Application of Machine Learning to Algorithm Selection for TSP

Josef Pihera
Vienna PhD School of Informatics
Vienna University of Technology
pihera@dbai.tuwien.ac.at

Nysret Musliu
Institute of Information Systems
Vienna University of Technology
musliu@dbai.tuwien.ac.at

***Abstract*—The Travelling Salesman Problem (TSP) has been extensively studied in the literature and various solvers are available. However, none of the state-of-the-art solvers for TSP outperforms the others in all problem instances within a given time limit. Therefore, the prediction of the best performing algorithm can save computational resources and optimise the results. In this paper, the TSP is studied in context of automated algorithm selection. Our aim is to identify the relevant features of problem instances and tackle this scenario as a machine learning task. We extend the set of existing features in the literature and propose several novel features to better characterise the problem. The contribution of the new features is statistically analysed and experiments show that adding our new features improves the prediction accuracy. We identified that our features based on kNN graph transformation are especially helpful.**

To create the training datasets, two state-of-the-art (meta-)heuristic algorithms are systematically evaluated on more than 2000 problems. Overall, we show that our prediction can be substantially more accurate than simple preference of an algorithm with the best performance for a majority of problem instances.

***Index Terms*—TSP; algorithm selection; features; machine learning**

I. Introduction

TSP is a famous NP-hard problem with many important applications such as logistics, genome sequencing and telescope aiming [1]. Hence, this problem has been extensively investigated and researchers have been proposing different state-of-the-art solution techniques. Such methods include the prominent exact solver Concorde [1], the heuristic solver LKH [2] based on Lin-Kernighan [3] heuristic, the meta-heuristic solver MAOS [4] etc. Currently, there is no best algorithm for the TSP which dominates the other algorithms in all classes of instances, if the runtime is limited.

An instance of the Travelling Salesman Problem is given as a graph $G = (V, E)$ with edge costs $c : E \mapsto \mathbf{R}$. The problem is to find a cyclic tour T which visits each vertex $v \in V$ exactly once and which minimizes the total cost $\sum c(e)_{e \in T}$. For so-called Euclidean TSP instances, the coordinates of vertices V are specified in a plane, edges connect each pair of vertices and edge cost c is equal to Euclidean distance between corresponding vertices.

In this paper we deal with the following question: given an instance of TSP and a runtime limit, which method should be used to solve that instance? In other words, we want to predict which of the given algorithms will find the best solution. To this aim, we carefully analyse the characteristics of TSP and exploit machine learning techniques to make the prediction based on extracted features of a problem instance.

The main contributions of this paper can be summarized as follows:

- We analyse the existing features (provided by the literature) for TSP and propose several categories of new features to better characterise this problem. We use a kNN graph transformation which allows us to introduce a whole new category of features (note, that kNN graph transformation and k-NN algorithm/classifier are two different things). We propose also new local search probing and geometric features. Some of these features are derived directly from the graph representation of the problem instance and could be generalised for other problems beside TSP. Experiments in this paper show that our new features based on kNN transformation and on local search probing are especially helpful. Features related to tour segments are also beneficial (see Section III).
- We create new training data sets for TSP. For this purpose, we evaluate the performance of two state-of-the-art (meta)heuristic algorithms (LKH and MAOS) on 2163 instances with average size of more than 2700 vertices. The run time data were compiled together with the computed features into the new training datasets.
- We investigate various machine learning algorithms for prediction of the performance of algorithms on new instances. To this aim, we assessed the classifiers with different parameter settings and present a comparison of three feature sets and of five classifiers based on statistical tests. Our results show that prediction of the best performing algorithm is significantly more accurate than simple selection of an algorithm which performs best in majority of cases.

A. Related work

The research most closely related to ours is the one of Hutter *el al.* [5]. They described a method for prediction of runtime of Concorde and LKH on problem instances with fewer than 1000 vertices on average. In our scenario with a time constraint, Concorde can not be used as it does not yield an interim solution for larger problems in a short time. Moreover,

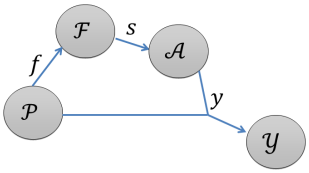

Fig. 1. Algorithm selection framework

we predict the best performing algorithm at given time point instead of the prediction of runtime. Kanda *et al.* [6] used a set of simple meta-heuristic algorithms and evaluated them on instances with fewer than 100 vertices. Their instances are defined on general graphs (both symmetric and asymmetric), whereas we focus on instances in Euclidean plane. Smith-Miles and Hemert [7] consider two variants of Lin-Kernighan heuristic where one of the heuristics addresses specifically the instances with clustered nodes. They studied which instances are hard and which are easy for the two algorithms. Therefore, they artificially created instances with 100 nodes which were intentionally easy or hard and investigated their features. Mersmann *et al.* [8] also used instances with up to 100 vertices but only a particular search operator was studied. Algorithm selection has been used also for different other combinatorial problems like for example graph coloring [9], [10] etc. For more information we refer the reader to the following surveys [11], [12].

