Learning to Sparsify Travelling Salesman Problem Instances

James Fitzpatrick[1], Deepak Ajwani[2], and Paula Carroll[1]

[1] School of Business, University College Dublin
{james.fitzpatrick1@ucdconnect.ie, paula.carroll@ucd.ie}
[2] School of Computer Science, University College Dublin
deepak.ajwani@ucd.ie

Abstract. In order to deal with the high development time of exact and approximation algorithms for NP-hard combinatorial optimisation problems and the high running time of exact solvers, deep learning techniques have been used in recent years as an end-to-end approach to find solutions. However, there are issues of representation, generalisation, complex architectures, interpretability of models for mathematical analysis etc. using deep learning techniques. As a compromise, machine learning can be used to improve the run time performance of exact algorithms in a matheuristics framework. In this paper, we use a pruning heuristic leveraging machine learning as a pre-processing step followed by an exact Integer Programming approach. We apply this approach to sparsify instances of the classical travelling salesman problem. Our approach learns which edges in the underlying graph are unlikely to belong to an optimal solution and removes them, thus sparsifying the graph and significantly reducing the number of decision variables. We use carefully selected features derived from linear programming relaxation, cutting planes exploration, minimum-weight spanning tree heuristics and various other local and statistical analysis of the graph. Our learning approach requires very little training data and is amenable to mathematical analysis. We demonstrate that our approach can reliably prune a large fraction of the variables in TSP instances from TSPLIB/MATILDA ($> 85\%$) while preserving most of the optimal tour edges. Our approach can successfully prune problem instances even if they lie outside the training distribution, resulting in small optimality gaps between the pruned and original problems in most cases. Using our learning technique, we discover novel heuristics for sparsifying TSP instances, that may be of independent interest for variants of the vehicle routing problem.

Keywords: Travelling Salesman Problem · Graph Sparsification · Machine Learning · Linear Programming · Integer Programming

1 Introduction

Owing to the high running time of exact solvers on many instances of NP-hard combinatorial optimisation problems (COPs), there has been a lot of research interest in leveraging machine learning techniques to speed up the computation of

optimisation solutions. In recent years, deep learning techniques have been used as an end-to-end approach (see e.g., [30]) for efficiently solving COPs. However, these approaches generally suffer from (i) limited generalisation to larger size problem instances and limited generalisation from instances of one domain to another domain, (ii) need for increasingly complex architectures to improve generalisability and (iii) inherent black-box nature of deep learning that comes in the way of mathematical analysis [6]. In particular, the lack of interpretability of these models means that (1) we do not know which properties of the input instances are being leveraged by the deep-learning solver and (2) we cannot be sure that the model will still work as new constraints are required, which is typical in industry use-cases.

In contrast to the end-to-end deep learning techniques, there has been recent work (see e.g. [13]) to use machine learning as a component to speed-up or scale-up the exact solvers. In particular, Lauri and Dutta [19] recently proposed a framework to use machine learning as a pre-processing step to sparsify the maximum clique enumeration instances and scale-up the exact algorithms in this way. In this work, we build upon this framework and show that integrating features derived from operations research and approximation algorithms into the learning component for sparsification can result in reliably pruning a large fraction of the variables in the classical Travelling Salesman Problem (TSP). Specifically, we use carefully selected features derived from linear programming relaxation, cutting planes exploration, minimum-weight spanning tree (MST) heuristics and various other local and statistical analysis of the graph to sparsify the TSP instances. With these features, we are able to prune more than 85% of the edges on TSP instances from TSPLIB/MATILDA, while preserving most of the optimal tour edges. Our approach can successfully prune problem instances even if they lie outside the training distribution, resulting in small optimality gaps between the pruned and original problems in most cases. Using our learning technique, we discover novel heuristics for sparsifying TSP instances, that may be of independent interest for variants of the vehicle routing problem.

Overall, our approach consists of using a pruning heuristic leveraging machine learning (ML) as a pre-processing step to sparsify instances of the TSP, followed by an exact Integer Programming (IP) approach. We learn which edges in the underlying graph are unlikely to belong to an optimal TSP tour and remove them, thus sparsifying the graph and significantly reducing the number of decision variables. Our learning approach requires very little training data, which is a crucial requirement for learning techniques dealing with NP-hard problems. The usage of well analysed intuitive features and more interpretable learning models means that our approach is amenable to mathematical analysis. For instance, by inserting the edges from the Christofides and double-tree approximations in our sparsified instances, we can guarantee the same bounds on the optimality gap. We hypothesise that our approach, integrating features derived from operations research and approximation algorithms into a learning component for sparsifi-

cation, is likely to be useful in a range of COPs, including but not restricted to, vehicle routing problems.

Outline. The paper is structured as follows: Section 2 describes the related works. In Section 3 we outline the proposed sparsification scheme: the feature generation, the sparsification model and post-processing techniques. Section 4 contains the experimental setup, exposition on the computational experiments and results. Discussion and conclusions follow in Section 5.

2 Notation and Related Work

Given a graph $G = (V, E; w)$ with a vertex set $V = \{1, ..., n\}$, an edge set $E = \{(u, v)|u, v \in V, u \neq v\}$ and a weighting function $w_G(e) \rightarrow \mathbb{Z}^+$ that assigns a weight to each edge $e \in E$, the goal of the TSP is to find a tour in G that visits each vertex exactly once, starting and finishing at the same vertex, with least cumulative weight. We denote by m the number of edges $|E|$ and by n the number of vertices $|V|$ of the problem.

