, LR outputs for each node a value between zero and one, which can be interpreted as the *probability* of a controller to be located on that node. We then take the $\boldsymbol{k}$ labels with highest probability as controller locations. To achieve this, the weighted sum of all input values is passed through a sigmoid function. For each label, a distinct set of weights for the weighted sum is used. The weights are jointly updated using the gradient of a loss function, which is in our case the Bernoulli cross entropy loss. Since the weights are jointly updated, this approach does *not* amount to binary relevance, and is able to exploit correlations between single labels in labelings [13].

Neural Network. NN and LR are similar in performing multi-label classification, and they both belong to the class of algorithm adaptation methods. NN can be viewed as LR, where the input undergoes a sequence of nonlinear transformations, as the input is propagated through the network. The output of the last hidden layer is then used as input to LR. NN is thus also able to exploit label correlations, and additionally nonlinear dependencies in the inputs [13].

IV. Evaluation

We evaluate all algorithms with Python on Ubuntu Server 14.04.1 LTS with 16 CPUs and 32 GB memory. Trainings are performed only with CPU and not with GPU support. We assess 6 topologies with different sizes from Topology Zoo [15]: AttMpls (25), Bics (33), Cernet (40), Uninett2010 (74), Deltacom (99) and Cogentco (180). For each topology, we test different $\boldsymbol{k}$s ranging from 5 to 20, with an interval of 5. The traffic load on each node follows a uniform distribution $U(1, 100)$. NN-LS represents the local search algorithm (Alg. 3) with neural network prediction and GDY denotes the greedy algorithm (Alg. 2).

A. Algorithms Comparison

We first evaluate the performance of three machine learning algorithms, with local search as a baseline. Tbl. II gives a summary of the settings for different machine learning algorithms. Local search generates data samples by solving 7 000 problem instances for each substrate. 6 500 samples are used for training the algorithms and 500 samples are used for evaluation. Parameter tuning for the neural network was performed on a separate dataset, with 6 500 samples for training and 500 samples for validation. Samples were obtained on the Bics substrate with $\boldsymbol{k}$ fixed to 3. We use hamming loss between ground truth and classifier prediction to measure predictive accuracy [13]:

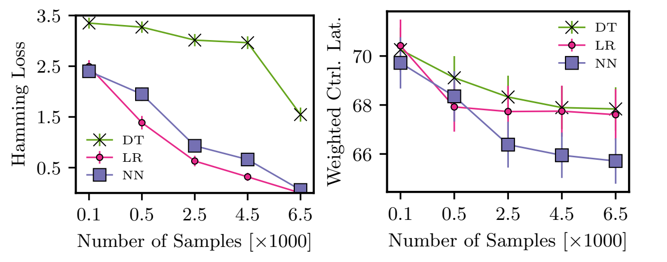
$$d_H(\mathbf{z}, \mathbf{y}) := \frac{1}{N} \sum_{j=1}^{|\mathcal{I}|} \operatorname{xor}(\mathbf{z}_j, \mathbf{y}_i), \tag{3}$$

where N is the number of samples. Hamming loss describes how many controllers are inaccurately predicted.

How much data is needed? Fig. 3 shows the hamming loss on the test set, as well as the weighted average control latency as a function of the training set size for the Bics substrate. For the weighted average control latency in Fig. 3b, the means with 95 % confidence intervals are shown. Fig. 3 clearly shows that the hamming loss and the objective function both decrease, as the number of training samples increases. In contrast to the hamming loss, the decrease of the objective function levels off starting at a training set size of 2 500. Interestingly, LR outperforms NN in terms of the hamming loss in Fig. 3a, but NN outperforms LR in terms of the objective function, visible in Fig. 3b. Also, the difference in the prediction accuracy in Fig. 3a between DT and LR does not translate to the objective function in Fig. 3b. DT and LR yield nearly the same objective starting from 4 500 samples. These observations indicate that hamming loss cannot fully represent our optimization objective.

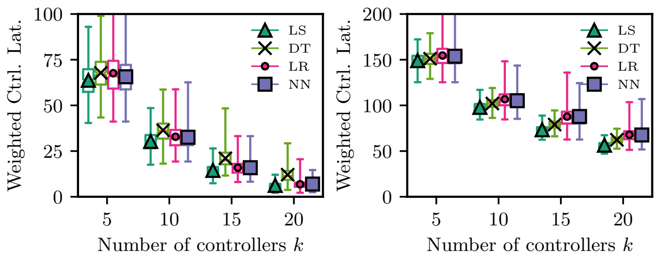
Impact of Topologies. Fig. 4 provides the performance comparison of the three machine learning algorithms for Bics and Cogentco and all $\boldsymbol{k}$s, where LS serves as a baseline. Intuitively, larger $\boldsymbol{k}$ leads to smaller objective for all setups. For Bics as well as Cogentco, NN and LS give competitive results. DT is outperformed by NN and LS on Bics, however, DT outperforms NN and LS on Cogentco in Fig. 4b. Especially the variability of the weighted average control latency obtained with the solutions of DT on Cogentco is small, in fact almost as small as the variability of LS. The respective differences of DT compared to LS and NN become more pronounced as $\boldsymbol{k}$ increases.

