o'zapft is: Tap Your Network Algorithm's Big Data!

Andreas Blenk, Patrick Kalmbach, Wolfgang Kellerer
Technical University of Munich, Germany

Stefan Schmid
Aalborg University, Denmark

ABSTRACT

At the heart of many computer network planning, deployment, and operational tasks lie hard algorithmic problems. Accordingly, over the last decades, we have witnessed a continuous pursuit for ever more accurate and faster algorithms. We propose an approach to design network algorithms which is radically different from most existing algorithms. Our approach is motivated by the observation that most existing algorithms to solve a given hard computer networking problem overlook a simple yet very powerful optimization opportunity in practice: many network algorithms are executed repeatedly (e.g., for each virtual network request or in reaction to user mobility), and hence with each execution, generate interesting data: (problem,solution)-pairs. We make the case for leveraging the potentially big data of an algorithm's past executions to improve and speed up future, similar solutions, by reducing the algorithm's search space. We study the applicability of machine learning to network algorithm design, identify challenges and discuss limitations. We empirically demonstrate the potential of machine learning network algorithms in two case studies, namely the embedding of virtual networks (a packing optimization problem) and k-center facility location (a covering optimization problem), using a prototype implementation.

CCS CONCEPTS

• **Networks → Network algorithms**;

KEYWORDS

Algorithms, Computer Networks, Machine Learning, Big Data

ACM Reference format:
Andreas Blenk, Patrick Kalmbach, Wolfgang Kellerer and Stefan Schmid. 2017. *o'zapft is:* Tap Your Network Algorithm's Big Data!. In *Proceedings of Big-DAMA '17, Los Angeles, CA, USA, August 21, 2017,* 6 pages.
https://doi.org/10.1145/3098593.3098597

1 INTRODUCTION

The Context: Network Algorithms. Algorithms play an important role in essentially any aspect of computer networking: from the design of new network topologies, the deployment of network functions (e.g., middleboxes), to the resource efficient operation. With the increasing virtualization and programmability of networked systems and the resulting more flexible resource allocation, also over time, the importance of network algorithms is likely to increase further in the future.

The Problem: Computational Hardness. The design of such algorithms, however, can be challenging as the underlying problems are often *computationally hard*, and at the same time, need to be *solved fast*. For example, traffic engineering or admission control problems, or the problem of embedding entire virtual networks, underly hard unsplittable network flow allocation problems; or the problem of placing a virtual network function, can be seen as k-center clustering and facility location problem; etc.

The Limitation: Fire-and-Forget. Researchers over the last decades have continuously improved the approximation quality and runtime of network algorithms, gaining deep insights into the underlying structures. Yet, in practice, many network algorithms today lead a fairly boring life: they are frequently invoked to solve some problem instances which look similar to hundreds of problems (the algorithms' big data) solved before, but are forced to compute each time a new solution *from scratch*. Our hypothesis in this paper is that employing algorithms in a fire-and-forget manner is not only boring, but can also be highly inefficient.

The Opportunity: Tap into Your Algorithm's Big Data. This paper is motivated by the observation that network algorithms generate a wealth of big data (e.g., in form of problem instance and solution pairs), which introduce an unexplored optimization opportunity. We hence promote a radically different approach to designing network algorithms, and ask: is it possible to learn an algorithm's behavior and even improve on the algorithm's solutions, by studying solutions generated by a given algorithm to similar problems in the past? In particular, we propose a Machine Learning (ML) approach, based on *supervised learning*, to tap into a network algorithm's knowledge base.

Contributions. This paper makes the case for a machine learning approach to network algorithm design. We investigate the feasibility of learning from previous solutions to similar problems to improve and speed up the solution process in the future: e.g., by predicting upper and lower bounds on solution values (either costs or benefits); by facilitating an effective pruning of the search space (e.g., of a mixed integer program); by predicting initial solutions, facilitating an efficient local search; by optimizing the parameters of a given algorithm; or by optimizing the selection of the to-be-used algorithm in the first place; etc. Our optimization data is publicly available for reproducibility at [11].

