Optimization of Decision Trees for TCP Performance Root Cause Analysis

Marco Weiss, Simon Bauer*, Benedikt Jaeger*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: marco.weiss@tum.de, bauersi@net.in.tum.de, jaeger@net.in.tum.de

Abstract—**Identifying the root cause for TCP throughput limitations helps to improve network performance and user experience. In previous work, decision trees (DTs) have been used as a tool for TCP performance root cause analysis (RCA) based on passive network measurements. The topology of those trees has been designed based on the working principles of TCP and the decision thresholds were chosen by inspection of measured data. We present how genetic algorithms (GAs) can be used to further optimize those DTs by fitting their threshold values to a dataset of synthetic network traffic with known root causes. In the next step, machine learning algorithms, namely decision tree learning, random forest and extremely randomized trees, are used to build DT-based classifiers in a purely data-driven fashion. It is shown that the classification accuracy of the hand-crafted DTs could be improved after optimizing their thresholds with our GA approach. However, even the optimized hand-crafted DTs were outperformed by the machine learning approaches with a significant margin.**

Index Terms—**TCP IP, decision trees, machine learning, evolutionary computation**

1. Introduction

As TCP is one of the most widely used transport layer protocols, any TCP performance issues might directly impact its users. RCA tools help to identify and overcome such issues. In [1] and [2], Siekkinen et al. and Stemplinger developed tools for TCP performance RCA based on passive network measurements using DTs. DTs are an intuitive and interpretable technique that employs the divide-and-conquer strategy for decision making. Due to their intuitive use, it is possible to design DTs from hand by analyzing the functionality of the underlying system. This is especially the case for "white-box" systems like TCP, where all internal structures and functions are in principal known. A different approach for building DTs comes from the field of machine learning, where the tree structure and its decision rules are purely based on statistical properties of data generated by the system. Despite having no knowledge of the data generating process, DT learning algorithms perform quite well in practice.

In this work, we aim to evaluate both approaches on the same dataset. To this end, we do not only use the dataset to train classifiers with different DT learning algorithms, but we also try to further improve the classification performance of the existing hand-crafted DTs by fitting their threshold values to our data. This is in fact not trivial because common DT learning algorithms need to have control over both the tree topology and the decision thresholds to achieve good performance. Thus, we need to formulate the task as a general optimization problem. To solve it, we chose to use GAs for two main reasons: GAs are easy to implement and they impose almost no limitations to the optimization problem at hand, compared to e.g. gradient-based methods that require a differentiable objective function or linear programming that requires a linear objective function (both is not the case for DT optimization which is in fact NP-complete [3]).

The remainder of this paper is organized as follows: First, related work to the fields of TCP RCA, DTs and GAs for DT oprimization is presented in section 2. After introducing our dataset in section 3, we present the baseline DTs, how they can be optimized with GAs and different machine learning approaches in section 4. The setup and results from our experiments are presented in section 5 before summarizing our findings in section 6.

2. Related work

In [4], Zhang et al. were the first to perform a holistic analysis on the limiting factors of throughput in internet connections. Based on their findings, they developed T-RAT, a tool for RCA based on trace files. Siekkinen et al. extend this work in [1] to overcome limitations of T-RAT that are discussed in [5] in detail. They introduce a set of quantitative metric, called *limitation scores*, which can be inferred from TCP headers and are then used in a DT-based RCA tool. In [2], Stemplinger extends their approach and uses synthetic training data generated by the Mininet network emulator to adapt the decision thresholds to more recent congestion control algorithms.

Closely related to the work done on throughput RCA is [6], where Jaiswal et al. aim to estimate the sender's congestion window size and the connection round trip time (RTT) from passive measurements. They explicitly demarcate their work from [4], but claim that the limitating factors of a TCP connection can be determined based on congestion window size and RTT. This work is of particular interest because in [7] and [8], Hagos et al. use machine learning techniques, namely random forest, gradient boosting and recurrent neural networks, to significantly improve prediction performance compared to the state machine approach from [6]. Quite similar to the machine learning part of our work is [9], where El Khayat et al. use decision tree boosting to discriminate between TCP package loss due to overflow or link errors in wireless networks.