2.1 Exact, Heuristic and Approximate Approaches

The TSP is one of the most widely-studied COPs and has been for many decades; for this reason, many very effective techniques and solvers have been developed for solving them. Concorde is a well-known, effective exact solver which implements a branch and cut approach to solve a TSP IP [1], and has been used to solve very large problems. An extremely efficient implementation of the Lin-Kernighan heuristic is available at [15], which can find very close-to-optimal solutions in most cases. Approximation algorithms also exist for the metric TSP that permit the identification of solutions, with worst-case performance guarantees, in polynomial time [7, 26]. Many metaheuristic solution frameworks have also been proposed, using the principles of ant colony optimisation, genetic algorithms and simulated annealing among others [5, 9, 16]. In each of these traditional approaches to the TSP, there are lengthy development times and extensive problem-specific knowledge is required. If the constraints of the given problem are altered, the proposed solution method may no longer be satisfactory, possibly requiring further development.

2.2 Learning to Solve Combinatorial Optimisation Problems

Inspired by the success of deep learning to solve natural language processing and computer vision tasks, the question of how effective similar techniques might be in COP solution frameworks arises. Interest in this research direction emerged following the work of Vinyals et al. [30], in which sequence-to-sequence neural networks were used as heuristics for small instances of three different COPs, including the TSP. This was quickly followed by other ML-based approaches [3, 18, 21] that solve larger problem instances and avoid the need for supervised learning where access to data is a bottleneck. Graph neural networks [25] and transformer architectures [29] lead to significant speedups

for learned heuristics, and have been demonstrated to obtain near-optimal solutions to yet larger TSP and vehicle routing problem (VRP) instances in seconds. Although they can produce competitive solutions to relatively small problems, these learning approaches appear to fail to generalise well to larger instance sizes, and most of these approaches require that the instance is Euclidean, encoding the coordinates of the vertices for feature computation. In cases of failure and poor solution quality, however, there is little possibility of interpreting why mistakes were made, making it difficult to rely on these models.

2.3 Graph Sparsification

Graph sparsification is the process of pruning edges from a graph $G = (V, E)$ to form a subgraph $H = (V, E' \subset E)$ such that H preserves or approximates some property of G [4, 11, 23]. Effective sparsification is achieved if $|E'| \ll |E|$. The running time of a TSP solver can be reduced if the underlying complete graph K_n can be sparsified such that the edges of at least one optimal Hamiltonian cycle are preserved. The work of Hougardy and Schroeder [17] sparsifies the graph defining symmetric TSP instances exactly, removing a large fraction of the edges, known as "useless" edges, that provably cannot exist in an optimal tour. Another heuristic approach due to Wang and Remmel [31] sparsifies symmetric instances by making probabilistic arguments about the likelihood that an edge will belong to an optimal tour. Both of these approaches have proven successful, reducing computation time significantly for large instances, but are unlikely to be easy to modify for different problem variants.

Recently, the sparsification problem has been posed as a learning problem, for which a binary classification model is trained to identify edges unlikely to belong to an optimal solution. Grassia et al. [12] use supervised learning to prune edges that are unlikely to belong to maximum cliques in a multi-step sparsification process. This significantly reduces the computational effort required for the task. Sun et al. [28] train a sparsifier for pruning edges from TSP instances that are unlikely to belong to the optimal tour. These approaches have the advantage that they can easily be modified for similar COP variants. The use of simpler, classical ML models lends them the benefits of partial interpretability and quick inference times. However, in the latter case, it is assumed that a very large number of feasible TSP solutions can be sampled efficiently, which does not hold for all routing-type problems, and in neither case are guarantees provided about the quality of the optimal solutions for sparsified problem instances.

3 Sparsification Scheme

The sparsification problem is posed as a binary classification task. Given some edge $e \in E$, we wish to assign it a label 0 or 1, where the label 0 indicates that the associated edge variable should be pruned and the label 1 indicates that it should be retained. We acquire labelled data for a set of graphs $\mathcal{G} = \{G_1, ..., G_n\}$ corresponding to TSP problem instances. For each graph $G_i =$

(V_i, E_i) we compute a set of p_i optimal tours $\mathcal{T}_i = \{t_i^1, ..., t_i^{p_i}\}$, as many as can be found within the tolerance level of the IP solver, and for each edge $e \in E_i$ we compute a feature representation $\vec{q}_e$. Each tour t_i has an implied set of edges $t_i \implies \epsilon_i \subset E_i$. Labelling each $e \in \bigcup_{j=1}^{p_i} \epsilon_i^j = \mathcal{E}_i$ with 1 and each edge $e \in E_i \backslash \mathcal{E}_i$ as 0, we train an edge classifier to prune edges that do not belong to an optimal tour. An optimal sparsifier would prune all but those edges belonging to optimal tours (potentially also solving the TSP). This classifier represents a binary-fixing heuristic in the context of an IP. In the following sections, we describe the feature representation that is computed for each edge and post-processing steps that are taken in order to make feasibility and approximation guarantees.

3.1 Linear Programming Features

We pose the TSP as an IP problem, using the DFJ formulation [8]. Taking A_{ij} as the matrix of edge-weights, we formulate it as follows:

$$\text{minimize} \qquad z = \sum_{i,j=1; i \neq j}^{n} A_{ij} x_{ij} \tag{1}$$

$$\text{subject to} \qquad \sum_{i; i \neq j}^{n} x_{ij} = 1, \qquad j = 1, ..., n \tag{2}$$

$$\sum_{j; j \neq i}^{n} x_{ij} = 1, \qquad i = 1, ..., n \tag{3}$$

$$\sum_{(i,j) \in W} x_{ij} \leqslant |W| - 1, \qquad W \subseteq V; \quad |W| \geqslant 3 \tag{4}$$

$$x_{ij} \in \{0, 1\}, \qquad i, j = 1, ..., m; \quad i \neq j \tag{5}$$

Useful information about the structure of a TSP problem can be extracted by inspecting solution vectors to linear relaxations of this IP; in this way we can obtain insights into the candidacy of edges for the optimal tour. In fact, in several cases, for the MATILDA problem set, the solution to the linear relaxation at the root of the Branch and Bound (B&B) tree is itself an optimal solution to the TSP.