Compared to the literature, our problem scenario includes a prediction of a relative performance of algorithms at any time point. We use two state-of-the-art solvers which yield interim solutions (i.e. anytime algorithms) and consider larger instances (with more than 2700 vertices in average) than in aforementioned works. Moreover, we invent new features which help us to better characterise the problem instances in our scenario.

This paper is organised as follows. In Section 2, the background information about algorithm selection is presented. Section 3 focuses on features used in algorithm selection for TSP; the new features are discussed in detail. Some details about our benchmarking problems, experiment objectives and processing methods are covered in Section 4. Experiments and the respective evaluation are given in Section 5. Finally, our conclusions are presented in Section 6.

II. Algorithm selection

An algorithm selection problem was already formulated by Rice [13] in 1976. We will briefly describe the algorithm selection framework and follow the description given by Smith-Miles [11]. We are given a problem domain $\mathcal{P}$, a set of available algorithms $\mathcal{A}$, a performance space $\mathcal{Y}$ and a feature space $\mathcal{F}$. Performance $y(A, P)$ of algorithm $A \in \mathcal{A}$ on problem $P \in \mathcal{P}$ is given by mapping $y : \mathcal{P} \times \mathcal{A} \mapsto \mathcal{Y}$. Features $f(P)$ characterizing the problem P are given by mapping $f : \mathcal{P} \mapsto \mathcal{F}$. The task is to find the mapping $s : \mathcal{F} \mapsto \mathcal{A}$ which for each problem $P \in \mathcal{P}$ selects an algorithm $A_P = s(f(P))$ such that the performance $y(A_P, P)$ is maximised. This framework is presented in Figure 1.

It is necessary to find and define relevant features $\mathcal{F}$ that characterise the problem well and to construct a good mapping s in order to apply algorithm selection successfully [13]. Also the set of training problem instances, according to which s is constructed, must be representative for the problem. The training instances must not bias the mapping s towards a special class of inputs only.

A. Algorithms

Our set of algorithms $\mathcal{A}$ comprises two state-of-the-art (meta-)heuristic algorithms: MAOS[1] and LKH[2]. We didn't include the popular exact solver Concorde [1] as it is not an anytime algorithm which is a necessary property for our task.

LKH is a solver based on Lin-Kernighan heurstic [3]. It is currently the most successful algorithm in the World TSP challenge [14]. Its search is restricted to candidate sets which, in simple words, focus on the closer vertices instead of the distant ones. This motivates us to apply the kNN transformation (see Section III-3) and examine the resulting graph.

MAOS is a multi-agent based solver which does not contain any explicit local search heuristic. Instead, it uses limited declarative and procedural knowledge [4]. Its performance was found to be competitive with LKH during our preliminary tests. We seek to exploit the differences between these two solvers.

III. Features for TSP

To apply algorithm selection, instances should be characterised by features. It is important to have a comprehensive set of features that can describe the problem well. In order to automate the algorithm selection, we designed new features which fall into these categories: *local search probing features*, *geometric features* and *nearest neighbourhood features*.

Several features characterizing the TSP have been proposed in the literature, they are described in [5], [7], [8]. We compile the existing features (78) together with our new features (319) in Table I. Our new features are in bold face and prepended with *. We now discuss the new features:

1) New local search probing features:

In this section we describe the new features whose values are based on application of Lin-Kernighan heuristic to a problem instance. Both the inner processing of the heuristic and its results are reflected. The heuristic is evaluated in 10 runs and each run produces series of different values. Features are computed as statistics of such series of values.

Number of improving and best improving steps capture the properties of the search neighbourhood of Lin-Kernighan heuristics. A step is considered improving if it decreases the total tour cost, it is considered the best improving if the tour cost is not decreased more by any other improving step (i.e. there is no better step). Information about improving and best improving steps directly relate to a situation when the search algorithm has to decide which direction it should proceed in. If there are few improving steps, the search is probably attracted to some local minimum. No improving steps correspond to position in local minimum.

[1]http://www.adaptivebox.net/main/maos-tsp-algorithm/
[2]http://www.akira.ruc.dk/~keld/research/LKH/

TABLE I
FEATURES FOR TSP. OUR NEW FEATURES RESPECTIVELY FEATURE GROUPS ARE IN BOLD FACE AND LABELLED WITH *

Basic
Number of nodes
Cost Matrix
Cost statistics - mean, std. deviation, skew
Minimum spanning tree
Cost statistics - sum, mean, std. deviation, skew
Node degree statistics - mean, std. deviation, skew
Node depth - mean, max, median, std. deviation
Cluster distance features
Cluster distance

- based on min. spanning tree - mean, std. deviation, skew
- based on GDBSCAN - #clusters, #outliers, variation coefficient of cluster sizes

Local search probing
Tour costs from construction heuristics - mean, std. deviation, skew
Local minimum of tour length - mean, std. deviation, skew
Improvement per step - mean, std. deviation, skew
Steps to local minimum - mean, std. deviation, skew
Distance between local minima - mean, std. deviation, skew
Probability of edges in local minima - mean, std. deviation, skew
****Number of improving steps*** - mean, std. deviation, skew
****Number of best improving steps*** - mean, std. deviation, skew
****Edge lengths in quartiles*** - mean, std. deviation, skew
****Tour segments***