The foundations of DTs, their extensions and learning algorithms can be found in [10], [11]. Decision tree learning, i.e. finding the combination of optimal split dimensions and thresholds, has been proven to be NP-complete [3]. State-of-the-art decision tree learning algorithms, e.g. classification and regression tree (CART) as implemented in the scikit-learn machine learning library [12], use a greedy heuristic to determine the split that maximizes the purity of the resulting distributions or the accuracy for every new node. In general, DTs have several advantages as they are easy to interpret, robust to outliers and scale well to large datasets. However, they are considered high-variance estimators, meaning their prediction performance might be worse than with other machine learning methods in some cases. To deal with this issue, several extensions to DTs have been proposed. The probably most-widely known one is random forests by Breiman [13], where the prediction is computed as the average of an ensemble of different DTs. Building on that, Geurts et al. later introduced extremely randomized trees (extra-trees), where the construction of all trees in the ensemble is completely randomized instead of using a split heuristic [14]. We will refer to both techniques as ensemble methods in the following.

A fundamental introduction to genetic algorithms is given in [15]. In combination with DTs, GAs have previously been used for pre-processing, i.e. selecting the best subset of a large feature space which is then used as input for a heuristic-based decision tree learning algorithm [16]. There exist also attempts to directly use GAs to build DTs. In [17], Papagelis et al. achieved comparable classification performance to heuristic-based approaches when optimizing their DT with GAs. In [18], Cha et al. used GA-based optimization to build compact, nearly-optimal decision trees.

3. RCA Dataset

We train and evaluate all our models on the dataset from [2]. It was generated using the network emulator Mininet with different test setups and network topologies to enforce different throughput limitations. In the context of TCP performance RCA, those throughput limitations will be referred to as the *root causes*. As in [1] and [2], only bulk transfer periods (BTP), i.e. the time window in which throughput is limited by the network connection and not the sending application, are analyzed. After the BTPs have been isolated from the application limited periods, five *limitation scores* were calculated for each BTP. All limitation scores are based on information contained in the TCP headers, so measurements can be obtained passively at any point in the connection [1]. In the following, we will give a brief summary of the possible root causes and limitation scores derived in [1], [2] and provide an overview of the used dataset.

3.1. Root Causes

Capacity bottleneck: The throughput of a connection can be limited by the bandwidth available at the bottleneck link. We distinguish between unshared bottlenecks (ub), where our connection uses the entire bandwidth of the bottleneck link and shared bottlenecks (sb), where parts of the bottleneck bandwidth are used for other transmissions.

Receiver window (rw): The receiver-side application sets the size of the receiver window, i.e. the number of possible bytes per packet, based on how fast it can process incoming data. The receiver window can be static or dynamically scaled by the receiver application during transmission. In the first case, a combination of small default window size, high bandwidth and rather long transmission times can limit the throughput unintentionally. In the latter case, the application can limit throughput intentionally if it cannot process incoming data fast enough.

Congestion avoidance (cw): On sender-side, the congestion control algorithm tries to estimate the best sending rate for the connection. Depending on its implementation, there might be phases in which the throughput is solely limited by the congestion control algorithm, e.g. at the beginning of a connection or after the congestion window was lowered due to detected packet loss.

3.2. Limitation Scores

Dispersion score: The dispersion score is defined as

$$s_{disp} = 1 - \frac{TP}{C}, \tag{1}$$

where TP is the average throughput of the BTP and C is the capacity of the bottleneck link. The dispersion score can be used to determine whether a connection is limited by an unshared bottleneck ($s_{disp} \approx 0$) or a shared bottleneck ($s_{disp} > 0$).

Retransmission score: The retransmission score is defined as the ratio of retransmitted bytes to transmitted bytes

$$s_{retr} = \frac{n_{retr}}{n_{trans}}. \tag{2}$$

A high retransmission score is an indicator for a network bottleneck where the link buffer is filled up until packets are dropped and have to be retransmitted.

RTT score: The RTT score is an alternative to the retransmission score for detecting network bottlenecks and is defined as

$$s_{RTT} = \frac{avg(RTT)}{min(RTT)}, \tag{3}$$

where RTT is the round trip time measurement of a packet, i.e. the time between sending the packet and receiving the acknowledgement. Essentially, the RTT score indicates a bottleneck if the current RTT is higher than the lowest possible RTT measured so far.

