Algorithm selection for TSP with time constraints on the algorithms

Josef Pihera[1,2] and Nysret Musliu[1]

[1] DBAI, TU Wien, Austria, {pihera, musliu}@dbai.tuwien.ac.at
[2] Vienna PhD School of Informatics

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 on all problem instances within a given time limit. Therefore, predicting the best performing algorithm for a particular instance is of a practical importance. In this paper, the TSP is studied in context of automated algorithm selection. We tackle this scenario as a machine learning task and experimentally investigate the prediction accuracy of various learning methods. To this aim, we extend the set of existing features in the literature and propose several novel features to better characterise the problem. To create the training data, two state-of-the-art (meta-)heuristic algorithms are systematically evaluated on more than 2000 problems. The contribution of the new features is statistically analysed and experiments show that adding our new features improves the prediction accuracy. We identified that two categories of new features, based on our kNN graph transformation and tour segments, are especially helpful. Overall, we show that our prediction can be substantially better than simple preference of an algorithm with the best performance for a majority of problem instances.

Keywords: TSP, algorithm selection, features, machine learning

1 Introduction

We analyse Travelling Salesman Problem (TSP) in context of algorithm selection where user needs an intelligent method to decide which of the available algorithms will perform best for a given problem instance. A new scenario is considered where a time constraint is given and user wants to obtain the most accurate interim solution. Properties of the TSP instances are captured by our novel features which are subsequently analysed and their contribution is confirmed by our experiments. Moreover, several machine learning techniques are compared and we study their convenience for our task.

Usually there is no universally best algorithm which dominates the other algorithms in all cases. Typically, an algorithm performs better than other algorithms only on a certain class of instances. Algorithm selection, as described in [1, 2], is predicting which of the given algorithms will show a maximum performance on a given problem instance. A problem instance is characterized by features, which should capture the relation between an algorithm and its performance on the instance. We study the scenario where the relative quality of interim solutions has to be predicted at a given time point. Our aim is to make such prediction possible for TSP.

TSP is a famous NP-hard problem with many important applications such as logistics, genome sequencing and telescope aiming [3]. However, even the prominent exact solver Concorde [3] may need days for a relatively small problem instance [4]. But imagine a driver of a parcel service, who needs to determine the route for tomorrow. A solver is useful for him/her only if it yields some solution before his/her work shift.

Because the user can wait for only a certain amount of time, we introduce a time constraint. The time constraint is considered in three variants – an algorithm runs for a short time, for a long time or the fastest from algorithms is allowed to finish. The details will be given in Section 4.2.

In order to meet the imposed time constraint, a solving algorithm must be able to yield some interim solution before it finishes its work. As the available computation time might be short, (meta-)heuristic solvers are the most suitable candidates. We have tested several different algorithms but LKH [5] and MAOS [6] were the best performing ones. LKH is a state-of-the-art heuristic solver based on Lin-Kernighan [7] heuristic. It provides very good results (though possibly suboptimal) in a shorter time than Concorde [8]. MAOS is a meta-heuristic which yields competitive interim solutions compared to those of LKH. Our goal is to predict the relative performance of these two state-of-the-art solvers which fulfil the requirements of our problem scenario.

As was mentioned above, algorithm selection requires relevant features which help us determine the best performing algorithm. We invent and describe several categories of new features which extend the set of existing features in the literature. The existing features [9–11] comprise categories such as local search probing, minimum spanning tree, branch and cut, clustering and geometric features. We introduce a kNN graph transformation which yields a whole new feature category and propose new local search probing and geometric features. Some of these features are derived directly from the graph of the problem instance and could be generalized for other problems beside TSP. We compile three feature sets – the features in literature, our new features and all of them together. This allows us to evaluate the contribution of the new features. Experiments in this paper show that our new features based on kNN transformation and on local search probing are especially helpful.