- segment length - mean, std. deviation, skew
- segment edge count - mean, std. deviation, skew
- edge lengths in segment - mean, std. deviation, skew

****Tour intersections in plane*** - mean, std. deviation, skew
Branch and cut
Improvement per cut - mean, std. deviation, skew
Ratio of upper bound and lower bound
Solution after probing - statistics on non-integer values
Ruggedness
Autocorrelation coefficient
Timing features
Time for computation of other groups of features
Node distribution features
Cost matrix standard deviation - after normalization to square $[0, 400]^2$
Fraction of distinct distances
Centroid coordinates
Radius - mean distance from node to centroid
Geometric features
Area - area of rectangle containing nodes
Hull area - area of convex hull (after normalization to square $[0, 1]^2$)
Fraction of nodes on the hull
Angle of edges connection 2 nearest neighbours - mean, std. deviation, skew, min, max, median
****Cosinus of the angle above*** - mean, std. deviation, skew, min, max, median
****Distance of nodes to hull contour*** - mead, std. deviation, min, max, median
****Edge lengths of the convex hull*** - mean, std. deviation, min, max, median
Nearest neighbourhood features
Nearest neighbour distance - std. deviation, variance
****Input degree in directed kNN graph*** - min, max, q1, median, q3, std. deviation, skew
****Strongly connected components in directed kNN graph***

- #components, #components / n
- size of components - min, max, median, mean, std. deviation, skew
- normalised size of components - min, max, median, mean, std. deviation, skew

****Weakly connected components in kNN graph***

- #components, #components / n
- size of components - min, max, median, mean, std. deviation, skew
- normalised size of components - min, max, median, mean, std. deviation, skew

****Ratio of number of strongly and weakly connected components***

Edge length in quartiles are derived from the analyses of generated tours. The histogram of edge lengths is analysed. We normalise the lengths of edges of each tour separately. Then the edges are sorted by their lengths and split to quartiles. Quartiles of edges are joined for all tours and statistics on the normalised edge lengths within quartiles are calculated. The intuition behind these features is that a bias in histogram to certain values relates to complexity of the search. If most values are at the beginning of the histogram (i.e. 0 after edge length normalization), the tour consists mostly of short edges and only few long links occur. Long links can, for example, connect some clusters.

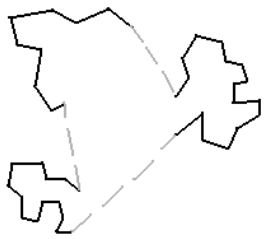

Fig. 2. Gray dashed line represent the removed long edges, black lines are the remaining edges which constitute the tour segments.

Tour segments' features are obtained by first cutting away the long edges from the tour. When a tour is found by the heuristic, its edge lengths are normalised. More precisely, the edges that are 1.5 times longer than shortest edge in 4th quartile are removed. But at least top 5% of the longest edges are always removed. The remaining tour segments (see Figure 2) are then analysed. The computed features are the statistics of the length of segments, statistics of the number of edges in each segment and statistics of the length of edges

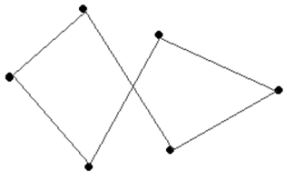

Fig. 3. A tour has 1 intersection which can be easily removed by an edge swap.

within segments. This information reflects how the short edges are grouped together within the tour. If segments are mostly short, it means that long edges are evenly present in the tour. Motivation for the features concerning the segments comes from the idea that segments might represent a part of the tour inside clusters of nodes while long edges interlink such clusters.

Tour intersections in plane reflect the placement of the tour in Euclidean plane. The number of line intersections of one tour is normalised and recorded for each heuristically generated tour. Statistics of these numbers are extracted as features. TSP tours in Euclidean plane tend to use edges which do not intersect any other edge in the tour when drawn in plane. One can quickly think of primitive cases where simple edge swap removes the intersection and probably also shortens the total tour, see Figure 3. Therefore, we expect that the number of intersections expresses how successful the heuristic was during its search. Note that these features are on the verge of local probing and geometric features.

2) New geometric features:

The novel features from this category are based on convex hull of the graph nodes in the Euclidean plane . Note, that convex hull of n points in plane can be computed in $O(n \cdot log(n))$ [15].

Edge lengths of the convex hull are the statistics reflecting the shape of the convex hull. Relatively short edges of the hull may indicate that optimal tour will contain them, on the other hand, long edges probably just reflect that interesting parts of the instance are more inside the hull.

Distance of the nodes to hull contour is calculated for all inner nodes (which are not on the hull) and corresponding statistics are extracted as features. For example, consider an instance, where all nodes are placed on circle. Then the distance to convex hull contour is zero for all nodes. However, if there are many points scattered inside this circle, the average distance grows. At the same time, the optimal solution for the instance is no longer obvious. Similar situation is in Figure 4. We expect these features to correspond with the complexity of the search.