Receiver window score: The receiver window score s_{rwnd} quantifies how much the sender is limited by the advertised receiver window. This is done by comparing the number of outstanding bytes and the receiver advertised window size over time. If the difference between both values is small, i.e. below a certain threshold, the congestion window of the sender is close to the limit set by the receiver. The receiver window score is then calculated as the average number of occurrences for this event over the duration of a BTP.

Burstiness score: The burstiness score (or b-score) s_b can be used to determine whether connections with high receiver window score are actually limited by the size of the receiver window or by a bandwidth bottleneck. When transmitting n packages in a receiver-limited setting, a burst of $n-1$ short inter-arrival times (IAT) is followed by a long IAT because the sender has to wait for the receiver's acknowledgement. If the transmission is however bandwidth-limited, the distribution of the IATs is more even due to buffering at the bottleneck link.

3.3. Overview

The complete dataset consists of 533 measurements that were taken during different BTPs. Each measurement vector $x = (s_{disp}, s_{retr}, s_{RTT}, s_{rwnd}, s_b) \in \mathbb{R}^5$ is labelled with the true root cause $y \in \{cw, rw, sb, ub\}$. It has to be noted that different combinations of root causes are possible in theory. In practice however, there is usually a single dominant root cause per connection [1]. To visualize the dataset, principle component analysis (PCA) is applied to the normalized data. Using the first two components, the dataset can be plotted as shown in Figure 1. It has to be noted that the visualization captures only 66% of the variance in the data. Nevertheless, this still gives a first impression how the data is structured. It can e.g. already be seen that the all measurements with $y = rw$ can be linearly separated from the other classes.

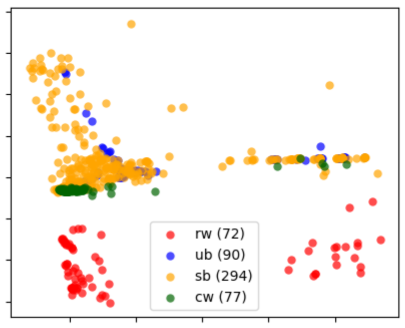

Figure 1: Dataset visualization using the first two components from PCA (capturing 66% of the variance in the data). The number in brackets indicates the number of data points per class.

4. Methodology

In the following, different strategies will be described to derive DTs for TCP performance RCA from the available data. As baseline, we use two hand-crafted DTs from [1] and [2], where decision thresholds were manually defined in [2] based on inspection of the dataset. In the next step, we aim to improve the classification accuracy of those trees by using GA to optimize the thresholds. To compare the performance of hand-crafted DTs with a purely data-driven machine learning approach, we use DT learning, namely CART, in chapter 4.3. In chapter 4.4, we use ensemble methods, namely random forest and extra-trees, which are expected to improve classification performance compared to single DT learning.

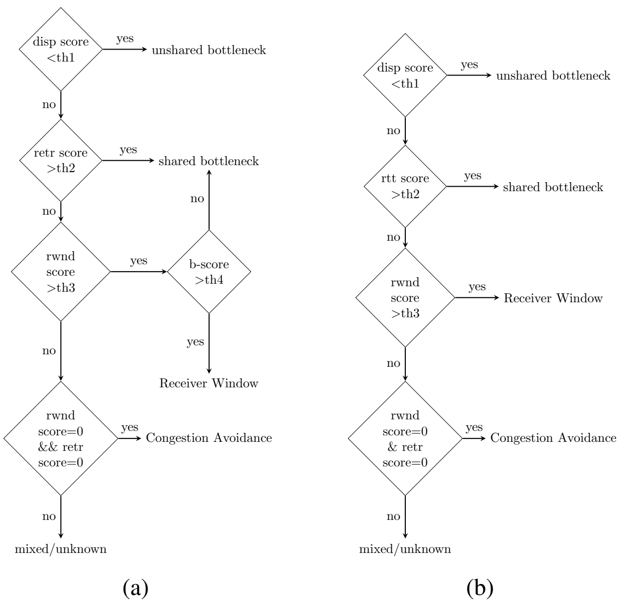

Figure 2: Baseline decision tree (a) and baseline RTT decision tree (b), both taken from [2].