We denote the solution to the linear relaxation z_{LP} of the integer programme at the root node of the B&B tree by $\hat{\vec{x}}^0$. At this point, no variables have been branched on and no subtour elimination cuts have been introduced. That is, the constraints (4) are dropped and the constraints (5) are relaxed as:

$$x_{ij} \in [0, 1], \qquad i, j \in \{1, ..., m\}, i \neq j. \tag{6}$$

One can strengthen this relaxation by introducing some subtour elimination constraints (4) at the root node. In this case, the problem to be solved remains a linear programming problem but several rounds of subtour elimination cuts

are added. One can limit the computational effort expended in this regard by restricting the number of constraint-adding rounds with some upper bound $k = \lceil \log_2(m) \rceil$. The solution vector for this problem after k rounds of cuts is denoted by $\tilde{\vec{x}}^k$. We can also use as features the associated reduced costs $\vec{r}^k$ of the decision variables, which are computed in the process of a Simplex solver, standardising their values as $\hat{\vec{r}}^k = \vec{r}^k / \max \vec{r}^k$.

In order to capture broader information about the structure of the problem, stochasticity is introduced to the cutting planes approach. Inspired by the work of Fischetti and Monaci [10], the objective of the problem is perturbed. Solving the perturbed problem results in different solution vectors, which can help us to explore the solution space. We solve the initial relaxation z_{LP} in order to obtain a feasible solution, which can sometimes take a significant amount of computing effort. Subtour-elimination constraints are added to the problem for each subtour in the initial relaxation. Following this, $k = \lceil \log_2(m) \rceil$ copies of this problem are initialised. For each new problem, the edge weights A_{ij} are perturbed, and the problem is re-solved. The normalised reduced costs are obtained from each perturbed problem and for each edge (i, j) the mean reduced cost $\tilde{r}_{ij}$ is computed. We use the vector $\tilde{\vec{r}}$ of such values as a feature vector[3].

3.2 Minimum Weight Spanning Tree Features

The MST provides the basis for the Christofides–Serdyukov [7, 26] and double-tree approximation algorithms, which give feasible solutions with optimality guarantees for a symmetric, metric TSP. Taking inspiration from these approximation algorithms, we use multiple MSTs to extract edges from the underlying graph thereby allowing us to compute features using them.

Algorithm 1 MST Features

Input: $G = (V, E), j$

$H \leftarrow (V, E'' = \varnothing)$
for $j \in \{1, 2, ..., \lceil \log n \rceil\}$ **do**
  $T = (V, E' \subset E) \leftarrow \text{MST}(G)$
  $G = (V, E) \leftarrow (V, E \backslash E')$
  $H = (V, E'') \leftarrow (V, E'' \cup E')$
  $w_H(e) = 1/j \quad \forall e \in E'$

Output: $H = (V, E'')$

First, a new graph $H = (V, \varnothing)$ is initialised with the vertex set but not the edge set of the complete graph $G = (V, E) = K_n$. For $j = \lceil \log_2(n) \rceil \ll n$ iterations the MST $T = (V, E')$ of G is computed and the edges E' are removed from E and added to the edge set of H. Then, at each step, $G(V, E) \leftarrow G(V, E \backslash E')$, giving a new MST at each iteration with unique edges. The iteration at which edges are added to the graph H is stored, so that a feature $\hat{q}^j_{il} = 1^p_{il}/p$ may be computed, where 1^p_{il} is the indicator, taking unit value if the edge $e = (i, l) \in E$ was extracted at iteration p and zero otherwise.

Since the value of j should be small, the vast majority of the edges will have zero-valued feature-values. This edge transferal mechanism can be used as a

[3] For additional detail regarding computation of these features, please see: `https://arxiv.org/abs/2104.09345`.

sparsification method itself: the original graph can be pruned such that the only remaining weighted edges are those that were identified by the successive MSTs, with the resulting graph containing $j(n-1)$ edges.

3.3 Local Features

The work of Sun et al. [28] constructs four local features on the graph $G = (E, V)$, comparing weights of an edge $(i, j) \in E$ to the edge weights $(k, j), k \in V$ and $(i, k), k \in V$. While relatively inexpensive to compute, yet less expensive features can be computed, comparing a given edge weight (i, j) to the maximum and minimum weights in E. That is, for each $(i, j) \in E$ we compute a set of features q_{ij} as:

$$q^a_{ij} = (1 + A_{ij})/(1 + \max_{(l,k)\in E} A_{lk}) \tag{7}$$

$$q^b_{ij} = (1 + A_{ij})/(1 + \max_{l\in V} A_{il}) \tag{8}$$

$$q^c_{ij} = (1 + A_{ij})/(1 + \max_{l\in V} A_{lj}) \tag{9}$$

$$q^d_{ij} = (1 + \min_{(l,k)\in E} A_{lk})/(1 + A_{ij}) \tag{10}$$

$$q^e_{ij} = (1 + \min_{l\in V} A_{il})/(1 + A_{ij}) \tag{11}$$

$$q^f_{ij} = (1 + \min_{l\in V} A_{lj})/(1 + A_{ij}) \tag{12}$$

The motivation for the features (7) and (10) is to cheaply compute features that relate a given edge weight to the weights of the entire graph in a global manner. On the other hand, motivated by the work of Sun et al. [28], the features (8), (9), (11), (12), compare a given edge weight to those in its direct neighbourhood; the edge weight associated with edge (i, j) is related only to the weights of the associated vertices i and j.