We identify challenges of such an approach, discuss the design space and optimization opportunities, and point out limitations. In particular, we describe a general framework, called *o'zapft is*[1], and empirically show its potential in two fundamental case studies,

[1]Inspired by the Bavarian expression for "[the barrell] has been *tapped*" (cf. Fig. 1), used to proclaim a famous festival in October.

Big-DAMA '17, August 21, 2017, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5054-9/17/08…$15.00
https://doi.org/10.1145/3098593.3098597

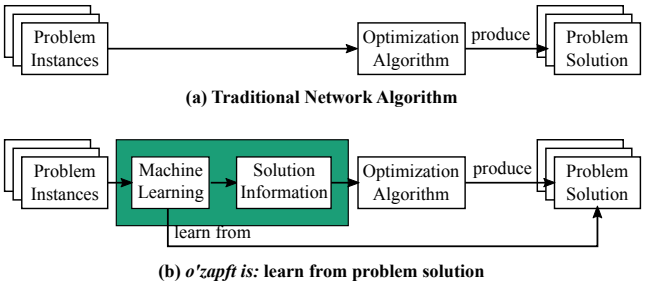


Figure 1: Traditional network algorithm vs. *o'zapft is*. The input to both systems are *Problem Instances*. Traditionally, an *Optimization Algorithm* provides, if possible, a problem solution for each problem instance independently. *o'zapft is* leverages *Machine Learning* to learn from prior problem solutions, to compute additional *Solution Information* (e.g., reduced search space, upper and lower bounds, initial feasible solutions, good parameters, etc.) and hence to optimize the network algorithm.

the first one representing a packing optimization problem and the second one representing a covering optimization problem:

(1) *Virtual network embeddings:* The embedding of virtual nodes and routes between them is an archetypal problem underlying many resource allocation and traffic engineering problems in computer networks. The common theme of these problems is that network resource consumption should be minimized, while respecting capacity constraints. As a first step, we investigate whether machine learning can be used to *predict the embedding costs* of virtual networks. Our results reveal that embedding costs can be predicted well, in particular for heuristic or greedy algorithms.

(2) *k-center problems ("facility location"):* Clustering and facility location problems arise in various flavors and forms in the optimization of computer networks. For example, the placement of middleboxes, virtualized network functions, CDN caches, or SDN controllers can be modeled as variants of k-placement problems. The common theme of these problems is that certain functionality should be placed "close" to the nodes using it. As a first step, we investigate whether ML can be used to *predict the optimal placement* of such functionality. Our results show that ML cannot exactly predict the optimal solutions, but provides valid means to reduce the search space to speed up optimization.

Organization. The remainder of this paper is organized as follows. We present our vision, identify challenges and present our approach in Section 2. We examine the virtual network embedding case study in Section 3 and the facility location one in Section 4. We report on related work in Section 5 and conclude in Section 6.

2 METHODOLOGY AND APPROACH

We envision history-aware network algorithms, which learn from the outcomes of state-of-the-art one-shot algorithms, see Figure 1: For a given network problem, the target of our system *o'zapft is* is to learn from problem solutions that were obtained earlier by an optimization algorithm, in order to improve the future execution of new problem instances. When faced with new problem instances, the optimization algorithm can make use of solution information that is provided by machine learning models.

In the following, we will focus on network problems which can be expressed in terms of *graphs*. Graphs are used to describe physical network topologies, but also traffic demands, routing requests and virtual networks (e.g., VPNs, virtual clusters, etc.) can be described in terms of graphs. In order to automatically learn solutions to network algorithms, four challenges have to be addressed:

Common patterns (C1): A model from one set of network optimization problems should be generalizable and applicable also to other, similar networking problems: e.g., solutions from smaller networks should be useful to predict solutions for larger networks as well. For example, routing problems could be classified based on requests between similar areas (e.g., subnets) of the network, or reflect distance measures.

Compact representations (C2): It is often infeasible to store the entire history of problem solutions (e.g., the costs and embeddings of all past virtual networks) in a learning framework explicitly. An efficient way to store and represent problem solutions could rely on probabilistic vectors, e.g., capturing likelihoods of nodes visited along a route computed by a routing algorithm.