4.1. Fitting by Inspection

In [1], Siekkinen et al. proposed a DT based on dispersion score, retransmission score, receiver window score and burstiness score. In [2], the threshold values of this DT where refined by manually analyzing the dataset described in the previous chapter. In the following, we will refer to this as the *baseline* approach. Additionally, a modified version of the baseline tree was presented using the RTT score instead of retransmission and burstiness score, called *baseline RTT*. The trees are depicted in Figure 2. It has to be pointed out that threshold fitting of both trees was done using the complete dataset. For our other approaches below, we perform a 80/20 train-test split to measure generalization performance on unseen data.

4.2. Optimization with Genetic Algorithm

The general idea of GAs is that evolution in biology can be seen as an optimization problem: In a population of individuals, the ones adapted best to their environment survive the longest and the older an individual gets the more time it has to reproduce. Thus, the genes of the best-fitting individuals spread while "unsuitable" genes vanish over many generations. In a more mathematical sense, natural selection can be seen as a search heuristic to find the best genes (hence the name Genetic Algorithms). To apply this heuristic, candidate solutions to the optimization problem have to be encoded in chromosomes. Their fitness is evaluated based on an objective function and the fittest candidate solutions generate new candidate solutions based on genetic operators [15]. In the following, we will describe how this process can be applied to the decision threshold optimization problem. The resulting DTs will be referred to as *optimized* and *optimized RTT*.

The first step is to define the encoding of candidate solutions. Compared to other approaches e.g. in [17] where a candidate solution has to encode a complete tree topology, our encoding is rather simple because we only want to optimize the decision thresholds for a fixed topology. In accordance with the notation of Figure 2, we have candidate solutions in the form of $c_{base} = \{th_1, th_2, th_3, th_4\}$ and $c_{baseRTT} = \{th_1, th_2, th_3\}$, where $\forall j \; th_j \in \mathcal{T}_j$ and $\mathcal{T}_j$ denotes the set of possible threshold values for the j-th dimension of the input vector. Using a set of discrete threshold values instead of real-valued numbers drastically reduces the search space without affecting the training accuracy of the candidate solutions. Analogously to heuristic-based DT learning [11], the threshold sets for a given training dataset $\mathcal{D}$ are obtained by $\mathcal{T}_j = \{x_j : x \in \mathcal{D}\}$.

After the encoding has been defined, a start population is created by generating n_{pop} individuals with random values from the threshold sets. To evaluate the *fitness* of a candidate solution, accuracy on the training dataset is used. *Selection* is done in an elitist way: The best n_{elit} individuals are kept for the next generation without any changes. On the remaining $n_{pop} - n_{elit}$ individuals, *crossover* and *mutation* can be applied. In original GA implementations, one or more crossover points are chosen at which the parent chromosomes are split and exchanged between both partners to maintain so-called building blocks [15]. In our implementation however, the order of the elements is completely arbitrary and not meaningful, so we do not need to maintain any building block structures in the solution. Therefore, we choose parameterized uniform crossover with probability of 50%, which essentially means that the elements in the offspring chromosome are randomly chosen from both parents. For every offspring, the event mutation happens with a configurable probability. In case of mutation, one element c_j of the chromosome is replaced by a random element in $\mathcal{T}_j$. In nature, mutation can only happen during reproduction. This does of course not apply to a virtual implementation of such genetic operators, so it is possible to apply mutation to any individual and not only to new offsprings. In Figure 3, graphical examples for the genetic operators described above are given.

Starting with the initial population, optimization is an iterative process where every iteration corresponds to a new *generation*. After a fixed number of generations, i.e. when convergence is expected based on experiments presented in the next section, the best candidate solution of the final generation is returned. To avoid converging to a local minimum, GAs try to maintain a set of good, but possibly very different solutions. Random mutation also helps solution candidates to overcome local minima. The effectiveness of those concepts highly relies on a suitable combination of hyperparameters, primarily the population size and the probabilities for crossover and mutation. We tune those hyperparameters by comparing the results from multiple runs of GA optimization to the actual global optimum obtained from brute force calculations. Detailed setup and results of those experiments can be found in section 5.1.

