Learning to Prune Instances of Steiner Tree Problem in Graphs

Jiwei Zhang
University College Dublin
Dublin, Ireland

Dena Tayebi
University College Dublin
Dublin, Ireland

Saurabh Ray
New York University
Abu Dhabi, United Arab Emirates

Deepak Ajwani
University College Dublin
Dublin, Ireland

ABSTRACT

We consider the Steiner tree problem on graphs where we are given a set of nodes and the goal is to find a tree sub-graph of minimum weight that contains all nodes in the given set, potentially including additional nodes. This is a classical NP-hard combinatorial optimisation problem. In recent years, a machine learning framework called learning-to-prune (L2P) has been successfully used for solving a diverse range of combinatorial optimisation problems. In this paper, we use this learning framework on the Steiner tree problem and show that even on this problem, the learning-to-prune framework results in computing near-optimal solutions on a large majority of the instances at a fraction of the time required by commercial ILP solvers. Furthermore, we show that on instances from the SteinLib and PACE Challenge datasets where the L2P framework does not find the optimal solution, the optimal solution can often be discovered by either using a lightweight local search algorithm to improve the L2P solution or using L2P solution as a warm start in an ILP solver. Our heuristic for Steiner tree problem leverages historical solutions of known solutions for past instances from the same distribution. Our results underscore the potential of the L2P framework in solving various combinatorial optimisation problems.

KEYWORDS

Minimum Steiner Tree, Combinatorial Optimisation, Machine Learning, Learning to Prune

1 INTRODUCTION

Steiner tree problem is a classical well-studied combinatorial optimisation problem. It is applied to various problems in research and industry [12] including various network design problems (see e.g., [9]).We consider the variant of this problem on graphs, where we are given an input weighted graph, a set of terminal nodes V and the goal is to compute a minimal connected subgraph that contains all nodes in V. This variant of the problem is NP-hard. Furthermore, it is even NP-hard to approximate it within a factor of 96/95 [3]. Given its applications and hardness, numerous approximation algorithms and heuristics have been developed to solve this problem efficiently. We refer the reader to the PACE challenge 2018 report [2] for a list and ranking of the various algorithms and heuristics developed for this problem.

Traditional approaches to solve combinatorial optimisation problems include the usage of integer linear programming solvers, constraint programming approaches, parameterised and approximation algorithms, various heuristics including nature based metaheuristics (e.g., genetic algorithms) and customising algorithms to specific input distribution. In recent years, machine learning techniques have been explored to speed-up the computation of solutions (c.f., [1] for a survey). Machine learning techniques are particularly useful in applications where the same optimisation problem is solved repeatedly on instances coming from the same distribution [6]. Many of these learning techniques aim to learn the optimal solution directly. An example of such an end-to-end machine learning technique on Steiner tree problem is the Cherrypick solution by Yan et al. [19], where a deep reinforcement learning technique called DQN is used together with an embedding to encode path-related information in order to predict the optimal solution directly. Such end-to-end approaches generally suffer from poor generalisation (resulting in poor solutions for larger and/or more complex problem instances) and large training requirement. To deal with these issues, these approaches would need to collect the training data by solving a large number of problem instances of the same size as the test instances. Furthermore, these end-to-end deep learning approaches also suffer from poor interpretability and explanability of the algorithms learnt. This is because the learnt algorithm is implicit in the millions of parameters of the deep learning architecture. Since in real industry setting, new constraints are routinely added to the problem, poor interpretability means that we do not know if the learnt model will still work with newly discovered constraints and thus, new models have to be learnt from scratch every time this occurs.