Data skewness and bias (C3): Making good predictions for highly biased problems (e.g., containing 1% yes-instances for which feasible virtual network embeddings actually exist and 99% no-instances) is trivial: simply always output *no*. Clearly, such solutions are useless and mechanisms are needed which account for skewness and bias in the data and solution space. Note that the problem arises both when quantifying the performance and also when training the models.

Data training/test split (C4): In terms of methodology, a good split between training and test set must be found. The training/test set typically cannot be split by randomly taking, e.g., network node samples from the overall data, as each node sample belongs to a unique graph.

In order to address the challenges (C1-C4) identified above, *o'zapft is* builds upon (and tailors to the networking use case) a number of existing concepts.

C1: A common method to classify graphs resp. networks (Challenge C1) is to employ *graph kernels*. However, kernels are expensive to compute [7, 14]. An interesting alternative applied in *o'zapft is* is to use node, edge, and graph *features*, e.g., based on existing network centrality concepts [3], to efficiently classify graphs resp. networks to represent problem-solutions pairs.

C2: We consider fixed length real valued feature vectors instead of storing whole adjacency matrices for solutions, which can potentially even outperform alternative methods such as graph kernels [7] as it has been originally shown in [14]. For the facility location problem, we also introduce the respective node features per substrate node. Furthermore, we additionally use minimum, maximum, average, and standard deviations of all graph/node features.

C3: There exist standard techniques to overcome bias and data skewness, e.g., *misclassification cost assignment*, *under-sampling*, or *over-sampling* [18]. Misclassification costs can be assigned if the real cost, e.g., the impact of a wrong prediction on the overall embedding performance, is known. To not loose valid data as it is the case for

under-sampling, we use over-sampling of the underrepresented instances. Note that sampling can only be applied to training data, as, of course, the true/false values are not known for test data beforehand.

C4: We split the overall data based on entire graphs, and hence avoid, e.g., problems of node-based samples.

In this paper, in order to validate our hypothesis that network algorithms can learn from similar past solutions, we will consider the following methodology.

Regressors and Classifiers Models. For our first case study, the virtual network embedding problem, we compare four machine learning regressors: *Linear Regression (LR)*, *Bayesian Ridge Regressor (BRR)*, *Random Forest Regressor (RF)*, and *Support Vector Regression (SVR)*. The regressors predict the cost of to-be-embedded virtual networks. For our second case study, facility location, we apply four classifiers: *Logistic Regression Classifier (LRC)*, *Extra Trees Classifier (ETC)*, *AdaBoost Classifier (ABC)*, and *Support Vector Classifier (SVC)*. For a given substrate network, the classifiers calculate the probability whether a node will host a facility. For all models, we apply a *grid search* on the parameters to find the best performance on the training set.

Data Processing. The data is split into training, validation and test data sets. A model is fitted with a parameter set on the training data set. The best models of the different regressors are chosen with respect to their performance on the validation data set. The best of all models, i.e., among the different model types, is finally chosen with respect to the performance on the test data set.

Machine Learning Metrics. To provide a more general comparison of models for different substrate networks and Virtual Network Embedding (VNE) algorithms, we report on the coefficient of determination metric R^2, the goodness of fit of a machine learning model, and Root Mean Squared Error (RMSE). For R^2, a value of 1 indicates a precise regression approximation and negative values (up to $-\infty$) indicate a bad performance; 0 means that the model is as good as predicting the average value. Moreover, to quantify the performance of classifiers, we will often consider the F1 score: a combination of *precision* and *recall*; both are useful when working with skewed data.

3 CASE STUDY I: VNE COSTS

In a first case study, we analyze the potential of *o'zapft is* to automatically learn and predict the embedding cost of a virtual network request. The inputs to the Virtual Network Embedding (VNE) [5] problem are virtual network requests (VNRs) consisting of nodes and edges, i.e., requests are represented as graphs. These graphs need to be placed on a substrate network, which is also represented as a graph. The virtual network requires node and link resources, while the capacities of the substrate nodes and links are limited. The embedding cost is given by the total length of the virtual links interconnecting the requested virtual nodes. While embedding costs are a main concern in any virtual network embedding problem, additional objectives are considered, such as min max load [17]. The problem is NP-hard in general.

Let us first introduce our experimental methodology for the virtual network embedding.