3.4 Postprocessing Pruned TSP Graphs

In this setting, sparsification is posed as a set of m independent classification problems. The result of this is that there is no guarantee that any feasible solution exists within a pruned problem instance. Indeed, even checking that any tour exists within a sparsified graph is itself an NP-hard problem. One can guarantee feasibility of the pruned graph by ensuring that the edges belonging to some known solution exist in the pruned graph; this forces both connectivity and Hamiltonicity (see Figure 1). The pruned graph has at least one feasible solution and admits an optimal objective no worse than that of the solution that is known. For the TSP we can construct feasible tours trivially by providing any permutation of the vertices, but it is likely that such tours will be far from optimal. Assuming the problem is metric, we can use an approximation algorithm to construct a feasible solution to the problem that also gives a bound on the quality of the solution.

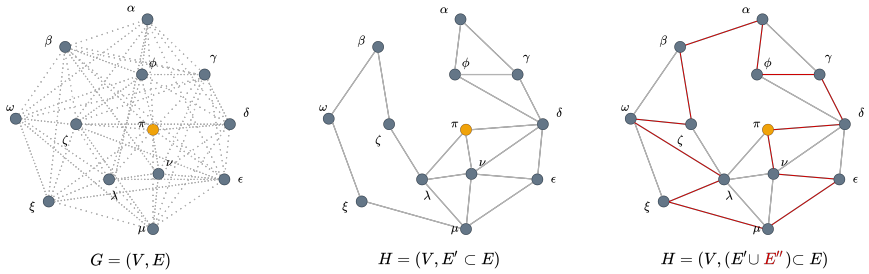

Fig. 1: The pruned graph H does not admit a feasible solution. *Given a known solution, we add the edges E'' of the solution to the pruned graph in order to guarantee the feasibility of the pruned instance. If we obtain E'' using an approximation algorithm, then we can make guarantees about the quality of the solutions obtained from H.*

4 Experiments and Results

All experiments were carried out in Python[4]. Graph operations were performed using the NetworkX package [14] and the training was carried out with the Scikit-Learn package [22]. The linear programming features were computed using the Python interfaces for the Xpress and SCIP optimisation suites [2, 20]. Training and feature computation was performed on a Dell laptop running Ubuntu 18.04 with 15. 6 GB of RAM an, Intel® Core™ i7-9750H 2.60GHz CPU and an Nvidia GeForce RTX 2060/PCIe/SSE2 GPU.

4.1 Learning to Sparsify

First we train a classification model to prune edges that are unlikely to belong to an optimal tour. That is, given the feature representation

$$\vec{q}_{il} = [q^a_{il}, q^b_{il}, q^c_{il}, q^d_{il}, q^e_{il}, q^f_{il}, \hat{\vec{r}}^k_{il}, \tilde{\vec{r}}_{il}, \hat{q}^j_{il}]^T$$

for the edge (i, l), we aim to train a machine learning model that can classify all the edges of a given problem instance in this manner. In each case we let the parameter $k = \lceil \log_2(m) \rceil$ and $j = \lceil \log_2(n) \rceil$, since these numbers grow slowly with the size of the graph and prevent excessive computation. In order to address the effects of class imbalance, we randomly under-sample the negative class such that the classes are equally balanced, and adopt class weights $\{0.01, 0.99\}$ for the negative and positive classes respectively to favour low false negative rates for the positive class, favouring correctness over the sparsification rate. Sample weights $w_{il} = A_{il} / \max_{il} A_{il}$ are applied to each sample associated with an edge (i, l) to encourage the classifier to reduce errors associated with longer edges.

[4] All code available at: `https://github.com/JamesFitzpatrickMLLabs/optlearn`

In many instances there exists more than one optimal solution. We compute all optimal solutions within the tolerance level of the solver (1×10^{-8}) and assign unit value to the label for edges that belong to any optimal solution. Any optimal solution gives an upper bound on the solution for any subsequent attempts to solve the problem. To compute each solution, one can introduce a tour elimination constraint to the problem formulation to prevent previous solutions from being feasible. The problem can then be re-solved multiple times to obtain more solutions to the problem, until no solution can be found that has the same objective value as that of the original optimal tour.

Training was carried out on problem instances of the MATILDA problem set [27], since they are small ($n = 100$ for each) and permit relatively cheap labelling operations. One third (63) of the CLKhard and LKCChard problem instances were chosen for the training set, while the remaining problem instances are retained for the testing and validation sets. These two problem categories were selected for training once it was identified through experimentation that sparsifiers trained using these problems generally performed better (with fewer infeasibilities). This is in line with the findings of Sun et al. [28]. Since each of the linear programming features must be computed for any given problem instance, it is worthwhile checking if any solution $\hat{\vec{x}}^0$ or $\tilde{\vec{x}}^k$ is an optimal TSP solution, which would allow all computation to terminate at this point. All of the edges of a given graph belong to exactly one of training, test or validation sets, and they must all belong to the same set. Each symmetric problem instance in the TSPLIB problem set [24] for which the order $n \leqslant 3038$ is retained for evaluation of the learned sparsifier only.