A key reason for the poor generalizability of end-to-end approaches is that they do not leverage any algorithmic insight into the problem, instead relying solely on the input and embedding vectors. This is also an important factor for them needing deep learning models with poor interpretability. To address some of these issues, a learning-to-prune approach [4, 10, 11, 16] has been proposed. Instead of trying to predict the optimal solution directly, it uses a supervised learning model to predict the elements (e.g., nodes/edges) that are unlikely to be part of optimal solution and prune them from further consideration. Once these elements are pruned out, the problem on the remaining elements (predicted to be in optimal solution by the classifier or where the classifier was not confident) is usually quite small and can, thus, be solved using existing exact/approximate approaches. The supervised learning approach leverages a large number of features that can capture the algorithmic insights from the state-of-the-art literature on the

problem. As such, classification techniques with significantly fewer parameters, such as random forest, SVM etc. work very well in this framework and there is no need for more complex deep learning models. An added advantage of this framework is that it requires far fewer labelled training instances for training, which is vital for NP-hard optimisation problems.

In this paper, we explore if the learning-to-prune framework can be used to solve the Steiner tree problem efficiently. Towards this end, we carefully select an ILP formulation with small integrality gap and the features used in the learning approach. We consider the instances from SteinLib dataset [7, 8] and the 2018 PACE challenge [2]. We show that on the instances where the LP relaxation doesn't return integral solutions, learning-to-prune framework is able to obtain optimal or near-optimal solution in orders of magnitude less time compared to solving the ILP formulation directly using Gurobi. Furthermore, we show that even on instances where the solution returned by learning-to-prune is not optimal, we can often achieve optimal solution by using a lightweight local search algorithm to improve the learning-to-prune solution. Using the learnt solution as a warm start in Gurobi is also quite effective in finding optimal or near-optimal solutions for many of the instances where the learning-to-prune solution is not already optimal.

We note that many state-of-the-art Steiner tree problem solvers [12–15] use preprocessing and problem reduction techniques and some of these techniques rely on features similar to ones used in our adaptation of learning-to-prune framework. While these solvers are very successful in practice, they need to be carefully redesigned for each new variant of the problem. In contrast, the effort required to adapt the learning-to-prune framework to the other variants is likely to be considerably less. Thus, our work paves the way for augmenting the ability of algorithm designers to develop preprocessing techniques that can leverage the specific application constraints and the input distributions. Our learnt preprocessing can be combined with well-known "exact" preprocessing techniques in different ways.

2 APPLYING LEARNING-TO-PRUNE TO STEINER TREE PROBLEM

In the context of minimum Steiner tree problem, the learning-to-prune framework learns a classification model to predict whether an edge will be part of the optimal solution or not. The training examples consists of edges from a set of training graphs and the classification model learns a mapping from a feature vector of an edge to its label. The edges, for which the classifier is highly confident that they are not part of the optimal solution, are pruned out and the remaining (hopefully much smaller) instance is then solved using an ILP solver.

The key decisions we need to make in order to apply the learning-to-prune framework involve (i) the choice of the ILP formulation for the Steiner tree problem, (ii) the choice of features and (iii) the classification models. We found that the choice of the ILP formulation was crucial for the success of the learning-to-prune framework on this problem. An ILP formulation with smaller integrality gap seems to be particularly suitable for this framework as its LP relaxation can be used as a highly discriminative feature in the process of search-space pruning. Thus, we carefully considered the various ILP formulations for this problem and opted for a formulation based on multicommodity flow transmission [18].

We note that there are other formulations such as those based on directed cut [13] that are algorithmically more efficient (see the survey article [12]). We expect that the gains from learning-to-prune approach will be similar for these other formulations as well.

2.1 Integer Linear Programming Formulation

In this formulation, we first convert an undirected Steiner tree problem into a directed version by replacing each edge $\{i, j\}$ with weight c_{ij} by two directed edges (i, j) and (j, i) of the same weight c_{ij}. Then, we consider the problem of connecting the set of terminal nodes in the undirected graph as sending a unit flow from an arbitrary terminal node (referred to as root and indexed as node 1) to the remaining terminal nodes in the corresponding directed graph. In particular, the k^{th} flow goes from the root to the k^{th} terminal node (the first flow goes from the root node to itself). Since all flows are moving away from root and towards the terminal nodes, the aggregation of these paths result in the Steiner tree in the undirected graph. Next, we describe this formulation in more detail:

2.1.1 Sets and Indices.

- $i \in N = \{1, 2, .., n\} = \{1\} \cup V \cup S$: The index number of root node is 1. Here, $\{1\} \cup V$ is the set of terminal nodes and S is the set of remaining nodes.
- $E = \{(i, j)\}$: Set of directed edges. Note that the size of E is double the size of the edge set of the undirected graph.
- $G = (N, E)$: A graph where the set N defines the set of nodes, the set E defines the set of directed edges and the set $\{1\} \cup V \subseteq N$ defines the set of terminals.
- $T \subset E$: Set of edges that represents a tree spanning $\{1\} \cup V$ in G.
- $c_{ij} \in R^+$: The cost of the arc (i, j), for all $(i, j) \in E$.

2.1.2 Decision Variables.

- $y_{ij} \in \{0, 1\}$: This variable is equal to 1, if edge (i, j) is in the set T. Otherwise, the decision variable is equal to zero.
- x_{ij}^k: This is the amount of commodity k (the amount of flow from node 1 to k) that goes through edge (i, j).

2.1.3 Objective Function. Minimize the total cost of T:

$$minimize \sum_{(i,j)\in E} c_{ij} \cdot y_{ij} \tag{1}$$

2.1.4 Constraints.

$$\sum_{h\in N} x_{ih}^k - \sum_{j\in N} x_{ji}^k = \begin{cases} 1, & i = 1, \\ -1, & i = k, \\ 0, & i \neq 1, k. \end{cases} \quad \forall k \in V, \tag{2}$$

$$x_{ij}^k \leq y_{ij}, \tag{3}$$

$$x_{ij}^k \geq 0, \quad \forall (i, j) \in E, k \in V, \tag{4}$$

$$y_{ij} \in \{0, 1\}. \tag{5}$$

2.1.5 Constraints Explanation. As described before, we use an embedded multi-commodity network flow problem to describe the connectivity of the Steiner tree problem. In constraint 2, one unit of commodity k must be transmitted from node 1 to node k. Constraint 3 indicates that when the flow is allowed in an edge, the edge must be in the solution. Constraint 4 enforces that the flow on any edge is non-negative while Constraint 5 enforces that the variables y_{ij} are binary. Constraints 2- 5 indicate that a feasible solution must have a directed path of edges (i. e. $y_{ij} = 1$) between node 1 and a node belonging to V.

2.2 Features

Another requirement for applying the learning-to-prune framework is to have a set of highly discriminative features that can separate the edges in the optimal solution from those that are not. In other words, we want to associate a set of features with each edge that will allow us to train a classifier separating the set of edges in the optimal solution from the other edges. For the Steiner tree problem, we consider features associated with the LP relaxation, weight of the edges with respect to other edge weights and centrality of associated nodes. Except for Eigenvector centrality, these features can be computed quite fast and they allow us to achieve a high degree of pruning with little loss in objective function value.

2.2.1 LP relaxation feature. The first feature is the value of edge variables y_{ij} in the LP relaxation of the problem. A high value of this variable suggests a higher likelihood of the edge appearing in the optimal solution. We didn't consider any signals from the dual of the problem and we also didn't utilise the variables x_{ij}^k in this study. Note that the computation of LP relaxation requires significantly less time compared to the ILP computation.

2.2.2 Weight-Related Features. We used the following weight-related features: (i) normalised weight $w_N(e) = (w(e) - w_{min})/(w_{max} - w_{min})$ where $w(e)$ is the weight of the edge e and w_{min} and w_{max} are the lightest and heaviest edge-weight in the graph, (ii) standardised weight $w_S(e) = (w(e) - \mu(w))/\sigma(w)$ where $\mu(w)$ and $\sigma(w)$ are the mean and standard deviation of the edge-weights, (iii) chi-square of normalised weight $w_C(e) = (w_N(e) - \mu(w_N))^2/\mu(w_N)$ where $\mu(w_N)$ is the mean of the normalised edge-weights and (iv, v) local rank of edge (i, j) at node i and j. Here, local rank refers to the rank of this edge in the sorted order of all edges (by weight) incident at the node.