Analyzed VNE Process & Data. We consider five frequently used substrate networks to evaluate virtual network embedding algorithms: two synthetic topologies (two random graphs: Erdős-Rényi (ER) and Barabási-Albert (BA)) and three real topologies (a Topology Zoo graph, and two datacenter topologies). From the Topology Zoo [13], we choose the KDL topology: This network consists of 734 nodes and serves as an example of a large scale ip-layer access network deployed in Kentucky, USA. To study datacenter networks, we consider a 6-ary Fat-tree (DC-FT) and a BCube_2 with 4^2-port switches (DC-BC).

Analyzed VNE Algorithms. We compare two heuristics, *Greedy* [20] and *GRC* [8], and one cost optimal algorithm (*SDP*), which is implemented as Mixed Integer Programs (MIPs) [17] in our existing framework [4]. The objective of SDP is to minimize the embedding cost. These algorithms are frequently studied in the literature and hence serve as good initial representatives to demonstrate the feasibility of *o'zapft is*. Since the underlying problem is NP-hard and SDP slow in large network topologies (> 50 substrate nodes), we prematurely interrupted computations after timeouts of 30, 60, and 90 seconds.

Strawman. For comparison, we implemented a Strawman approach (SM). SM is based on the intuition that the number of VNR nodes and links are mostly related to the cost. As equal-sized VNR are not always embedded similarly, we use the average among all VNRs of the training data. To predict the embedding cost, SM takes the number of virtual (requested) links and virtual nodes as an input and calculates the cost based on a steadily interpolated 2D surface, which has been fitted to the training data.

Instances. For each setup, i.e., combination of VNE algorithm and embedding process, at least 5 runs with 2 500 Virtual Network Requests (VNRs) each are generated. For training and testing, only the accepted VNRs are used, as only those provide information on the true embedding cost.

3.1 Tapping Data is Useful

We first investigate how much learning is required to make useful predictions of the virtual network embedding costs. Our experiments show that already after a short training period, virtual network embedding costs can be estimated well. Fig. 2 shows boxplots of the cost deviation over an increasing amount of training samples for all VNE algorithms. We do not depict outliers and the whiskers of the boxplots show the 90^{th} percentile of the test data. Interestingly, while the mean values are quite stable, models typically underestimate the cost of VNRs. The prediction quality of all models increases with more training data. For Greedy, the accuracy of the 90^{th} percentile of the data improves from an underestimation of 40 % to an underestimation of 20 %. GRC performed similarly and is omitted here for space constraints.

Let us next investigate to what extent the performance depends on the machine learning model, the substrate network and the algorithm. To provide a more general comparison of models for different substrate networks and VNE algorithms, we report on the coefficient of determination metric R^2, the goodness of fit of a machine learning model, and Root Mean Squared Error (RMSE) in Tab. 1. In our experiments, we find that the ML models can achieve high R^2 scores between 0.8 and 0.99 across all substrate network

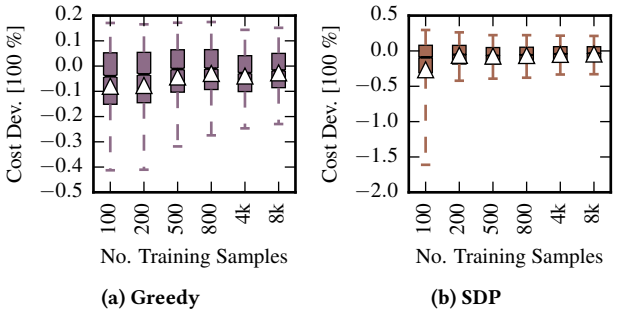


Figure 2: Boxplots of VNE cost prediction deviations with RF model in percentage as a function of training samples. *Estimator model:* RF. Greedy and SDP VNE algorithm. Note the different x-axis and y-axis scales.