Following pruning, edges of a known tour are inserted, if they are not already elements of the edge set of the sparsified graph. In this work, we compute both the double-tree and Christofides approximations , since they can be constructed in polynomial time and guarantee that the pruned graph will give an optimality ratio $\hat{\ell}_{opt}/\ell_{opt} \leqslant 3/2$, where $\hat{\ell}_{opt}$ is the optimal tour length for a pruned problem and ℓ_{opt} is the optimal tour length for the original problem. The performance of the classifier is evaluated using the optimality ratio $\tilde{\ell} = \hat{\ell}_{opt}/\ell_{opt}$ and the retention rate $\tilde{m} = \hat{m}/m$, where $\hat{m}$ is the number of edges belonging to the pruned problem instance. Since the pruning rate $1 - \tilde{m}$ implies the same result as the retention rate, we discuss them interchangeably. Introducing any known tour guarantees feasibility at a cost in the sparsification rate no worse than $(\hat{m} + n)/\hat{m}$.

4.2 Performance on MATILDA Instances

The sparsification scheme was first evaluated on the MATILDA problem instances, which are separated into seven classes, each with 190 problem instances. Logistic regression, random forest and linear support vector classifiers were evaluated as classification models. Although no major advantage was displayed after a grid search over these models, here, for the sake of comparison with Sun et al.,

an L1-regularised SVM with an RBF kernel is trained as the sparsifier. We can see from Table 1 that the optimality ratios observed in the pruned problems are comparable to those of Sun et al, with a small deterioration in the optimality gap for the more difficult problems (around 0.2%) when the same problem types were used for training. However, this comes with much greater sparsification rates, which leads to more than 85% of the edges being sparsified in all cases and almost 90% on average, as opposed to the around 78% observed by Sun et al. We similarly observe that the more difficult problems are pruned to a slightly smaller extent than the easier problem instances.

	Problem Class						
Statistic (Mean)	CLKeasy	CLKhard	LKCCeasy	LKCChard	easyCLK-hardLKCC	hardCLK-easyLKCC	random
$\tilde{l}$	1.00000	1.00176	1.00000	1.00385	1.00049	1.00035	1.00032
$1 - \tilde{m}$	0.90389	0.89021	0.91086	0.88298	0.88874	0.90357	0.89444

Table 1: Evaluation of the trained sparsifier against the problem subsets of the MATILDA instances. Here each cell indicates the mean value of the optimality ratio or pruning rate over each subset of problems (not including training instances)

4.3 Pruning With and Without Guarantees

This section describes experiments carried out to test how much the sparsifier would need to rely on inserted tours to produce feasibly reduced problem instances. The optimality ratio statistics are shown in Table 2. Before post-processing, pruning was achieved with a maximum rate of edge retention $\max\{\hat{m}/m\} = 0.139$, a minimum rate $\min\{\hat{m}/m\} = 0.075$ and a mean rate $\langle\hat{m}/m\rangle = 0.100$ among all problem instances. In the vast majority of cases, at least one optimal tour is contained within the pruned instance (column 2), or one that is within 2% of optimal (column 4). For the more difficult problem instances, there were several infeasible instances produced each time. In some cases the pruned instances admitting sub-optimal solutions with respect to the original problem had improved optimality ratios (columns 3 and 4). The worst-case optimality improves for each problem class (except CLKeasy, where all problems were sparsified without losing an optimal solution) after post-processing. The distribution of these values with post-processing is depicted in Figure 2.
This sparsification scheme was also tested on the TSPLIB problem instances (see Table 3). Before post-processing, for the majority of the problem instances, an optimal tour is contained within the sparsified graphs (column 2), and the vast majority of the problem instances have optimality ratios no worse than $\tilde{\ell} = 1.05$ (column 4). Unlike with the MATILDA instances, the pruning rate varies significantly. This is in accordance with the findings of Sun et al. [28] and indicates that the sparsifier is less certain about predictions made, and fewer edges are therefore removed in many cases. For some smaller problem instances (smaller than the training set problems) the pruning rate approaches 0.5167.

	# of Problems With Below Condition True			Worst Case
Problem Class	$\hat{\ell}_{opt}/\ell_{opt} = 1.0$	$\hat{\ell}_{opt}/\ell_{opt} < 1.01$	$\hat{\ell}_{opt}/\ell_{opt} < 1.02$	$\max\{\hat{\ell}_{opt}/\ell_{opt}\}$
CLKeasy	190 → 190	190 → 190	190 → 190	1 → 1
CLKhard	110 → 123	153 → 181	165 → 190	∞ → 1.018
LKCCeasy	189 → 190	189 → 190	190 → 190	1.016 → 1
LKCChard	35 → 65	82 → 168	104 → 185	∞ → 1.024
easyCLK-hardLKCC	172 → 181	180 → 186	180 → 188	∞ → 1.024
hardCLK-easyLKCC	167 → 181	175 → 188	176 → 189	∞ → 1.030
random	174 → 176	186 → 188	188 → 190	∞ → 1.016
Total	1037 → 1106	1155 → 1291	1193 → 1322	∞ → 1.030

Table 2: Optimality ratio statistics before and after approximate tour insertion following learned sparsification (MATILDA problem instances). *Each cell indicates the number of problem instances of each class for which the optimality ratio resided within the bounds stated before and after post-processing. For example, the cell in column 2 containing* 174 → 176 *indicates that* 174 *purely sparsified instances admitted unit optimality ratios, but* 176 *did so after the post-processing. The last column contains the maximum optimality gap for each class. If there are infeasibly sparsified graphs, this value is denoted by* ∞.