4.3. Decision Tree Learning Algorithm

For DT learning, we use the DT classifier class from scikit-learn [12]. It uses the CART algorithm and the best split is chosen based on the Gini score, a measurement

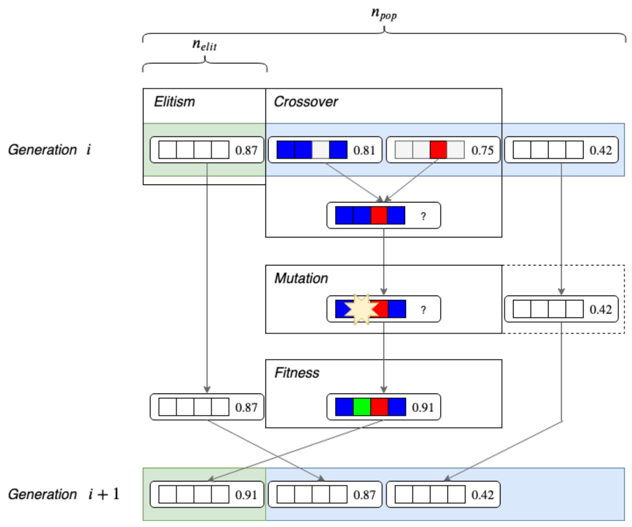

Figure 3: Schematic visualization of genetic operators. White boxes with rounded corners represent candidate solutions with four genes (squares on the left) and a fitness score (right). Red and blue color indicate the origin of a gene during crossover. After mutation, one gene in the offspring is replaced by a random value from the threshold set (green color). The dashed line indicates that depending on the implementation, mutation is also possible for individuals that are not the result of a crossover operator. For clarity, only a single crossover operation is shown and only one individual is passed to the next generation. In practice, both has to be done multiple times to keep the population size constant.

for the impurity of a distribution. As pointed out in the scikit-learn documentation, the training algorithm is biased towards the dominant classes when training a DT on an unbalanced dataset, i.e. data with an uneven distribution of class labels. To overcome this issue and still make use of the complete dataset, each class is weighted with the reciprocal of its relative occurrence. Before training the tree on the complete training dataset, we perform grid search with K-fold cross validation to find the set of hyperparameters that performs best on the test data. A good selection of hyperparameters is mainly important to prevent overfitting. Without any limitations, the tree can be grown until every leaf node is pure, thus giving a training accuracy of 100 % but bad generalization performance. We tune the following hyperparameters to control growth of the tree: Maximum depth, maximum number of leaf nodes, minimum impurity decrease and minimum number of samples for a split.

It has to be noted that in contrast to our baseline trees, the scikit-learn DT implementation does not output *mixed/unknown* as possible root cause. The classifiers always predicts the most likely class, i.e. the one dominating the distribution in the leaf node - a class which is not in the training dataset cannot be predicted. If a classification as *mixed/unknown* is desired, a discrimination based on the likelihood of the predicted class could be a possible solution.

4.4. Ensemble Methods

The scikit-learn library [12] also provides implementations of random forest and extra-trees that we use here. The most important hyperparameters for both methods are the number of trees in the ensemble and the number of features that are considered for every split. For every tree that is built within the ensemble, the same hyperparameters for DT learning apply as discussed in the previous section. In contrast to a single DT however, it is recommended to grow all trees to full size because generalization is essentially achieved by averaging over all trees in the ensemble.

5. Experiments

In the following, two different experiments are presented. First, it is shown on small subsets of the dataset that our GA optimization converges to a nearly-optimal solution with a high probability using a suitable set of hyperparameters. In the second experiment, DT learning and ensemble methods are trained on the dataset and compared to the accuracy of the hand-crafted DTs.

5.1. Convergence of Genetic Algorithm

As motivated in section 4.2, we want to obtain a good set of hyperparameters for our GA-based DT optimization. To this end, we create 3 subsets by randomly sampling 10% of the training dataset. We use a brute-force approach to obtain the parameter sets of the baseline and the baseline RTT tree that maximize the training accuracy on every subset. The resulting accuracies, marked as horizontal dashed lines in Figure 4, are then used as benchmark for the GA-based optimization. To account for the reduced search space by using only 10% of the data, we scale down the population size to 30. We run the GA 10 times per subset and per DT with different random seeds and show the average best-of-generation fitness in Figure 4. It can be seen that the GA converges to a nearly-optimal solution with high probability, while its average training time is faster than the brute-force approach by approximately factor 100. Due to the reduced number of threshold parameters, optimization of the baseline RTT tree with 3 parameters converges in fewer generations than the baseline tree with 4 parameters. Based on tuning the GA hyperparameters to good convergence, we use a crossover probability of 0.5, mutation probability of 0.2 and an elitism ratio of 0.1. We found that a population size of 100 is good compromise between final accuracy and training time on the complete dataset.