VNE Alg.	Mean Perf.	Regression	R^2	RMSE
Greedy	1274.55	BRR	0.989	131.8
		RF	0.990	127.8
		LR	0.989	131.8
		SVR	0.990	125.1
		SM	-0.860	1710.3
GRC	1571.68	BRR	0.974	249.5
		RF	0.979	224.2
		LR	0.974	249.9
		SVR	0.983	203.3
		SM	-0.920	2129.3
SDP	1026.54	BRR	0.893	278.5
		RF	0.907	259.7
		LR	0.892	279.2
		SVR	0.898	272.2
		SM	-0.923	1180.4

Table 1: Regressors' performances for all VNE algorithms. Performance given via R^2 score and *Root Mean Squared Error (RMSE)* for ER substrates.

types, while RF and SVR always achieve the highest values. The Strawman (SM) approach generally shows the worst performance. Note the lower R^2 scores for SDP. The same observations hold for RMSE values, where values are in 30 % range of the mean performance. We find that while *o'zapft is* can predict the performance of heuristic algorithms well, the performance is worse for optimal algorithms: given the high complexity of the Mixed Integer Program solutions, this is not surprising (it is impossible to perfectly and quickly guess an optimal solution to an NP-hard problem). Yet, the solutions are promising, and they show the potential of the approach (and for the design of more advanced machine learning algorithms).

3.2 Speeding-up Computations

o'zapft is comes with interesting flexibilities to speed up our algorithms further: Perhaps a small number of low-complexity features are sufficient to obtain good approximations? First, to study which

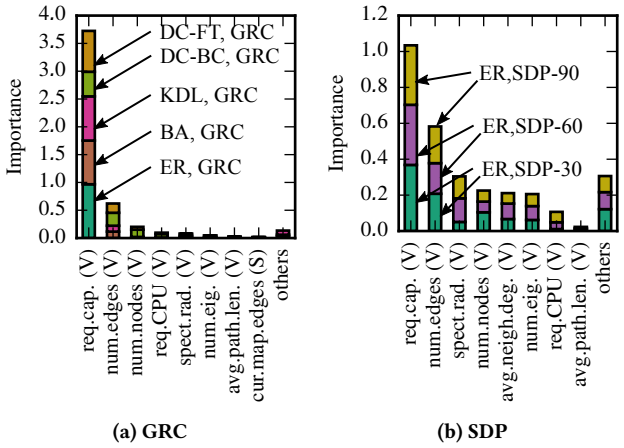


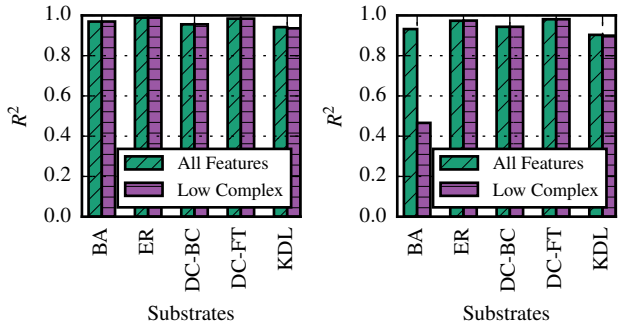
Figure 3: Feature importance of RF models. VNR features are indicated via (V), substrate ones via (S). All remaining importances are accumulated in "others". For GRC, the results are shown for different substrate types. For SDP, the results are shown for the timeouts of 30/60/90 seconds.

features are actually important for the solution learning, let us focus on the Random Forest (RF) model, which allows to investigate feature importance. Fig. 3 shows that the most important feature is the requested link capacity. For GRC, we report on different substrate network types. For the BCube topology, the number of edges and the number of nodes have a significant influence. For SDP, we report on the feature importance for all timeout settings of 30/60/90 seconds. Here, we cannot note a significant difference between the different timeout settings. Compared to the GRC, the feature importance is more distributed across all features. This is to be expected as for optimal algorithms, the search space and hence the variation of solutions, is larger.

So we may ask: is it even sufficient to focus on linear-time graph features to obtain accurate cost predictions? To find out, we trained our regressors with features that have complexity $O(n)$, $O(n+m)$ and $O(n \cdot \log n)$ (see [14]). As Fig. 4 shows for the R^2 score of LR, interestingly, for Greedy (Fig. 4(a)), even the low complex features provide high R^2 scores. In case of GRC, however, the BA substrate prediction is negatively affected by the features choice and the simple models cannot compensate for some of the high complexity features.