2.2.3 Centrality Features. To capture the discriminative power of an edge further, we also use the centrality of the incident nodes. Intuitively, the edges between highly central nodes are more likely to be part of optimal Steiner trees as they are crucial for low-weight connectivity. Specifically, we use the following centrality features: degree centrality, betweenness centrality and Eigenvector centrality. Degree centrality is defined as the number of links incident upon a node, which is the simplest centrality feature to calculate. It simply measures the importance of a node by how many edges are incident to it. Betweenness centrality is widely used in weighted graphs as it captures the fraction of shortest paths passing through a given node [17]. The betweenness centrality of a node v is defined as $C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$ where σ_{st} is the total number of shortest paths between s and t and $\sigma_{st}(v)$ is the number of shortest paths between s and t that pass through v. Eigenvector centrality [20] is also an important centrality feature that captures the "influence" of a node in the network: A high eigenvector score means that a node is connected to many nodes who themselves have high scores.

For all centrality features, NetworkX is used to construct the graph and calculate the feature values [5]. As these centrality features are focusing on nodes instead of edges, each centrality metric results in two features for an edge (i, j) corresponding to the smaller and the bigger value of the two incident nodes i and j.

2.3 Classification

As noted by the previous work on learning-to-prune [4, 10, 11, 16], the exact classification model is not so crucial in this framework. We experimented with five different classification techniques: **Random forest** (RF), **Support vector machine** (SVM), **Logistic Regression** (LR), **K-nearest neighbour** and **Gaussian naive bayes**. While the SVM performs best on this problem, the main insights from the experimental results remain the same for all these classifiers. This provides us confidence that our results are not too specific to a particular classification model, but are more broad-based.

2.4 ILP on the pruned subgraph

We run the exact Steiner tree ILP formulation on the unpruned graph so obtained. This can be done by fixing all the edge variables of the pruned edges to 0 in the ILP formulation and solving the modified ILP using an ILP solver. The output of this modified ILP is then returned as the output of the learning-to-prune approach.

2.5 Ensuring Feasibility

One issue with the learning-to-prune framework is that the remaining set of edges may not contain any feasible solution of the problem that satisfies all constraints. In other words, the pruned graph (graph remaining after the pruned edges have been removed) may have multiple connected components with nodes in V divided between these components. To resolve this issue, we add back all edges for which the corresponding variable has a non-zero value in the LP relaxation. Assuming that the input graph was connected, the set of edges with non-zero values in LP relaxation solution will maintain connectivity among the terminal nodes and thus, with their addition, feasibility will be guaranteed.

Next, we evaluate the quality of this solution as well as the running time of this approach and the relative importance of the different features used in the framework.

3 RESULT

3.1 Experimental Setup

In the benchmark SteinLib [7] dataset, we found that there are only 55 problem instances for which the LP relaxation of the considered ILP formulation does not return integral solutions. Thus, we only focus on these instances and select 80% of them with the smallest running times as training and use the remaining 20% of the instances with the largest running time as the test dataset. This is because we want to show that our model generalises from smaller instances to larger and more complex instances in this dataset.

3.1.1 Feature Importance. The feature importance for training the model is shown in Table 1 for a SVM classification model. Unsuprisingly, Table 1 shows that the LP relaxation feature is the most discriminative of all. An important observation here is that even though LP relaxation feature is important, it accounts for less than half of the discriminative power of all the features. This implies that this feature alone isn't enough, but other features also contribute significantly to improving the accuracy of the classification model and the entire learning framework is needed. We also considered the feature importance from a logistic regression classification model[1]. Similar to the case of SVM, it showed that while the LP relaxation feature is the most discriminative, the other features (such as the maximum degree centrality of the two incident nodes and the minimum local rank of the two incident nodes) also prove to be quite useful in the classification.