For our experiments, LKH and MAOS algorithms were evaluated on more than 2000 instances with average size of more than 2700 vertices. The run time data were collected after weeks of computations and compiled together with the computed features into training datasets. The datasets were used for training of the state-of-the-art learning algorithms (so called classifiers). Several machine learning techniques were applied to our datasets, both for pre-processing and for classification. We assessed the classifiers with many different parameter settings on all our datasets and present a comparison of the 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.

The most related research to our scenario is the one of Hutter el al. [9]. They described a method for prediction of runtime of Concorde and LKH on problem instances with less 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. Moreover, we predict the best performing algorithm at given time point instead of the prediction of runtime.

Kanda et al. [13] used a set of simple meta-heuristic algorithms and evaluated them on instances with less than 100 vertices. Their instances are defined on general graph (both symmetric and asymmetric), whereas we focus on instances in Euclidean plane. As a result, their features can not be used in our scenario. Smith-Miles and Hemert [14] consider only 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 evolutionary created instances with 100 nodes which were intentionally easy or hard and investigated their features. Mersmann et al. [11] also used instances with up to 100 vertices but only a particular search operator is studied. Their aim was also to study which instances are easy and which are hard and evolved them accordingly. Moreover, they morphed the hard instances into the easy ones. We use features from the aforementioned works; however, we do not study hardness of instances but we want to predict the best performing algorithm. More information on algorithm selection for combinatorial problems can be found in a survey [15].

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 and consider larger instances than in aforementioned works. Moreover, we invent new features which help us to better characterize the problem instances in our scenario.

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

- Invention of new features which enhance the accuracy of algorithm selection
- Study of the algorithm selection for TSP with a time constraint
- Identification of those new features which are important for prediction of the best performing algorithm in our scenario
- Identification of convenient machine learning algorithms for our scenario
- Extensive assessment on comprehensive set of benchmark instances

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

2 Background information

2.1 Travelling Salesman Problem

We focus on a symmetric version of TSP, particularly in the Euclidean plane. Therefore, consider a complete undirected graph $G = (V, E)$, where $V = \{1, \ldots, n\}$ is a vertex set and $E = \{\{i, j\} | i, j \in V, i < j\}$ is an edge set. A cost function $C : E \to \mathbb{R}$ determines the cost of edges. For Euclidean plane instances, the coordinates of vertices V are given, E is considered as $E = V \times V$ and C corresponds to Euclidean L2 norm, i.e. the distance between vertices in plane.

We define a Hamiltonian circuit $L = \{v_i\}_{(i=1)}^{n}$ as a sequence of vertices $v_i \in V$ where $v_i \neq v_j$ for $1 \leq i < j \leq n$. Because the graph G is complete, the edges $\{v_i, v_{i+1}\}$ for $1 \leq i < n$ always exist, as well as the edge v_n, v_1. A cost $c(L)$ of the circuit L is defined as a $c(L) = \sum_{i=1}^{n-1} C(\{v_i, v_{i+1}\}) + C(\{v_n, v_1\})$.

Given a graph G with a cost function C, the travelling salesman problem is to find a Hamiltonian circuit L such that for all other Hamiltonian circuits L' holds $c(L) \leq c(L')$. Karp [16] proved that undirected Hamiltonian circuit problem is NP-complete, which implies NP-hardness of the travelling salesman problem. But even the subclass of the TSP where the vertices are in the Euclidean plane and the cost function corresponds with their distance was proven in [17] to be NP-complete.

2.2 Algorithm selection

An algorithm selection problem was formulated by Rice [1] already in 1976. The same idea emerged in the machine-learning community where the goal was to a choose proper learning algorithm for a given dataset. As it was in fact learning about learning, the term meta-learning came into existence without noticing the work of Rice [2]. Algorithm selection became popular again in last decade and it was applied to various domains, we refer reader to surveys [2, 15].

One may ask, whether we can not simply run all the algorithms in parallel instead of selecting an algorithm. This is possible of course, but algorithm selection offers much more efficient usage of resources. For example, if you have more problem instances to be solved, you can solve different instances in parallel with algorithm selection, whereas you could solve only one instance with all algorithms otherwise.