4 CASE STUDY II: FACILITY LOCATION

We now consider a second archetypal network optimization problem: facility location. The input to this problem is a network consisting of nodes and edges, a set of demands (in our case all network nodes) which need to be connected to k to-be-allocated facilities. Where in the network to place k facilities and how to connect the demand to them? We will study the *minimum k-center* problem variant, which is NP-hard in general. *We analyze whether machine learning can predict for a substrate node whether it will host a facility.* For this, we compare the results of existing facility location algorithms and our machine learning variants when solving the facility location problem.



(a) Greedy **(b) GRC**

Figure 4: Comparison of R^2 of LR model for *All Features* versus *Low Complex* for all substrate networks for Greedy and GRC embedding algorithms.

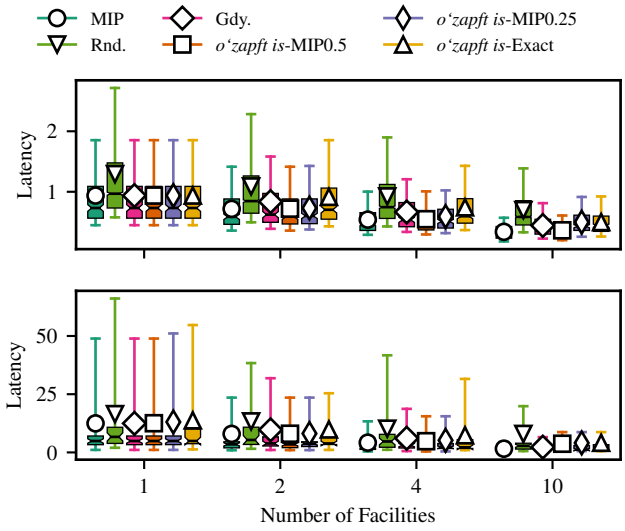


Figure 5: Boxplots comparing the k-center performance of MIP, Rnd., Gdy., *o'zapft is* algorithms. Upper plot: BA. Lower plot: Topology Zoo (TZ).

Facility Location Algorithms. We use three baseline algorithms in our study: an optimal Mixed Integer Programming algorithm (*MIP*), a greedy algorithm (*Greedy*), and a random sampling algorithm (*Random*). Greedy solves the facility location by iteratively taking the next node which maximizes the reduction of the objective function.

Machine Learning-based Algorithms. Given a substrate graph, the training data of the ML-based algorithms is given by the optimal (MIP) solutions. But instead of taking the 0/1 results of our classifiers directly, we use the provided probability of every node to host a facility: the ML algorithms rank the nodes based on the given probabilities. The probability ranking of the nodes is used for two types of ML-based facility location algorithms, an exact algorithm and preselection-based algorithms. The target of the preselection-based algorithms is to reduce the search space. The exact algorithm (*o'zapft is*-Exact) uses exactly the k nodes with the highest probability ranking. A preselection-based algorithm (*o'zapft is*-MIPX) uses a $ratio$ (either 0.25 or 0.5) parameter to determine the number of nodes to select among all substrate nodes. For instance, for a network of 40 substrate nodes and a ratio of 0.25, *o'zapft is* – *MIP*0.25 selects the ten nodes with the highest probabilities. Then for *o'zapft is*-MIPX, the MIP is executed on the preselected subset of nodes to find the best solution.

4.1 Exact is Hard, But Still Useful

Learning optimal placements is a challenging goal, as shown for the latency values in Fig. 5. All results show that MIP achieves the best performance as expected, and Random the worst; however, *o'zapft is*-Exact already comes next to Random among all scenarios. This means that the facility locations cannot be exactly learned and more research is required. Nevertheless, as we will see in the following, *o'zapft is* can be attractive for supporting exact algorithms.

Fig. 5 illustrates that *o'zapft is*-MIP0.5 performs as good as the MIP for a small number of facilities (1-4) and as good as or better than Greedy, while only scanning half of the substrate nodes. This holds for most substrate, facility number, and objective combinations. This also holds for *o'zapft is*–*MIP*0.25 except for 10 facilities. The reason is that for 10 facilities, *o'zapft is* – *MIP*0.25 works like *o'zapft is*-Exact. While this only affects *o'zapft is* – *MIP*0.25 for BA topologies (upper subplot), the lower subplot shows that also *o'zapft is* – *MIP*0.5*'s* performance is worse for 10 facilities. The reason is that the Topology Zoo dataset contains more topologies with less then 40 nodes. For these topologies, even a ratio of 0.5 selects always only 10 nodes. Thus, the algorithms work like *o'zapft is* – *Exact*, which was in general not able to keep pace with Greedy in such cases.