Feature	Importance
LP relaxation feature	0.462
Normalised Weight	0.108
Variance	0.083
Degree Centrality Max	0.052
Eigenvector Centrality Max	0.048
Betweenness Centrality Max	0.048
Degree Centrality Min	0.046
Local Rank j	0.041
Eigenvector Centrality Min	0.039
Betweenness Centrality Min	0.038
Local Rank i	0.036

Table 1: Relative feature importance of different features based on a SVM classification model

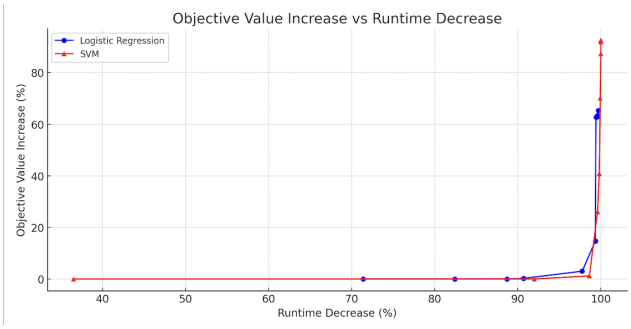

Figure 1: Trade-off obtained by varying the threshold of a SVM classifier

3.1.2 Objective Function vs. Running Time Trade-off. As noted in Figure 1, both SVM classifier and logistic regression classifiers obtained a drastic reduction in running time at little loss in objective function value. In these plots, we vary the pruning thresholds. A pruning rate of 60-70% resulted in a significant reduction in running time while increasing the objective function only slightly. While the general trends are similar between SVM and logistic regression, SVM gives a better trade-off between the objective function value and running time. The drastic reduction in running time at little loss in objective function value is further illustrated in Table 2 and 3, which presents the results of the learning-to-prune approach on the 10 test instances using the SVM classification model. We first note that on these larger and more complex instances, the time to compute all the features including the LP values is very small compared to the time to run the original ILP. More importantly, the running time of the learning-to-prune approach (including the time to compute features and then running the ILP with hard pruning constraint) is around 99% less than the original ILP solver time on these instances (using the Gurobi solver). In 7 of these 10 instances, the hard pruning is able to find the optimal solution itself in significantly less time. In the remaining three instances, the resultant increase in the objective function value because of the mistakes in the pruning process is still very small (less than 0.6%).

	Objective (Original)	Objective (After Pruning)	Objective Increase %
i160-344	8307	8324	0.20
i160-244	5076	5103	0.53
i160-345	8327	8327	0
i160-343	8275	8275	0
i160-342	8348	8355	0.08
i160-313	9159	9159	0
i160-241	5086	5086	0
i160-341	8331	8331	0
i160-245	5084	5084	0
i160-242	5106	5106	0

Table 2: Objective function values returned by Gurobi ILP solver and the learning-to-prune framework (with SVM classifier) for different test instances

	Runtime (Original)	Time to Compute Features	ILP Solver Runtime	Runtime Decrease %
i160-344	27245.21	54.17	157.71	99.22
i160-244	7762.75	25.68	47.13	99.06
i160-345	70653.84	51.93	242.82	99.58
i160-343	20897.36	50.54	114.90	99.21
i160-342	91351.38	60.09	1384.39	98.42
i160-313	3832.54	15.25	84.73	97.39
i160-241	6446.48	24.78	32.78	99.11
i160-341	52473.68	53.65	104.74	99.70
i160-245	3014.05	28.05	15.95	98.54
i160-242	4817.80	27.34	42.81	98.54

Table 3: Running time taken by Gurobi ILP solver and the learning-to-prune framework (with SVM classifier) for different test instances

At this stage, a natural question to ask is how do these results compare with directly pruning based on the LP relaxation values

[1]calculated as the product of the feature coefficient with the standard deviation of feature values in the training set

with different thresholds. Next, we consider the three instances where the hard pruning doesn't get optimal results and compare the results of the hard pruning with pruning based on the LP relaxation value. Figures 2 and 3 presents the result of such a comparison. We observe that the hard pruning provides significantly better objective value vs running time trade-off compared to the Gurobi ILP solver. In particular, note that Gurobi requires considerably more time to reach a comparable objective function value. In all three instances, hard pruning based on a diverse range of features provides solutions with better objective function values compared to directly pruning based on LP relaxation values, even though it takes some more time. In these plots, the dashed orange horizontal line in these curves represent the objective function value of the optimal ILP solution on the instance obtained by pruning all edges with zero LP relaxation value. Note that even this value is higher than the solutions from our hard pruning approach.