5.2. Comparison to Decision Tree Learning Algorithm and Ensemble Methods

As described in section 4.3, we perform grid-search and K-fold cross validation to determine the best hyperparameter set for the DT learning algorithm. As it can be seen in Figure 5, overfitting is in fact not a problem in our case. Although training accuracy is at 100%, there is no significant decrease of validation accuracy. This could be an indicator that there might not be significant noise in the synthetic training data. However, a more detailed analysis of this phenomena is required.

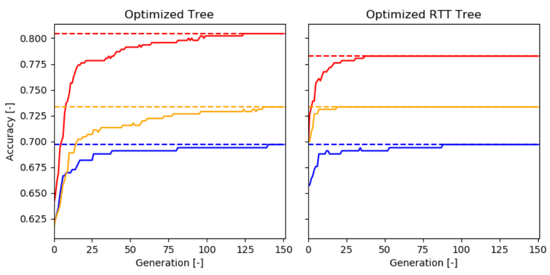

Figure 4: Best-in-generation accuracy of DTs optimized with GA compared to upper boundary (dashed line) for 3 small training data subsets.

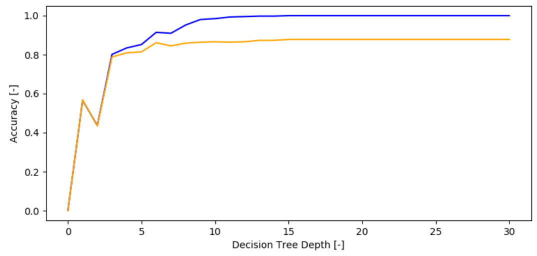

Figure 5: Training accuracy (blue) and validation accuracy (orange) of the DT learning algorithm as function of maximum depth.

For our final evaluation, we use a maximum depth of 15. It has to be noted that a smaller tree, e.g. with depth of 5, achieves only slightly worse performance. In Figure 6, the resulting DT is shown up to a depth of 2 for the sake of readability. It can be seen that the first split separates all points of class *rw* in the training set from the other classes as expected from Figure 1. This leads to a missclassification rate of 0% for class *rw* on the training data. The hand-crafted DTs, which have a missclassification rate of over 34% for this class [2], could maybe be improved by also performing the split on s_{rwnd} first.

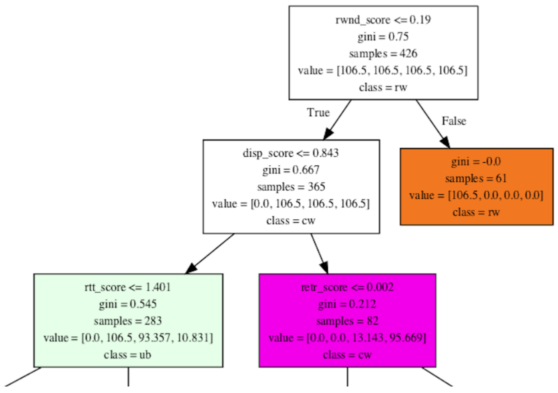

Figure 6: DT fitted to the training data shown up to a depth of 2.

Training of both ensemble methods as implemented in the scikit-learn library is comparable straight forward using the default hyperparameters. The final accuracies of all presented approaches are listed in Table 1. It has to be noted that the accuracy of both baseline trees is somewhat biased. On the one hand, the decision thresholds were determined using the complete dataset, whereas all other approaches are tested on unseen data. On the other hand, the baseline trees and their GA-optimized versions output classifications of type *mixed/unknown*, which might be useful in some cases in practice, but is always considered a wrong prediction when calculating accuracy.

Method	Train Time	Accuracy	Improvement
Baseline	-	0.73	-
Baseline RTT	-	0.70	-
Optimized	46.2s	0.79	8.2%[1]
Optimized RTT	44.2s	0.75	7.1%[2]
DT learning	< 0.1s	0.92	26.0%[1]
Random Forest	0.1s	0.93	27.4%[1]
Extra-Trees	0.1s	0.94	28.8%[1]

TABLE 1: Train times on 1.6 GHz Intel Core i5, final accuracies on test data and relative improvement compared to the respective baseline tree accuracy.