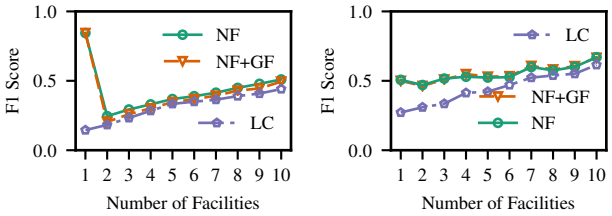
4.2 Faster Placements

Again, we find that graph and complex node features are not important when predicting facility locations for random networks: Fig. 6 compares the F1 score of the true data for three feature settings: low complexity (LC), node (NF), and node+graph features (GF). The results demonstrate the need to differentiate among topology types and sources. While the F1 score for random networks is less affected, the Topology Zoo results show a clearer gap between low complexity features and more advanced graph/node features, where, e.g., betweenness centrality and eigenvector centrality are added.

We conclude that preselection based on ML does not diminish the performance of optimal algorithms: an interesting angle to improve network algorithms' efficiencies.

5 RELATED WORK

Obviously, we are not the first to observe the potential of machine learning in the context of networked and distributed systems optimization. To just name a few examples, Jim Gao [6] showed that a neural network framework can learn from actual operations data to model plant performance and help improving energy consumption in the context of Google data centers.



(a) Barabási-Albert **(b) Topology Zoo**

Figure 6: F1 score comparison for different feature complexities: NF, GF, and LC.

Another emerging application domain for artificial intelligence is the optimization of networking protocols. Many Internet protocols come with several properties and parameters which have not necessarily been optimized rigorously when they were designed, but which offer potential for improvement. To just name one example, Winstein and Balakrishnan [19] have proposed an interesting computer-driven approach to design congestion control protocols as they are used by TCP. The automatic optimization of "protocol knobs" may also be interesting for approaches like Active Queue Management [1], which are in principle attractive but still not widely used today, as their configuration is too complex.

Just recently, the concept of deep reinforcement learning has been applied to resource management of cloud network resources [16]. The proposed system learns to manage resources from experience. In a wider sense, Bello et al. [2] propose Neural Combinatorial Optimization (NCO), a framework that relies on reinforcement learning and neural networks to solve combinatorial optimization problems, as demonstrated for the Traveling Salesman Problem (TSP). In contrast to our approach, these concepts do not rely on labeled data, i.e., operate in an unsupervised fashion.

The potential of learning methods has been demonstrated in many many domains already, beyond distributed systems and networks. A particularly interesting example is the parameterization of Mixed Integer Programming (MIP) solvers, such as CPLEX, GUROBI, and LPSOLVE [9, 10], or the design of branching strategies [12].

Liu et al. [15] propose that data mining can be used to preprocess the search space of high-dimensional problems, hence speeding up the solution of combinatorial optimization and continuous optimization problems.

While our recent work demonstrated the application of our idea to a specific problem [4], we proposed in this paper a generalization of our framework. Thus, we showcased two further use cases; one where we predict the actual placement; and another one where we predict metrics in a multi-objective problem.

6 CONCLUSION

We have shown that a supervised learning approach may be able to predict solutions of network optimization algorithms. Clearly, our approach so far is simple, and we believe that our paper opens many interesting directions for future research. For example, we believe that machine learning optimal solutions, e.g., generated by Mixed Integer Programs, can be interesting in several scenarios and still offers many optimization opportunities. We focused on standard machine learning approaches so far, and it will be interesting to consider more sophisticated approaches, e.g., based on neural networks or using reinforcement learning. At the same time, it will be interesting to better understand the limitations of such approaches.

ACKNOWLEDGMENT

This work is part of a project that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation program (grant agreement No 647158 - FlexNets) as well as from Aalborg University's project *PreLytics*.