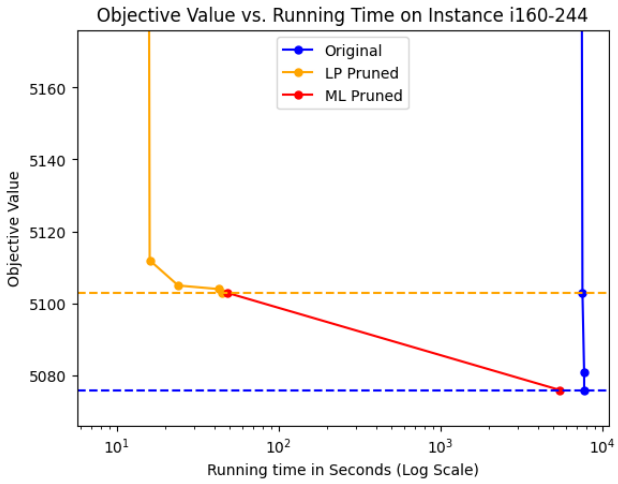

Figure 2: Comparing hard pruning (referred "ML Pruned") with LP-based pruning and Gurobi ILP solver on the original formulation of Steiner tree problem on instances i160-244. The dashed blue horizontal line represents the optimal ILP solution, while the dashed orange horizontal line represents the optimal ILP solution on the instance obtained by pruning all edges with zero LP values.

In applications where we wish to reduce the optimality gap even further, we can use the soft pruning approach. The idea here is that instead of adding a hard constraint that no edge can be taken from the set of pruned edges (fixing those edge variables to 0), we add a soft constraint that a small constant number of edges can be taken from the set of pruned edges in the returned solution. In other words, we add the constraint that sum of all edge variables corresponding to pruned edges has to be less than equal to a small constant. This is implemented by simply adding the corresponding constraint in the ILP formulation. While the soft pruning still retains all the edge variables in the ILP formulation, it prunes the search space considerably. When applied on the instance i160-344, the soft pruning approach, that allows just one edge from the pruned set,

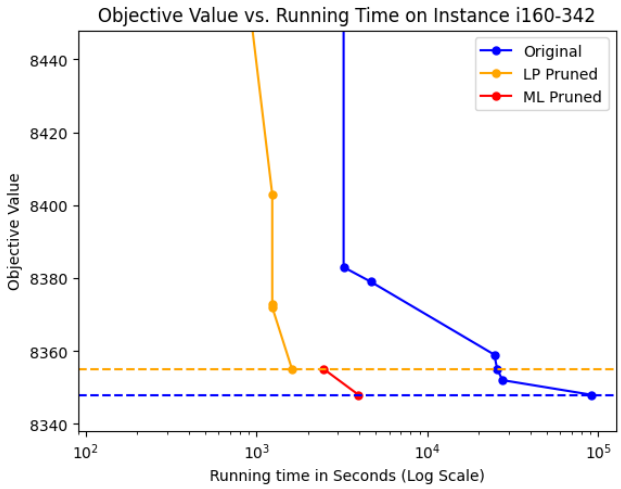

Figure 3: Comparing hard pruning (referred "ML Pruned") with LP-based pruning and Gurobi ILP solver on the original formulation of Steiner tree problem on instance i160-342.

finds the optimal solution of the original problem. The running time of this approach on this instance is around 3000 seconds, which is still considerably less than the original ILP time of around 27000 seconds, but more than the time of the hard pruning approach (around 150 seconds).