3) New nearest neighbourhood features:

This category of features requires a transformation of an original instance's graph. The transformation will allow us to quantify the local relations between nodes by focusing on the k nearest neighbours of each point. We can, for example, reflect on groupings of points in a sense which is different from the standard clustering approach. Therefore, we expect

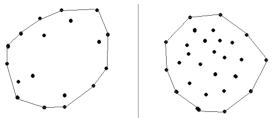

Fig. 4. Two instances are shown. Average distance of points to hull contour is bigger for the instance on the right side.

Fig. 5. Above is a kNN undirected graph of some graph G. Each node is connected with its three nearest neighbours, i.e. $k = 3$. The result of the transformation has 2 connected components.

our novel features to capture new important information about the problem instance. To best of our knowledge, only features stemming from simple statistics of nearest neighbour distances are present in the literature. For further description of the novel features, we have to define two transformations of an original instance's graph.

Definition 1: Let a graph $G = (V, E)$ and a cost function $C : V \times V \mapsto \mathbb{R}$ be given. Let $N(v, i), v \in V, 1 \leq i < |V|$ be an i-th nearest neighbour of vertex v, i.e. $C(v, N(v, 1)) \leq C(v, N(v, 2)) \leq \ldots \leq C(v, N(v, |V| - 1))$.

Let, for some $k \in \mathbb{N}$, $E_k = \{(v, N(v, i))|v \in V, 1 \leq i \leq k\}$. Then we call $G_k = (V, E_k)$ a kNN directed graph of G.

Let for some $k \in \mathbb{N}$, $E'_k = \{\{v, N(v, i)\}|v \in V, 1 \leq i \leq k\}$. We call $G'_k = (V, E'_k)$ a kNN (undirected) graph of G.

Informally, we defined a directed and undirected version of graph G, where only edges that connect k nearest neighbours of each node are preserved. Figure 5 shows an example of kNN undirected graph, where $k = 3$. Reinelt [16] mentions such graphs with regard to heuristics for TSPs; however, we aim to extract new features and use the transformation as a mid-step. Moreover, such a transformation can be done in $O(n \cdot log(n) + k \cdot n)$ [17] and consequently the calculation of features is relatively fast.

Number of strongly (resp. weakly) connected components is information which can be directly observed from kNN (un)directed graph of TSP instance. There are two arguments why such information is relevant.

Firstly, we consider this as a different approach to a characterisation of presence of clusters in the problem instance. Standard clustering approaches usually try to dissect the nodes into clusters, while calculating the cluster centroids and adjusting the number of clusters. The number of clusters and centroid positions can be seen as rather global parameters affecting which cluster a node pertains to. On the other hand, number of nearest neighbours is a more local property and the splitting to components is a natural result. Number of components is not determined a priori or by optimization of any metric but merely stems from the neighbourhood relations of the nodes.

Secondly, the transformations are related to internal characteristics of the search algorithm, particularly of LKH. Lin-Kernighan heuristics is performing k-opt moves but it is rather

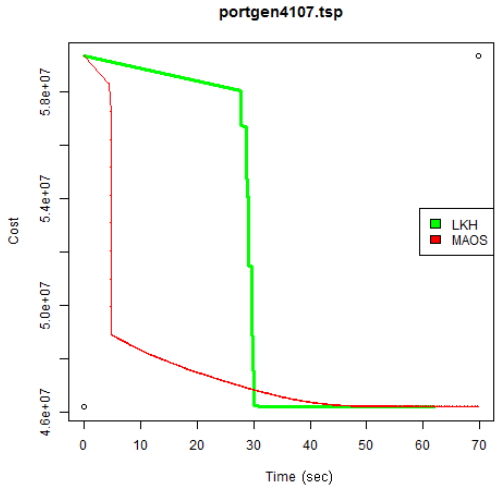

Fig. 6. Performance of algorithms on one problem instance

inefficient to check all such possible moves. Hence the search neighbourhood is usually focused on nearest neighbours of some current node. The kNN (un)directed graph thus relates to what the search algorithm considers in its processing.

Sizes of components and *ratio of strongly and weakly connected components* are statistics which we extract as features. The values are present also in a normalised version, i.e. divided by the number of nodes in the graph.

Input degree of nodes is another set of statistical features obtained from the kNN directed graph. Note that average input degree is omitted since it is always k.

All the aforementioned features were computed for several values of k, namely $k \in \{3, 5, 7, N^{1/3}, 2{\cdot}N^{1/3}, 1/2{\cdot}N^{1/2}, N^{1/2}\}$, where $N = |V|$.

IV. Data preparation and processing

This section will discuss our benchmarking problem instances, how we compile the training datasets and which machine-learning techniques we investigate.

We focus on Euclidean TSP instances in our study. We took subset of 163 instances applicable to our study from TSPLIB [18] and VLSI and National TSPs problem libraries [19], [20], which are based on real world problems. Another 2000 instances were generated using *port(c)gen* tools [21]. A half of them have uniform random placement of nodes, the others use clustered random placement. We consider this set of instances to be representative, as it includes real world problems and the hardness of instances differs. Moreover, the algorithms perform differently on the chosen instances.