We will briefly describe the algorithm selection framework, we follow the description in [2]. 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 maximized. See the scheme of the situation 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

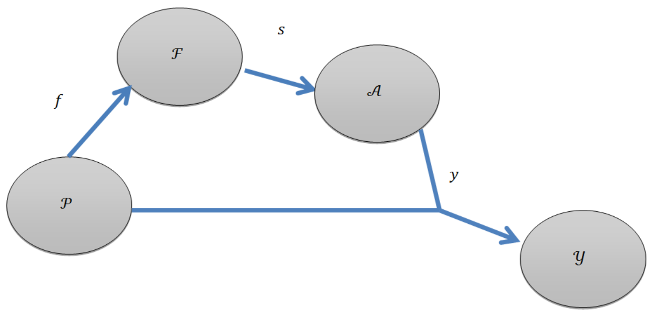


Fig. 1. Algorithm selection framework

selection successfully [1]. 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 the mapping s towards a special class of inputs only.

3 Features for TSP

Problem instances are characterized by features; therefore, it is important to have a comprehensive set of features that can describe the problem well. In order to solve the algorithm selection problem, we designed new features which fall into these categories:

- Local search probing features
- Geometric features
- Nearest neighbourhood features

Some features were already described in the literature. However, features must be informative with regard to the set of algorithms $\mathcal{A}$. In [14], authors state that their set of features is sufficient, but $\mathcal{A}$ contains two variants of Lin-Kernighan heuristic only. Hutter et. al [9] use Concorde and LKH and we observe that each of the algorithms corresponds with a group of features in their feature set. Kanda et. al [13] selects an algorithm only from a set of simple meta-heuristic ones. Mersmann [11] focuses only on the type of search which is performed by Lin-Kernighan heuristic. To best of our knowledge, the algorithms MAOS and LKH were never considered for this task; therefore, we introduce completely new features which help us to construct the mapping s.

We include the existing features in our study in order to analyse the contribution of the new features. The overview of all features used in our experiments is given in Table 1, we follow the table layout given in [9] for easier comparison.

Sets of features described by Hutter et al. [9] and Smith-Miles et al. [10] were adopted together with their computation code for feature extraction. We also used features from Mersmann [11], but implemented them on our own because of efficiency reasons. We refer the reader to find the description of adopted features in aforementioned papers. The new features we introduced will now be described.

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 coeficient
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
- normalized 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
- normalized size of components - min, max, median, mean, std. deviation, skew

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

Table 1. Features for TSP, new features resp. feature groups are labelled with *

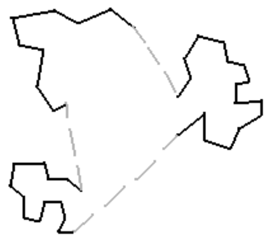


Fig. 2. Gray dashed line represent the removed long edges, black lines are the remaining edges which constitute the tour segments.

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 run repeatedly and features are computed as statistics of series of different 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 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. Similarly, one best improving step attracts the search to a certain direction, but many best improving steps probably lead to several different local minima. We expect these collected characteristics to correlate with search algorithm's performance.

Edge length in quartiles are derived from the analyses the of generated tours. The histogram of edge lengths is analysed. We normalize 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 normalized edge lengths within quartiles are calculated. The intuition behind these features is that bias in histogram to certain values relates to complexity of 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.

Tour segments' features are obtained by first cutting away the long edges from the tour. When a tour is found by heuristic, its edge lengths are normalized. 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

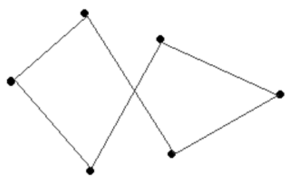


Fig. 3. A tour has 1 intersection which can be easily removed by an edge swap.

number of edges in each segment and statistics of the length of edges within segments. This information reflects how the short edges are groupped 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 normalized and recorded for each heuristically generated tour. Statistics of these numbers is 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 is on the verge of local probing and geometric features.