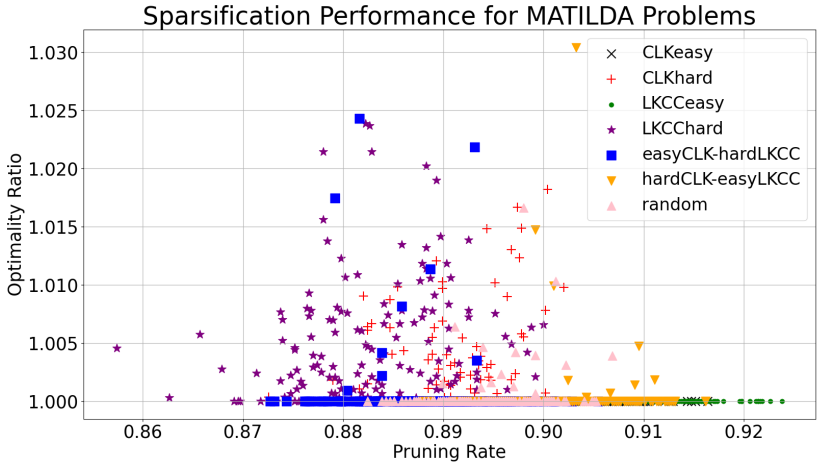

Fig. 2: Each point represents a problem instance, showing the pruning rate and the optimality ratio achieved for each problem in the MATILDA benchmark set.

The median pruning rate was 0.9109, with the mean pruning rate at 0.8904 and a standard deviation of 0.086. The highest pruning rate was for the problem *u2152.tsp*, for which $0.9758m$ edges were removed, achieving an optimality ratio of 1.0.

Introducing the approximate tours brought the worst-case optimality ratio to 1.01186 (column 5) for *d657.tsp*, with median and mean pruning rates of, respectively, 0.8991 and 0.8718. The mean optimality ratio (excluding the infeasi-

ble instances) before approximate tour insertion was 1.01546 and (also without these same problems) 1.00073 after insertion, indicating that in most cases there is little reduction in solution quality as a result of sparsification. In Figure 3 we can see depicted the relationship between the problem size (the order, n) and the pruning rate. Almost all of the instances retain optimal solutions after pruning, those that don't are indicated by the colour scale of the points. We can see from this plot that the drilling problem *d657.tsp* and problems *pr*[144, 226, 266]*.tsp* *ts225.tsp* have some of the largest optimality deviations (up to 1.2%), likely because in these problems the vertices are aligned in regular, often clustered patterns, which leads to problems with edge-weight distributions much different from the problems of the training set.

	# of Problems With Below Condition True			Worst Case
Problem Class	$\hat{\ell}_{opt}/\ell_{opt} = 1.0$	$\hat{\ell}_{opt}/\ell_{opt} < 1.005$	$\hat{\ell}_{opt}/\ell_{opt} < 1.010$	$\max\{\hat{\ell}_{opt}/\ell_{opt}\}$
TSPLIB	$55 \rightarrow 56$	$69 \rightarrow 70$	$72 \rightarrow 73$	$\infty \rightarrow 1.01186$

Table 3: Optimality ratio statistics before and after approximate tour insertion following learned sparsification (TSPLIB problem instances). *Results for each symmetric problem instance with* $n \leqslant 1000$ *using a logistic regression sparsifier. This set contained* 76 *problem instances, some of which are not metric TSPs. Although we cannot make provable guarantees for the non-metric problem instances, we can still use the approximate tours to ensure feasibility.*

.

4.4 Minimum Weight Spanning Tree Pruning

Empirical experiments demonstrated that using only successive MSTs, as outlined in Section 3.2, one can effectively sparsify symmetric TSP instances. This scheme proceeds without the need for a classifier, by building a new graph H with only the edges of the MSTs and their associated edge weights in G. This sparsified instance achieves a retention rate of jn/m, where j is the number of trees computed. In many cases, simply using this as a scheme for selecting edges for retention was sufficient to sparsify the graph while achieving a unit optimality ratio.

On the MATILDA problems, again with $j = \lceil \log_2(n) \rceil$, this achieves a worst-case optimality ratio of 1.0713 and a uniform pruning rate of 0.86, with the majority of the pruned instances containing an optimal solution. Under these conditions the easier problems (CLKeasy and LKCCeasy) had optimal tours preserved in every problem instance except for one. For the more difficult problem instances, in the vast majority of cases, the ratio does not exceed 1.02 (see Table 4). Insertion of approximate tours in this scenario does not lead to much improvement, since every instance of the MATILDA problem set is sparsified feasibly in this scheme. For the TSPLIB problems, all but two can be sparsified feasibly without the insertion of approximate edges: *p654.tsp* and *fl417.tsp*. Since k is a function of the problem order, $n = 100$, 86% of the edges of a graph will be removed, whereas at order $n = 1000$, the pruning rate reaches 98%. The worst optimality ratio for any