As a conclusion, an exact prediction of controller placement is still hard to achieve. Besides, an algorithm that wins in all different scenarios does not exist. Due to space constraints, we report only on the performance of NN in the remainder



(a) Prediction accuracy (b) Objective function

Fig. 3. The prediction evaluation with different training set sizes for topology Bics ($\boldsymbol{k} = 5$).



(a) Bics (b) Cogentco

Fig. 4. The comparison of different machine learning algorithms for different topologies.

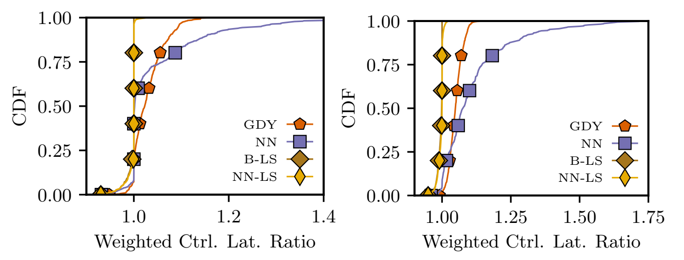
of this section. We decide for NN, as it shows comparable performance to DT and LS, but outperforms DT in terms of the hamming loss and is slightly better than LS in terms of objective.

B. Enhancing Heuristic Algorithms

We now evaluate the benefit of heuristic algorithms from machine learning on the example of local search. Since the performance of LS depends on the initial solution, we run LS for 10 times, each with a unique random initial solution, take the one that performs best. The straw-man case (B-LS) should provide us with insights on the problem complexity of our use case: whether we can already achieve, with a small initial solution sub-sampling, a significant benefit.

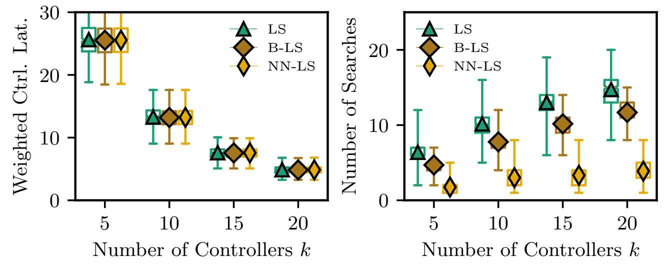
Fig. 5 compares two particular problem setups. We quantify the performance gain of different algorithms in comparison to LS, i.e. $\text{obj}_{\text{alg}}/\text{obj}_{\text{LS}}$ for each sample. The cumulative distributive function is then plotted of all 500 samples in the test set. In general, the performance of NN-LS is on a par with that of LS, indicating that neural network provides good initial solutions for local search. For B-LS, we conclude that the objective cannot be significantly improved with different random initial solutions. Notably different setups provide distinct comparisons for NN and GDY. NN is better than GDY in Fig. 5a, whereas beaten by GDY in Fig. 5b.

In Fig. 6, we evaluate the performance of LS, B-LS and NN-LS in terms of the objective and the runtime. We variate $\boldsymbol{k}$ for Uninett2010. Interestingly, B-LS does not necessarily provide a solution with dominant objective. The runtime of LS depends



(a) Uninett2010, $\boldsymbol{k} = 5$ (b) Deltacom, $\boldsymbol{k} = 10$

Fig. 5. The comparison of different algorithms for two particular setups.



(a) Objective funtion (b) Runtime

Fig. 6. The comparison of two advanced algorithms, i.e. B-LS and NN-LS, in terms of objective and runtime for topology Uninett2010.

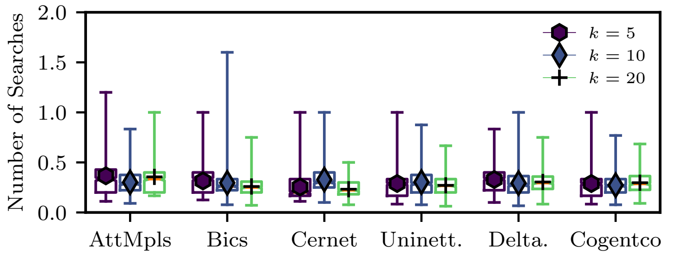


Fig. 7. The comparison of all evaluated topologies in terms of runtime saving.