The new features from local search probing category were implemented taking advantage of existing code. We have adjusted the code of Hutter et al. (available from [20]) and of Applegate et al. [3]. The implementation of Lin-Kernighan heuristic in Concorde was extended to provide information about properties of search neighbourhood.

New geometric features

The novel features from this category are based on convex hull of the graph nodes in the Euclidean plane. *Cosinus angle between edges connecting 2 nearest neighbours* is the only exemption, however it is a transformation of the existing feature from Mersmann [11].

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.

Mersmann [11] used some features concerning the convex hull, however we found that the aforementioned problem characteristics were left unnoticed.

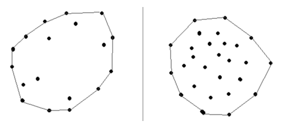


Fig. 4. Two instances are shown. Average distance of points to hull contour is bigger for the instance on the right side.

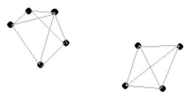


Fig. 5. Above is an kNN (undirected) graph of some graph G. Each node is connected with its three nearest neighbours, i.e. $k = 3$.

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 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 $k \in \mathbb{N}$, $E'_k = \{\{v, N(v, i)\} | v \in V, 1 \leq i \leq k\}$. Then 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 $k = 3$.

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 apriori 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 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 extract as features. The values are present also in a normalized 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 $k \in \{3, 5, 7, N^{1/3}, 2{\cdot}N^{1/3}, 1/2{\cdot}N^{1/2}, N^{1/2}\}$.

4 Experiment settings

This section will discuss what are our benchmarking problem instances and what are the objectives. We will also explain how we compile the training datasets and which machine-learning techniques we apply.

4.1 Benchmark instances

We focus on Euclidean TSP instances in our study. We took convenient subset of instances from TSPLIB [18] and VLSI and National TSPs problem library [21, 8], which are based on real world problems. More instances were generated using *portgen* and *portcgen* tools [22]. We consider this set of instances to be representative, as it includes real world problems and hardness of instances differs. Moreover, the algorithms perform differently on the chosen instances. There are total 2163 problem instances in our dataset. 163 of those come from the problem libraries, 995 of them were generated as uniform random placement of nodes and 1005 as clustered random placement of nodes.

The size of instances was selected with hardware and software limitations in mind. Since MAOS algorithm exceeded memory limit when number of nodes was more than 8000, only few instances are larger. Very small instances (less than 100 vertices) caused problems to existing feature computation code and were omitted if that was the case. The randomly generated instances cover uniformly the interval between 500 and 5000 vertices. You can see the histogram in Figure 6.

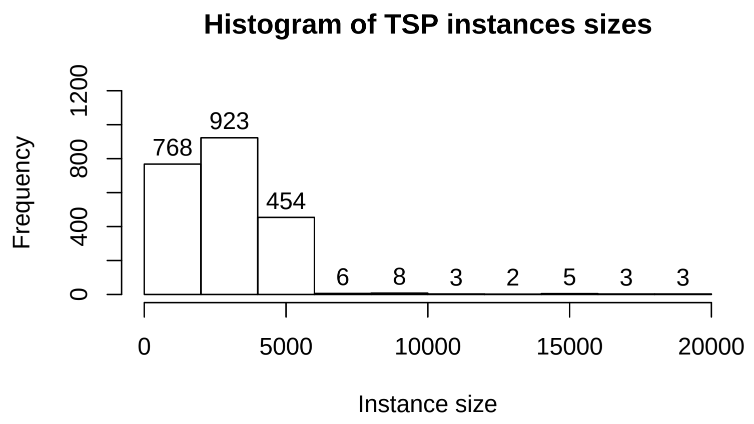


Fig. 6. Sizes of TSP instances in dataset

4.2 Problem definition and data

Typical algorithm selection problem setting comprises problem instances, algorithms, features and performance measure. Usual performance measure is either solution quality or run time. Given enough time (exponential with input size), user can obtain optimal solution with tools like Concorde. However, we focus on a natural situation where time is a constraint. The question we pose is: *What is the best solution we can get within given amount of time?* We rephrase it for algorithm selection context: *Which algorithm yields the best solution within given amount of time?*