3.2 Results on PACE Challenge datasets

Next, we consider the 200 instances from the track 1 of the 2018 PACE challenge [2]. This track provided the benchmark dataset for the exact search techniques. Of the 200 instances, the LP relaxation of our ILP formulation was able to compute the optimal integral solution on 148 instances. On another 17 instances, the LP solution was very close to the optimal integral solution. Thus, we focused on the remaining 35 instances. Out of these instances, we used 13 for training and tested on the remaining 22 instances. Based on the results on the SteinLib dataset, we decided to use SVM as the classification technique for this dataset. Table 4 shows that on all of these instances, the learning to prune approach provides optimal or near-optimal solutions.

On the few instances where the solution returned by learning-to-prune was not optimal, we use a lightweight local search algorithm to improve the solution. We define the local neighbourhood function to consist of solutions that can be obtained by removing one edge from the current solution and adding one replacement edge (from outside the current solution) in its place. We find the best solution in this local neighbourhood. This can be computed efficiently as the returned solution is a tree and removing an edge leaves the tree disconnected. Thus, to find a replacement edge, we only need to consider edges that connect the two components and find a minimum weight replacement edge. We found that this local search algorithm was able to yield the optimal solution when initialised with the learning-to-prune solution. For instance, on instance 010, the objective function values of the optimal solution and learning-to-prune solution are 2338 and 2344, respectively. The local search

	Objective (Original)	Objective (After Pruning)	Objective Increase %
010	2338	2344	0.26
109	939	942	0.32
141	2200557	2200560	0.0001
160	1996	1996	0.0
161	5199	5209	0.19
162	5193	5193	0.0
164	5205	5205	0.0
165	5218	5218	0.0
171	42	42	0.0
172	7229	7304	0.07
173	71	72	1.4
176	10519	10519	0.0
195	54	54	0.0
196	100	101	1.0

Table 4: Objective function values returned by Gurobi ILP solver and the learning-to-prune framework for different test instances of PACE Challenge dataset

was able to improve the learning-to-prune solution to the optimal solution. Similarly on instance 109, the optimal solution and the learning-to-prune solution had objective function values of 939 and 942 respectively. Again, the local search was able to obtain the optimal solution.

We found that using the learning-to-prune solution as a warm start in the Gurobi ILP solver is also quite effective. For instance, on instance 010 (one of the few instances where the learning-to-prune doesn't return an optimal solution), Gurobi warm start from learning-to-prune returns an optimal solution.

The track 2 of the PACE challenge was dedicated to instances that have small tree-width. Surprisingly, we discovered that our learning model that was trained on track 1 instances (exact algorithms track) was quite effective on track 2 instances as well. The solution obtained after pruning was optimal in 9 out of 15 track 2 instances where the LP and ILP solutions had different objective function values. On the remaining 6 test instances, the solution obtained using the learning-to-prune framework was within a multiplicative factor of 1.0002 of the optimal solution. Thus, we conclude that our learning model is also quite robust to changes in input distribution and instance sizes.

4 CONCLUSION

Our experiments show that the learning-to-prune framework provides optimal or near-optimal solutions on instances of the SteinLib and PACE challenge benchmarks at a fraction of the costs of the Gurobi ILP solver. While the feature based on LP relaxation is unsurprisingly the most discriminatory feature for classification, the hard pruning is able to achieve better objective function value compared to pruning directly based on LP relaxation values. It shows that combining the signal from different features using classification models is an effective strategy to prune the problem instances. On the few instances where the learning-to-prune solution is not optimal, we show that using it as a warm start for Gurobi ILP solver or using it as the initial solution for a lightweight local search heuristic can often yield the optimal solution. We expect that this framework will also be effective on many other network optimisation problems.

ACKNOWLEDGMENTS

The research of first and last author is partially supported by Science Foundation Ireland under Grant number 18/CRT/6183.

REFERENCES

[1] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. 2021. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research* 290 (2021), 405–421. Issue 2.