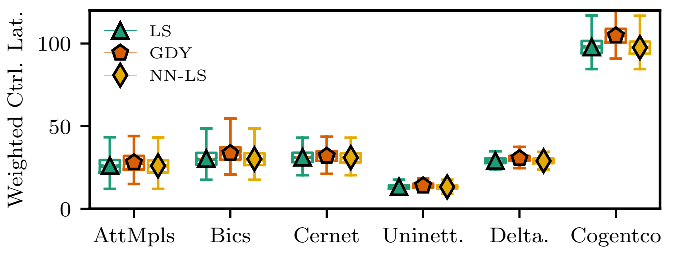


Fig. 8. The comparison of all evaluated topologies in terms of objective and $\boldsymbol{k} = 10$.

on the number of searches. One search indicates one move towards an improved objective, i.e. Line 5 to Line 18 in Alg. 3. The highlight is in Fig. 6b, where NN-LS can save as much as half the number of searches compared with LS for most samples. Provided good initial random solution as in B-LS, we could expect less runtime. Nevertheless, the runtime saving of NN-LS is still dominant. We conclude that for the use case of WCPP, initial solution subsampling provides mostly non-dominant solutions in terms of the objective and has no advantage in runtime saving.

Lastly, we compare the performance of all evaluated topologies. We observe in Fig. 8 that NN-LS provides the same achieved objective as LS, for all topologies with different sizes and $\boldsymbol{k} = 10$. Fig. 7 gives a normalized comparison of runtime saving. For each sample, the number of searches in NN-LS is divided by that in LS. For all topologies NN-LS takes only one third the number of searches for most samples, which is independent of $\boldsymbol{k}$. In conclusion, for a heuristic algorithm that starts with an initial feasible solution, applying machine learning prediction can result in solutions of decent quality and reduce runtime significantly.

V. Conclusion And Future Work

We performed a case study of data-driven communication network optimization. Our approach shows promising first results: neural network prediction as initial solution for heuristic could save considerable amount of algorithm runtime. It also opens several interesting directions for future research. For example, it would be interesting to extend our framework to account for additional features, such as network connectivity. Moreover, it will be interesting to consider additional use cases for dynamic and algorithmic data-driven optimizations.

Acknowledgment

This work has been performed in part in the framework of the CELTIC EUREKA project SENDATE-PLANETS (Project ID C2015/3-1) and is partly funded by the German BMBF (Project ID 16KIS0473), and in part in the framework of the EU project FlexNets funded by the European Research Council under the European Unions Horizon 2020 research and innovation program (grant agreement No 647158 - FlexNets). Stefan Schmid was supported by Aalborg University's PreLytics project. The authors alone are responsible for the content of the paper.

References

[1] W. Kellerer, A. Basta, and A. Blenk, "Using a flexibility measure for network design space analysis of sdn and nfv," in *Computer Communications Workshops (INFOCOM WKSHPS), 2016 IEEE Conference on*, pp. 423–428, IEEE, 2016.

[2] A. Blenk, P. Kalmbach, W. Kellerer, and S. Schmid, "o'zapft is: Tap your network algorithm's big data!," in *Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, pp. 19–24, ACM, 2017.

[3] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pp. 267–280, ACM, 2010.

[4] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 92–99, 2010.

[5] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 7–12, ACM, 2012.

[6] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann, "Heuristic approaches to the controller placement problem in large scale sdn networks," *IEEE Tran. on Netw. and Serv. Mgmt.*, vol. 12, no. 1, pp. 4–17, 2015.

[7] G. Yao, J. Bi, Y. Li, and L. Guo, "On the capacitated controller placement problem in software defined networks," *IEEE Communications Letters*, vol. 18, no. 8, pp. 1339–1342, 2014.

[8] M. He, A. Basta, A. Blenk, and W. Kellerer, "Modeling flow setup time for controller placement in sdn: Evaluation for dynamic flows," in *Proc. of IEEE ICC 2017*, IEEE, 2017.

[9] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit, "Local search heuristics for k-median and facility location problems," *SIAM Journal on computing*, vol. 33, no. 3, pp. 544–562, 2004.

[10] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *International Journal of Data Warehousing and Mining*, vol. 3, no. 3, 2006.

[11] Z.-J. Zha, X.-S. Hua, T. Mei, J. Wang, G.-J. Qi, and Z. Wang, "Joint multi-label multi-instance learning for image classification," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8, IEEE, 2008.

[12] A. de Carvalho and A. Freitas, "A tutorial on multi-label classification techniques," *Foundations of Computational Intelligence Volume 5*, pp. 177–195, 2009.

[13] M.-L. Zhang and Z.-H. Zhou, "A review on multi-label learning algorithms," *IEEE transactions on knowledge and data engineering*, vol. 26, no. 8, pp. 1819–1837, 2014.

[14] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[15] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.