REFERENCES

[1] Sanjeewa Athuraliya, Steven H Low, Victor H Li, and Qinghe Yin. 2001. REM: Active queue management. *IEEE Network* 15, 3 (2001), 48–53.
[2] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. 2017. Neural Combinatorial Optimization with Reinforcement Learning. In *ICLR Workshop (2017)*. https://openreview.net/pdf?id=Bk9mxlSFx
[3] Michele Berlingerio, Danai Koutra, Tina Eliassi-Rad, and Christos Faloutsos. 2012. NetSimile: A Scalable Approach to Size-Independent Network Similarity. *CoRR* abs/1209.2684 (2012).
[4] Andreas Blenk, Patrick Kalmbach, Patrick Van Der Smagt, and Wolfgang Kellerer. 2016. Boost Online Virtual Network Embedding : Using Neural Networks for Admission Control. In *Proc. IFIP/IEEE CNSM*.
[5] Andreas Fischer, Juan Felipe Botero, Michael Till Beck, Hermann de Meer, and Xavier Hesselbach. 2013. Virtual Network Embedding: A Survey. *IEEE Communications Surveys & Tutorials* 15, 4 (Jan. 2013), 1888–1906.
[6] Jim Gao. 2014. Machine learning applications for data center optimization. *Google White Paper* (2014).
[7] Thomas Gärtner, Peter Flach, and Stefan Wrobel. 2003. On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines*. Springer, 129–143.
[8] Long Gong, Yonggang Wen, Zuqing Zhu, and Tony Lee. 2014. Toward profit-seeking virtual network embedding algorithm via global resource capacity. In *Proc. IEEE INFOCOM*. 1–9.
[9] Frank Hutter. 2014. Machine Learning for Optimization: Automated Parameter Tuning and Beyond. (2014).
[10] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2010. Automated configuration of mixed integer programming solvers. In *Proc. CPAIOR*. Springer, 186–202.
[11] Patrick Kalmbach, Johannes Zerwas, Michael Manhart, Andreas Blenk, Stefan Schmid, and Wolfgang Kellerer. 2017. Data on "o'zapft is: Tap Your Network Algorithm's Big Data!". (2017). https://doi.org/10.14459/2017md1361589
[12] Elias Boutros Khalil, Pierre Le Bodic, Le Song, George L Nemhauser, and Bistra N Dilkina. 2016. Learning to Branch in Mixed Integer Programming.. In *Proc. AAAI*. 724–731.
[13] Simon Knight, Hung X. Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. 2011. The Internet Topology Zoo. *IEEE J. on Sel. Areas in Communications* 29, 9 (2011).
[14] Geng Li, Murat Semerci, Bülent Yener, and Mohammed J Zaki. 2012. Effective graph classification based on topological and label attributes. *Statistical Analysis and Data Mining* 5, 4 (Aug. 2012), 265–283.
[15] Ruoqian Liu, Ankit Agrawal, Wei-keng Liao, and Alok Choudhary. 2014. Search space preprocessing in solving complex optimization problems. In *Proc. IEEE Big Data*. IEEE, 1–5.
[16] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource Management with Deep Reinforcement Learning. In *Proc. 15th ACM Hotnets*. ACM, New York, NY, USA, 50–56.
[17] Marcio Melo, Susana Sargento, Ulrich Killat, Andreas Timm-Giel, and Jorge Carapinha. 2013. Optimal Virtual Network Embedding: Node-Link Formulation. *IEEE Trans. Network and Service Management* 10, 4 (Dec. 2013), 356–368.
[18] Maria Carolina Monard and Gustavo E A P A Batista. 2002. Learning with skewed class distributions. *Advances in Logic, Artificial Intelligence and Robotics* (2002), 173 – 180.
[19] Keith Winstein and Hari Balakrishnan. 2013. TCP Ex Machina: Computer-generated Congestion Control. In *Proc. ACM SIGCOMM (SIGCOMM '13)*. ACM, New York, NY, USA, 123–134.
[20] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. 2008. Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration. *SIGCOMM Comput. Commun. Rev.* 38, 2 (Mar. 2008), 17–29.