We continuously recorded the interim solutions yielded by MAOS and LKH algorithms. Both algorithms were executed 10 times on each problem. We define the performance $Y_a^p(t)$ of algorithm $a \in \mathcal{A}$ on problem $p \in \mathcal{P}$ at time t as a median of solution qualities in the 10 runs. An example of such Y for both MAOS and LKH can be seen in figure 6.

Three training datasets are compiled – for time point 60s, for time point 1800s and for a time point when one of the algorithms finishes. The first two represent which algorithm performs best in 60s resp. 1800s. The last dataset assumes that we are willing to wait until one of the algorithms finishes. Note, that if an algorithm finished it does not necessarily mean it found the best solution.

Training datasets consist of rows where each row represents one problem instance. The row contains a set of features $\mathcal{F}$ which was described above and the class label (MAOS or LKH), which indicates the best performing algorithm on a given instance. Moreover, a weight w is assigned to each problem instance: Let C_{LKH} and C_{MAOS} be an interim tour cost found by the two algorithms, then we define the weight $w = \frac{max(C_{LKH}, C_{MAOS})}{min(C_{LKH}, C_{MAOS})} - 1$.

The datasets were normalised and discretised. Therefore, each dataset participated in the experiments in 3 forms: original, normalised and discretised. According to several preliminary experiments, we concluded that discretisation using Kononenko's MDL criterion [22] performs best for our purposes. Our instances, collected runtime data and datasets can be found online at: *http://www.dbai.tuwien.ac.at/user/pihera/tsp*

The task where a training dataset is given and the corresponding class label should be determined for the new problem instance using its features is a supervised classification task. In our study, we investigate different machine-learning techniques and compare their performance. The following techniques are applied: Bayesian networks, decision tree (J48), k-nearest neighbours (IBk), random forests (RF) and support vector machine (SVM) using their implementations in Weka 3.7 [23], which contains a collection of state-of-the-art machine learning techniques.

V. Experimental results

The run time data were collected for all the benchmark instances on machines with 4 processors Intel i3-2120 at 3.3GHz and 4GiB RAM with SUSE Linux. Four problem instances were solved in parallel on each machine using GNU Parallel [24] tool. MAOS used Java virtual machine with 800MiB heap limit. Machine dependent features were computed on a station with 8 processors Intel Xeon E5345 at 2.33GHz and 48GiB RAM.

We experimented with two types of datasets. The first type has three class labels LKH, MAOS and ANY. Label ANY represents problem instances with weight $w < 10^{-4}$, i.e. the two algorithms perform similarly. The second type of datasets is obtained from the first one by removing the instances with label ANY because we assume that selection of algorithm is irrelevant in such cases. Therefore, we have these two types of datasets: *3-class* datasets with labels LKH, MAOS and ANY and *2-class* datasets with labels LKH and MAOS.

To study the contribution of our new features, we consider three feature subsets and split the datasets accordingly. Let us denote the set of the features taken from literature as O (old) and the new features as N. Note that $\mathcal{F} = O \cup N$, where $\mathcal{F}$ denotes all features. Each dataset D is assessed in 3 versions: $D_{\mathcal{F}}, D_O, D_N$ where the index denotes the subset of features in the dataset.

Overall, we have 18 datasets $D_f^{t,q} \in \mathcal{D}$ where $t \in \{60, 1800, finishfirst\}$, $q \in \{2-class, 3-class\}$ and

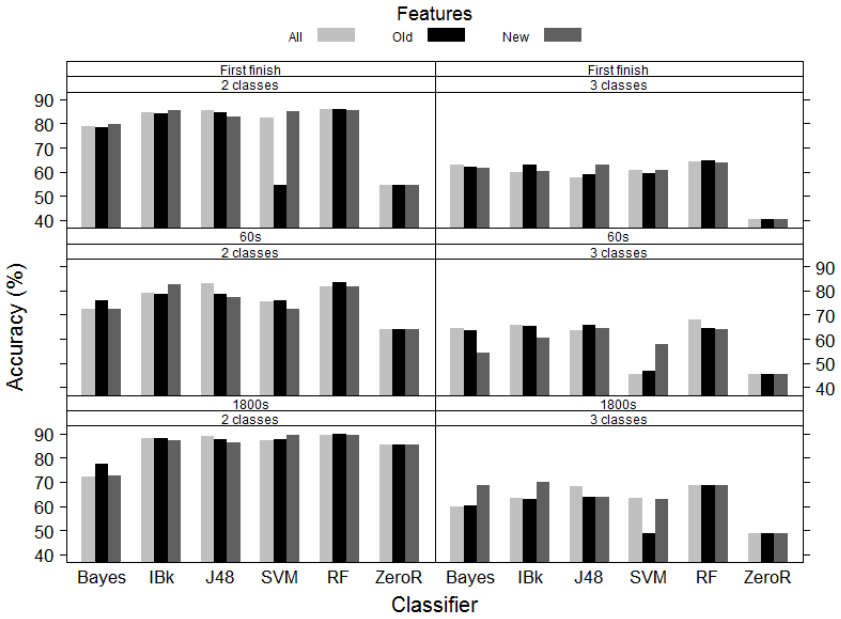

Fig. 7. Performance summary of all classifiers

$f \in \{\mathcal{F}, O, N\}$. Each such dataset is present in original, normalised and discretised form (see Section IV), which results in 54 datasets in total.