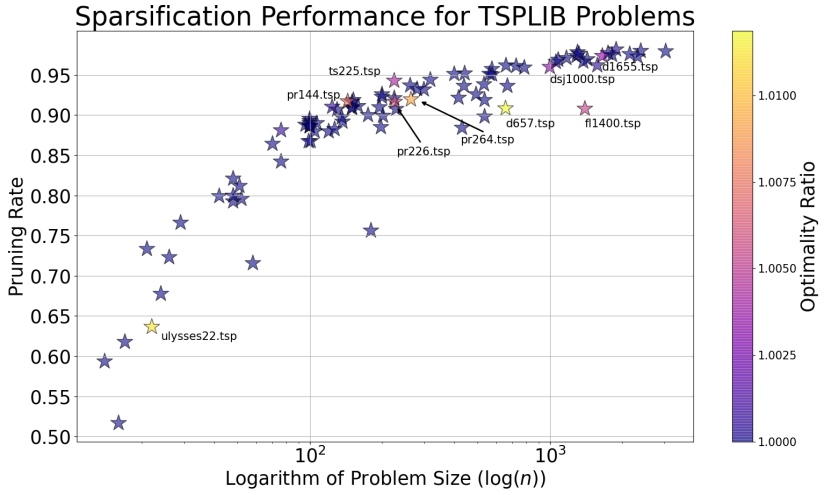

Fig. 3: Here each point represents a problem instance. The horizontal axis depicts the logarithm of the problem size in terms of the number of nodes. The vertical axis shows the sparsification rate and the colouring of the points indicates the optimality ratio observed.

	# of Problems With Below Condition True			Worst Case
Problem Class	$\hat{\ell}_{opt}/\ell_{opt} = 1.0$	$\hat{\ell}_{opt}/\ell_{opt} < 1.02$	$\hat{\ell}_{opt}/\ell_{opt} < 1.05$	$\max\{\hat{\ell}_{opt}/\ell_{opt}\}$
CLKeasy	190 → 190	190 → 190	190 → 190	1 → 1
CLKhard	85 → 88	185 → 185	190 → 190	1.029 → 1.029
LKCCeasy	189 → 190	190 → 190	190 → 190	1.002 → 1.002
LKCChard	77 → 78	183 → 183	189 → 189	1.071 → 1.054
easyCLK-hardLKCC	173 → 190	190 → 190	190 → 190	1.012 → 1.012
hardCLK-easyLKCC	171 → 173	190 → 190	190 → 190	1.015 → 1.015
random	176 → 176	190 → 190	190 → 190	1.006 → 1.006
All	1061 → 1330	1318 → 1330	1329 → 1330	1.071 → 1.054

Table 4: Optimality ratio statistics before and after approximate tour insertion following MST sparsification (MATILDA problem instances). *Comparison between the pure multiple MST pruning scheme and the same scheme with double-tree post-processing. In every case for the MATILDA problem set, the pruned instances are feasible.*

.

feasible problem was for *pr226.tsp*, at $\tilde{l} = 1.106$, with a retention rate of 0.071. The majority of the sparsified instances retained an optimal tour (see Table 5), with the mean optimality ratio $\langle\tilde{\ell}\rangle$ for all feasible problems taking the value 1.0061. Including approximate tour edges in the sparsified graphs results in none having an optimality gap greater than 1.084 (*pr654.tsp*, which was previously infeasible, with a retention rate of 3.1%). The other previously infeasibly-sparsified

problem, *fl417.tsp* admits an optimality ratio $\tilde{\ell} = 1.077$, retaining just 4.3% of its edges. The mean optimality ratio for all problems emerged as 1.0053.

	# of Problems With Below Condition True			Worst Case
Problem Class	$\hat{\ell}_{opt}/\ell_{opt} = 1.0$	$\hat{\ell}_{opt}/\ell_{opt} < 1.02$	$\hat{\ell}_{opt}/\ell_{opt} < 1.05$	$\max\{\hat{\ell}_{opt}/\ell_{opt}\}$
TSPLIB	$51 \rightarrow 51$	$66 \rightarrow 67$	$71 \rightarrow 71$	$\infty \rightarrow 1.084$

Table 5: Optimality ratio statistics before and after approximate tour insertion following MST sparsification (TSPLIB problem instances). *For each problem instance the number of trees j differed, according to the order n.*

4.5 Specifying the Pruning Rate

One of the advantages of including the post-processing step that guarantees the feasibility of the pruned instance is that we can exert greater control over the pruning rate. Training a sparsifier in the manner outlined above necessitates a trade-off between the optimality ratio and the pruning rate. Higher pruning rates result in sparsified problems that are easier to solve but that have typically poorer optimality ratios. Indeed, if the sparsification rate is too high, many of the sparsified instances will also become infeasible. Including approximate tours, however, allows us to choose effectively any decision threshold for the classifier, ranging from total sparsification and removing all edges, to choosing a threshold that results in no sparsification at all and the retention of all edges from the original graph. So long as there is at least one known feasible tour, regardless of its quality, then the pruned instance will be feasible after post-processing.

5 Discussion and Conclusions

In this work we have demonstrated that it is possible to learn to effectively sparsify TSP instances, pruning the majority of the edges using a small classification model while relying on LP and graph-theoretic features that can be efficiently computed. Providing optimality guarantees is possible by inserting edges of approximate tours, ensuring that even out-of-distribution problem instances can be tackled with this scheme. These features are supplemented with local statistical features, comparing edge weights to the global and neighbouring distributions. This scheme successfully generalises to larger problem instances outside of the training distribution. Where there is a lack of training data or where expediency is favoured, it has been shown that the MST extraction mechanism with inserted approximate tours performs well on most problem instances.