[1] Compared to *baseline* tree.
[2] Compared to *baseline RTT* tree.

6. Conclusion and Future Work

The main goal of this work was to optimize existing decision trees for TCP performance RCA on a given dataset. With GA-based optimization, we were able to improve their classification accuracy by up to 8%. We could show for small subsets of the data that the GA-based optimization of DTs converges to a near-optimal solution with high probability. It can therefore be assumed that the performance of the baseline DTs is limited by their design. Consequently, we used DT learning to optimize not only the decision thresholds of the DT but also its topology. By doing so, we could improve classification performance by 26% compared to the baseline trees. With ensemble techniques, namely random forest and extra-trees, we achieved marginally better performance than with DT learning. However, it has to be considered that interpretability and explainability of the predictions decrease significantly with more complex methods compared to the original DTs: To explain why a certain prediction has been made, it is in practice possible to trace every step in a DT of depth 5. For an ensemble consisting of 100 full-grown DTs, this is very likely not the case. If it is however desired to further increase the prediction accuracy, the machine learning approach could be taken even further in future work: Instead of using hand-crafted features as input for classification, i.e. the limitation scores in our case, classifiers like neural networks could be trained directly on the temporal data extracted form the TCP header files.

References

[1] M. Siekkinen, G. Urvoy-Keller, E. W. Biersack, and D. Collange, “A root cause analysis toolkit for tcp,” *Computer Networks*, vol. 52, no. 9, pp. 1846–1858, 2008.

[2] L. J. Stemplinger, “Tcp flow performance root cause monitoring,” Bachelor’s Thesis, Technical University of Munich, 2019.

[3] S. K. Murthy, “Automatic construction of decision trees from data: A multi-disciplinary survey,” *Data mining and knowledge discovery*, vol. 2, no. 4, pp. 345–389, 1998.

[4] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker, “On the characteristics and origins of internet flow rates,” in *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4. ACM, 2002, pp. 309–322.

[5] M. Siekkinen, “Root cause analysis of tcp throughput: Methodology, techniques, and applications,” in *PhD thesis, Institut Eurécom/Université de Nice-Sophia Antipolis, Sophia Antipolis*, 2006.

[6] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, “Inferring tcp connection characteristics through passive measurements,” in *IEEE INFOCOM 2004*, vol. 3. IEEE, 2004, pp. 1582–1592.

[7] D. H. Hagos, P. E. Engelstad, A. Yazidi, and Ø. Kure, “A machine learning approach to tcp state monitoring from passive measurements,” in *2018 Wireless Days (WD)*. IEEE, 2018, pp. 164–171.

[8] ——, “Recurrent neural network-based prediction of tcp transmission states from passive measurements,” in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2018, pp. 1–10.

[9] I. El Khayat, P. Geurts, and G. Leduc, “Improving tcp in wireless networks with an adaptive machine-learnt classifier of packet loss causes,” in *International Conference on Research in Networking*. Springer, 2005, pp. 549–560.

[10] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[11] K. P. Murphy, *Machine Learning. A Probabilistic Perspective*. The MIT Press, 2012.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[13] L. Breiman, “Random forests,” *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[14] P. Geurts, D. Ernst, and L. Wehenkel, “Extremely randomized trees,” *Machine learning*, vol. 63, no. 1, pp. 3–42, 2006.

[15] M. Mitchell, *An Introduction to Genetic Algorithms*. The MIT Press, 1996.

[16] J. Bala, J. Huang, H. Vafaie, K. DeJong, and H. Wechsler, “Hybrid learning using genetic algorithms and decision trees for pattern classification,” in *IJCAI (1)*, 1995, pp. 719–724.

[17] A. Papagelis and D. Kalles, “Ga tree: genetically evolved decision trees,” in *Proceedings 12th IEEE Internationals Conference on Tools with Artificial Intelligence. ICTAI 2000*. IEEE, 2000, pp. 203–206.

[18] S.-H. Cha and C. C. Tappert, “A genetic algorithm for constructing compact binary decision trees,” *Journal of pattern recognition research*, vol. 4, no. 1, pp. 1–13, 2009.