A. Classification

Each classification algorithm $C \in \mathcal{C}$ (see Section IV) was evaluated on each of the datasets $D \in \mathcal{D}$ with multiple different parameter settings. One of the classifiers' parameters determines which of the 3 forms (original, normalised, discretised) of each dataset is used. Accuracy of a classifier C for some parameter setting par_C is calculated as an average accuracy in 10 runs of 10-fold cross-validation. The best reached accuracy using some parameter settings par_C^{max} is taken as the resulting accuracy of a classifier C. The average accuracies of each 10-fold cross-validation run with par_C^{max} is recorded and used for Welch's statistical tests. All tests use the threshold $p < 0.01$.

We present the comparison of classifiers' accuracy in Figure 7, where for each time point $t \in \{60, 1800, firstfinish\}$ and set of classes $q \in \{2-class, 3-class\}$ a separate sub-figure is given. We included the performance of simple classifier which always predicts the majority class (ZeroR). We will now analyse and compare how particular classifiers performed.

In case of 1800s time constraint (bottom row in Figure 7), the RF (random forest) performed best and the difference between algorithms is confirmed to be statistically significant. Only for 3-class dataset with new features only, the IBk (k-nearest neighbours) classifier performed better. Similar scenario repeats for the $firstfinish$ dataset (the top row of Figure 7). RF dominates again with only exception of IBk for 2-class dataset and new features only. The 60s time constraint breaks the pattern, where J48 (decision tree) performs best in 3 cases, RF in two cases and IBk in one case. Interestingly, the 3-class dataset with new features is the situation where difference between J48 and RF is insignificant; otherwise the best and the second best classifier always differ significantly.

There is a substantial difference in obtained accuracy when 2 or 3 classes are considered. Let us first focus on the 3-class datasets (the right column of Figure 7). In all cases, the best achieved prediction is better than simple choice of the majority class (ZeroR). For 1800s, the best accuracy is around 69%; $\sim$64% for $firstfinish$ and 64%-67% for 60s dataset. For 3-class datasets, the level of hardness of prediction seems to be relatively the same for all time points.

The 2-class datasets were predicted with much higher accuracy. In case of 1800s time constraint, the best classifier (RF) achieved $\sim$89.5% over $\sim$85.4% of ZeroR. For 60s, the performance is $\sim$83% and for $firstfinish$ $\sim$86% but in both cases it is big improvement over ZeroR. Similarly as for 3-class datasets, the 1800s time point is predicted with highest accuracy; however, it seems that it is hard to improve a lot over the ZeroR.

In certain cases, the average accuracy of classifier was higher when only subset of all features was used. Therefore, we performed another set of tests after applying feature selection to each dataset $D \in \mathcal{D}$. The bi-directional *LinearForwardSearch* algorithm inside Weka with 10 back-tracking steps and *CfsSubsetEval* as evaluator selected the features. We report the average accuracy of classifiers on these datasets in Figure 8.

Feature selection caused that all classifiers have now at least once best performance. However, J48 and Bayesian network mostly perform best. When all features are given, Bayesian network dominates for $firstfinish$ datasets for both 2 and 3 classes, for 60s and 1800s time point with 3 classes. J48 dominates for the 60s time point in most cases. We observe that RF and IBk outperform the other classifiers before feature selection; however, J48 and Bayesian network perform well for datasets with fewer features.

B. Feature evaluation

Firstly, we analyse the datasets before feature selection. As we see in Figure 7 for case of $firstfinish$ datasets, there