The motivation of this work has been to use the methods of ML to aid in the design of heuristics for solving combinatorial optimisation problems. This is in the hope that such an approach can reduce the development time required. Development time for ML solutions depends typically on the engineering of either features or problem-specific models and architectures. This can effectively

transfer the the engineering effort from one task to another, without producing tangible benefits. In this approach, classical ML models are used, which means that feature design is paramount to the success of the model. Here, LP features are designed that are not dependent upon the formulation of the problem; any TSP problem that requires formulation as binary IP with subtour-elimination (or capacity) constraints can have features produced in the same manner. This is attractive because it simply requires knowledge of IP formulations that give tight relaxations. Such relaxations may also be sufficient to solve the problem, potentially obviating the need for any further computation. Optimality ratio performance does not appear to depend on the problem size, which indicates that this pre-processing scheme might help extend the applicability of learned solving heuristics. Some preliminary experimental results indicated that this approach can be leveraged to speed up solving times for a custom-built Branch and Cut solver for the TSP using the DFJ formulation, but further experimentation and implementation in C will be necessary to determine the extent to which speedups can be achieved.

The use of the approximation algorithms guarantees the feasibility of the pruned problem instance, but it does not provide a tight bound on the optimality ratio. Tighter bounds can be achieved by using these in combination with effective heuristics. Alternatively, one could compute successive approximations by removing from consideration edges belonging to a tour once they have been computed, analogously to Algorithm 1. The observations from experimentation indicate that while these approximate tours rarely improve the optimality ratio at lower pruning rates, they are not only necessary for feasibility at higher pruning rates, but can help immensely to improve the final optimality ratio.

This scheme has the potential to be developed in a similar manner for other routing problems, in particular vehicle routing problems, for which solvers are not as effective in practice as they are for the TSP. To realise the benefits of scheme, an implementation would also have to be rewritten in a lower level language. Subsequent work could be done to evaluate the smallest problem sizes for which training can be carried out effectively and applicability to the VRP, where it might be possible to extend the use of the vehicle-flow IP formulation to larger problems, which typically need to be solved using a Branch, Cut and Price approach.

Acknowledgements

This publication has emanated from research supported in part by a grant from Science Foundation Ireland under Grant number 18/CRT/6183. For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

References

1. David L Applegate, Robert E Bixby, Vašek Chvátal, William Cook, Daniel G Espinoza, Marcos Goycoolea, and Keld Helsgaun. Certification of an optimal tsp

tour through 85,900 cities. *Operations Research Letters*, 37(1):11–15, 2009.
2. Robert Ashford. Mixed integer programming: A historical perspective with xpress-mp. *Annals of Operations Research*, 149(1):5, 2007.
3. Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.
4. András A Benczúr. Approximate st min-cuts in o (nˆ 2) time. In *Proc. 28th ACM Symposium on Theory of Computing, 1996*, 1996.
5. Heinrich Braun. On solving travelling salesman problems by genetic algorithms. In *International Conference on Parallel Problem Solving from Nature*, pages 129–133. Springer, 1990.
6. G. A. Di Caro. A survey of machine learning for combinatorial optimization. In *30th European Conference on Operations Research (EURO)*, 2019.
7. Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.
8. George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.
9. Marco Dorigo and Luca Maria Gambardella. Ant colonies for the travelling salesman problem. *biosystems*, 43(2):73–81, 1997.
10. Matteo Fischetti and Michele Monaci. Exploiting erraticism in search. *Operations Research*, 62(1):114–122, 2014.
11. Wai-Shing Fung, Ramesh Hariharan, Nicholas JA Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. *SIAM Journal on Computing*, 48(4):1196–1223, 2019.
12. Marco Grassia, Juho Lauri, Sourav Dutta, and Deepak Ajwani. Learning multi-stage sparsification for maximum clique enumeration. *arXiv preprint arXiv:1910.00517*, 2019.
13. Prateek Gupta, Maxime Gasse, Elias B. Khalil, Pawan Kumar Mudigonda, Andrea Lodi, and Yoshua Bengio. Hybrid models for learning to branch. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.
14. Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.
15. Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.
16. John J Hopfield and David W Tank. "neural" computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152, 1985.
17. Stefan Hougardy and Rasmus T Schroeder. Edge elimination in tsp instances. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 275–286. Springer, 2014.
18. Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.
19. Juho Lauri and Sourav Dutta. Fine-grained search space classification for hard enumeration variants of subset problems. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of*

Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019, pages 2314–2321. AAAI Press, 2019.
20. Stephen Maher, Matthias Miltenberger, Joao Pedro Pedroso, Daniel Rehfeldt, Robert Schwarz, and Felipe Serrano. Pyscipopt: Mathematical programming in python with the scip optimization suite. In *International Congress on Mathematical Software*, pages 301–307. Springer, 2016.
21. Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, pages 9839–9849, 2018.
22. Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.
23. David Peleg and Alejandro A Schäffer. Graph spanners. *Journal of graph theory*, 13(1):99–116, 1989.
24. Gerhard Reinelt. Tsplib—a traveling salesman problem library. *INFORMS Journal on Computing*, 3(4):376–384, 1991.
25. Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.
26. A.I. Serdyukov. On some extremal walks in graphs. *Upravlyaemye Sistemy 17: 76–79*, (17):76–79, 1978.
27. Kate Smith-Miles, Jano van Hemert, and Xin Yu Lim. Understanding tsp difficulty by learning from evolved instances. In *International Conference on Learning and Intelligent Optimization*, pages 266–280. Springer, 2010.
28. Yuan Sun, Andreas Ernst, Xiaodong Li, and Jake Weiner. Generalization of machine learning for problem reduction: a case study on travelling salesman problems. *OR Spectrum*, pages 1–27, 2020.
29. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.
30. Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in neural information processing systems*, pages 2692–2700, 2015.
31. Yong Wang and Jeffrey Remmel. A method to compute the sparse graphs for traveling salesman problem based on frequency quadrilaterals. In *International workshop on frontiers in algorithmics*, pages 286–299. Springer, 2018.