[2] Édouard Bonnet and Florian Sikora. 2019. The PACE 2018 Parameterized Algorithms and Computational Experiments Challenge: The Third Iteration. In *13th International Symposium on Parameterized and Exact Computation (IPEC 2018) (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 115. 26:1–26:15.

[3] Miroslav Chlebík and Janka Chlebíková. 2008. The Steiner tree problem on graphs: Inapproximability results. *Theoretical Computer Science* 406, 3 (2008), 207–214.

[4] James Fitzpatrick, Deepak Ajwani, and Paula Carroll. 2021. Learning to Sparsify Travelling Salesman Problem Instances. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer Cham.

[5] Aric Hagberg, Pieter Swart, and Daniel S Chult. 2008. *Exploring network structure, dynamics, and function using NetworkX*. Technical Report LA-UR-08-5495. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

[6] Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. 2017. Learning Combinatorial Optimization Algorithms over Graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*. 6348–6358. https://proceedings.neurips.cc/paper/2017/hash/d9896106ca98d3d05b8cbdf4fd8b13a1-Abstract.html

[7] T. Koch, A. Martin, and S. Voß. 2000. *SteinLib: An Updated Library on Steiner Tree Problems in Graphs*. Technical Report. Konrad-Zuse-Zentrum für Informationstechnik Berlin ZIB-Report 00-37.

[8] Thorsten Koch, Alexander Martin, and Stefan Voß. 2001. *SteinLib: An Updated Library on Steiner Tree Problems in Graphs*. Springer US, Boston, MA, 285–325.

[9] Moyukh Laha and Raja Datta. 2023. A Steiner Tree based efficient network infrastructure design in 5G urban vehicular networks. *Computer Communications* 201 (2023), 59–71.

[10] Juho Lauri and Sourav Dutta. 2019. Fine-grained search space classification for hard enumeration variants of subset problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33.

[11] Juho Lauri, Sourav Dutta, Marco Grassia, and Deepak Ajwani. 2023. Learning fine-grained search space pruning and heuristics for combinatorial optimization. *Journal of Heuristics* 29, 2-3 (2023), 313–347.

[12] Ivana Ljubic. 2021. Solving Steiner trees: Recent advances, challenges, and perspectives. *Networks* 77, 2 (2021), 177–204.

[13] Tobias Polzin and Siavash Vahdati Daneshmand. 2003. On Steiner trees and minimum spanning trees in hypergraphs. *Operations Research Letters* 31, 1 (2003), 12–20.

[14] Daniel Rehfeldt and Thorsten Koch. 2022. On the Exact Solution of Prize-Collecting Steiner Tree Problems. *INFORMS J. Comput.* 34, 2 (2022), 872–889.

[15] Daniel Rehfeldt and Thorsten Koch. 2023. Implications, conflicts, and reductions for Steiner trees. *Math. Program.* 197, 2 (2023), 903–966.

[16] Dena Tayebi, Saurabh Ray, and Deepak Ajwani. 2022. Learning to Prune Instances of k-median and Related Problems. In *Proceedings of the ACM-SIAM symposium on algorithm engineering and experiments (ALENEX)*. 184–194.

[17] Huijuan Wang, Javier Martin Hernandez, and Piet Van Mieghem. 2008. Betweenness centrality in a weighted network. *Physical Review* 77, 4 (2008).

[18] Richard T. Wong. 1984. A dual ascent approach for Steiner tree problems on a directed graph. *Mathematical programming* 28 (1984), 271–287. Issue 3.

[19] Zong Yan, Haizhou Du, Jiahao Zhang, and Guoqing Li. 2021. Cherrypick: Solving the Steiner Tree Problem in Graphs using Deep Reinforcement Learning. In *IEEE 16th Conference on Industrial Electronics and Applications (ICIEA)*. 35–40.

[20] Mohammed J. Zaki and Wagner Meira Jr. 2014. *Data Mining and Analysis: Fundamental Concepts and Algorithms*. Cambridge University Press. http://www.dataminingbook.info/