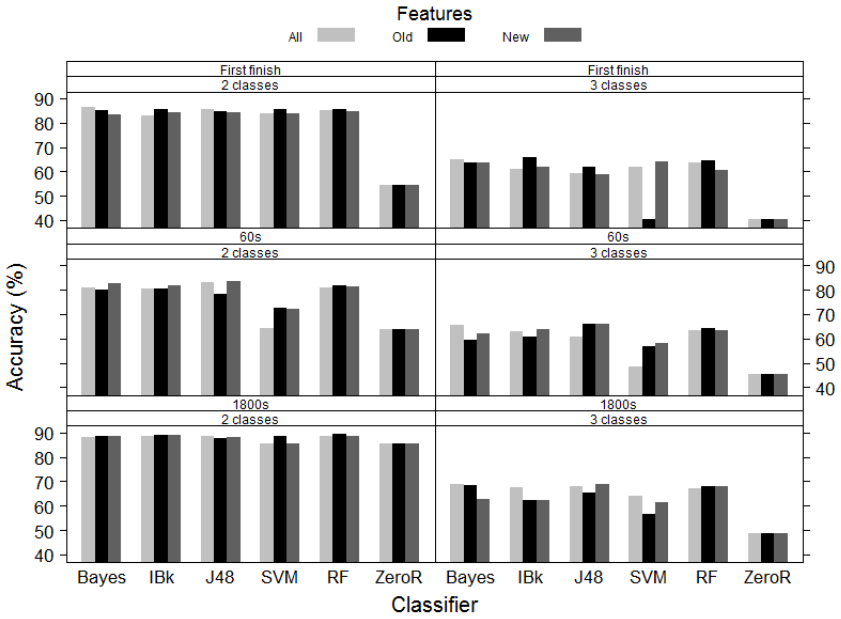

Fig. 8. Performance summary of all classifiers applied to datasets after feature selection

is no statistically significant difference between the feature sets (resp. between the best performing classifiers using these feature sets). For 1800s time constraint and 3 classes, the IBk trained on the new features performed significantly better than the best performing classifiers (RF) on the all features and the old features. The 60s time constraint breaks the pattern, as for 2 classes the old features perform best but all features perform best for 3 classes.

Feature selection changed which classifiers perform best but also the relative difference between the best performances on given feature sets. For 1800s time constraint and 2 classes, the old features are better than all features but not better (statistically) than the new features, i.e. feature selection failed to identify the important features in the set of all features. However, for 1800s and 3 classes, both the all and new features are better than the old ones. The very same result holds for 60s and 2 classes but there is no significant difference between feature sets for 60s and 3 classes. The $firstfinish$ dataset with 3 classes has the old feature set as the best performing one; however, for 2 classes it is the set of all features.

If we neglect the cases, where old features or new features outperform all features together, the addition of new features can statistically significantly improve the results. It is surprising that new features alone were competitive with the set of all features and old features because new features did not contain a lot of basic features. The number of vertices in the TSP instance is the only basic feature in set of new features.

We have further inspected which our new features remained in the set of all features after the feature selection process. Between 25% to 50% of the selected features in all datasets were related to kNN graph transformation. An example of their role in decision tree can be seen in Figure 9. We derive (for $firstfinish$, 2 classes and all features) that when there are more weakly connected components after kNN transformation and with bigger size, the LKH is the preferable algorithm.

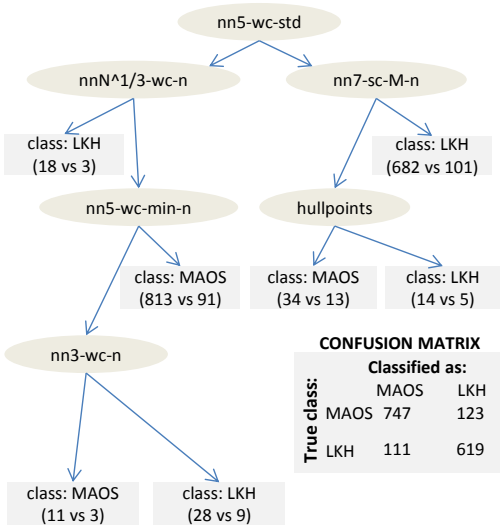

Fig. 9. Simplified decision tree for time point $firstfinish$ with all features and 2 classes. Accuracy of prediction is over 85% compared to 54.37% with ZeroR and confusion matrix does not indicate any bias. Features whose label starts with "nn" are based on kNN transformation. The values near the class labels denote the number of correctly and incorrectly classified instances in full dataset. Note, that feature values on the edges are left out for brevity.

Note, that the features used by the decision tree can be computed in $O(n^{4/3})$ time, where n is the number of vertices in the problem instance.

These results confirm that kNN features characterise the algorithm performance rather well. Edge lengths of the convex hull and tour segments' features were also present in most datasets after feature selection. The number of tour intersections was selected in datasets for $firstfinish$ time point. Distance to convex hull contour and number of (best) improving steps appeared in some of the datasets as well. Interesting observation is that the portfolio of selected features changes with the considered time point. To sum up, the feature

selection indicates that the highest contribution comes from the kNN features, lengths of convex hull edges and features related to tour segments.

VI. Conclusion

We tackled the scenario where user requires the best possible solution of a given problem instance within a certain time limit. Our approach is based on algorithm selection techniques which characterise the problem instance by features and determines the best solver by machine-learning algorithms. We used LKH [2] and MAOS [4], which, to best of our knowledge, are the state-of-the-art heuristic algorithms for TSP and they have not been considered together in algorithm selection context before.

For purpose of problem characterisation, we introduced a set of novel features and assessed their contribution with regard to the existing features described in literature. We compiled a set of more than 2000 benchmark problems, on which the feature sets and machine-learning algorithms were evaluated. Our experiments show that predicting which of the two TSP solvers will perform best can be substantially more successful than simple majority class prediction. We also show that addition of new features can improve the prediction accuracy and we confirmed that the difference is significant by statistical tests.

We identified the new kNN graph transformation features and tour segments' features (see Section III-1) to be especially helpful for improvements of algorithm selection accuracy. The kNN features were obtained by analysis of the LKH algorithm and the results confirmed our assumptions. They are the most important features among all the new features we introduced. Some of our features are derived purely from the graph of a problem instance. We believe that they could improve the characterisation of other problems beside TSP and be beneficial for other domains.

Our comparison of machine learning algorithms for this task revealed that k-nearest neighbours and random forests are convenient choice for datasets with a lot of features. However, if the dataset is preprocessed by feature selection, the decision trees and Bayesian networks outperform the other classifiers. The feature selection was beneficial, with respect to accuracy, mainly for datasets with almost 400 features. Moreover, the smaller subset of features is needed, faster is the algorithm selection process.

In future, we would like to study the factors affecting that some feature is more important at certain time point than at the other one. Moreover, we want to consider the time point as a variable parameter instead of working with a fixed set of values. As we have shown, the set of existing features can be extended and improved and we aim to invent new features and refine the existing ones as well.

VII. Acknowledgements

This work was supported by the Austrian Science Fund (FWF): P24814-N23. 1st author is financially supported by the Vienna PhD School of informatics [3].

[3] http://www.informatik.tuwien.ac.at/teaching/phdschool

References

[1] Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics). Princeton University Press (January 2007)

[2] Helsgaun, K.: An effective implementation of the Lin-Kernighan traveling salesman heuristic. Eur. Jour. of Operational Research **126**(1) (October 2000) 106–130

[3] Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling-salesman problem. Operations Research **21**(2) (March 1973) 498–516

[4] Xie, X.F., Liu, J.: Multiagent optimization system for solving the traveling salesman problem (TSP). IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **39**(2) (April 2009) 489–502

[5] Hutter, F., Xu, L., Hoos, H.H., Leyton-Brown, K.: Algorithm runtime prediction: Methods & evaluation. Artificial Intelligence **206** (2014) 79–111

[6] Kanda, J., de Carvalho, A.C.P.L.F., Hruschka, E.R., Soares, C.: Selection of algorithms to solve traveling salesman problems using meta-learning. In: Int. J. Hybrid Intell. Syst. Volume 8. (2011) 117–128

[7] Smith-Miles, K., van Hemert, J.: Discovering the suitability of optimisation algorithms by learning from evolved instances. Annals of Mathematics and Artificial Intelligence **61**(2) (2011) 87–104

[8] Mersmann, O., Bischl, B., Trautmann, H., Wagner, M., Bossek, J., Neumann, F.: A novel feature-based approach to characterize algorithm performance for the traveling salesperson problem. Annals of Math. and Artif. Intell. (March 2013)

[9] Smith-Miles, K., Wreford, B., Lopes, L., Insani, N.: Predicting metaheuristic performance on graph coloring problems using data mining. In Talbi, E.G., ed.: Hybrid Metaheuristics. Volume 434 of Studies in Computational Intelligence. Springer Berlin Heidelberg (2013) 417–432

[10] Musliu, N., Schwengerer, M.: Algorithm selection for the graph coloring problem. In Nicosia, G., Pardalos, P., eds.: Learning and Intelligent Optimization. Volume 7997 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 389–403

[11] Smith-Miles, K.A.: Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Computing Surveys **41**(1) (Dec 2008) 1–25

[12] Kotthoff, L.: Algorithm selection for combinatorial search problems: A survey. CoRR **abs/1210.7959** (2012)

[13] Rice, J.R.: The algorithm selection problem. In: Advances in Computers. Volume 15. Elsevier (1976) 65–118

[14] Cook, W.: World TSP. http://www.math.uwaterloo.ca/tsp/world/index.html (September 2013) Acessed: 2014-04-14.

[15] Toussaint, G.T.: A historical note on convex hull finding algorithms. Pattern Recognition Letters **3**(1) (1985) 21 – 28

[16] Reinelt, G.: The Traveling Salesman: Computational Solutions for TSP Applications. Springer-Verlag, Berlin, Heidelberg (1994)

[17] Callahan, P.B., Kosaraju, S.R.: A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. J. ACM **42**(1) (January 1995) 67–90

[18] Reinelt, G.: TSPLIB. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/ (January 2003) Acessed: 2014-01-21.

[19] Rohe, A.: VLSI data sets. http://www.math.uwaterloo.ca/tsp/vlsi/ (May 2013) Accessed: 2014-01-21.

[20] Cook, W.: National Traveling Salesman Problems. http://www.tsp.gatech.edu/world/countries.html (April 2009) Acessed: 2014-04-14.

[21] McGeoch, L.: Instance generating code for DIMACS TSP challenge. http://dimacs.rutgers.edu/Challenges/TSP/codes.zip Accessed: 2014-01-13.

[22] Kononenko, I.: On biases in estimating multi-valued attributes. In: IJCAI. Volume 95. (1995) 1034–1040

[23] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: An update. SIGKDD Explor. Newsl. **11**(1) (November 2009) 10–18

[24] Tange, O.: GNU Parallel – The Command-Line Power Tool. The USENIX Magazine **36** (2011) 42–47