To answer this question, we continuously recorded the interim solutions yielded by MAOS and LKH algorithms. Both algorithms were executed 10 times on each problem. For each interim solution (tour) we know its cost and a time which an algorithm needed to reach such a solution. Therefore, for each run $1 \leq i \leq 10$ of algorithm $a \in \mathcal{A}$ on problem $p \in \mathcal{P}$ we can construct a performance function $f_a^p(i,t)$ whose value is the cost of the tour yielded after t seconds from execution. We define the performance $F_a^p(t)$ of algorithm a on problem p at time t as a median of $f_a^p(1,t), f_a^p(2,t), \ldots, f_a^p(10,t)$. An example of such F for both MAOS and LKH can be seen in figure 7.

The performance of an algorithm on a given problem instance is calculated as a median of the performances during 10 runs, which should ensure that the most common algorithm run is represented. Hence, we can answer which algorithm performs best on a given problem at any given time point. We compiled 3 training datasets – dataset for time point 60s, dataset for time point 1800s and dataset for a time point when some of the algorithms finishes as first. The interpretations of the first two datasets are clear, they represent which algorithm performs best (finds best solution) in 60s resp. 1800s. The last dataset assumes that we are

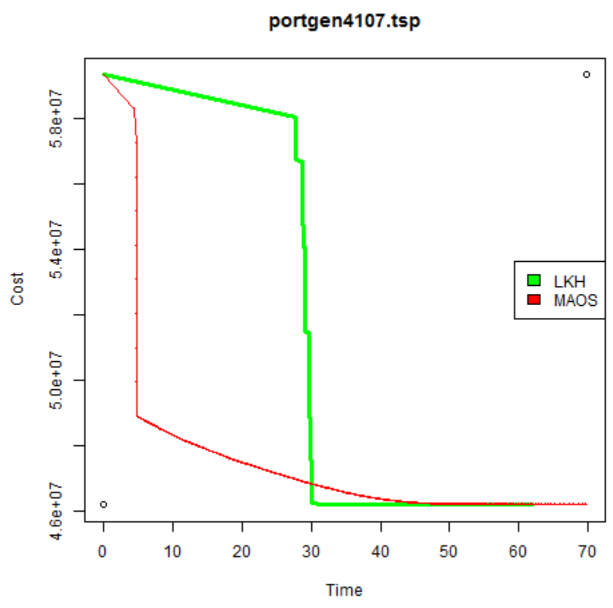


Fig. 7. Performance of algorithms on one problem instance

willing to wait until one of the algorithm finishes. The one which finished has not necessarily found the best solution. Which one performed best is represented in the dataset.

The 60s represent a short time for which user might be willing to wait and it is long enough for both the algorithms to mostly get over the initialisation phase. 1800s is long enough to fine-tune some tour and get relatively close to optimum. The first-finish dataset presents interesting question whether the faster algorithm is really the better one.

The problem instances in dataset are not merely labelled by algorithm which performed best, but also a weight is assigned to this answer. Note, that in some cases the difference between algorithms is negligible, or they both found an optimal answer. If C_LKH and C_MAOS is a tour cost found by the two algorithms, we define the weight as $w = \frac{max(C_{LKH}, C_{MAOS})}{min(C_{LKH}, C_{MAOS})} - 1$.

To sum it up, 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, the class label and the weight w. Such datasets can now be subjected to machine learning algorithms.

4.3 Classification Algorithms

The task where a training dataset is given and a corresponding class label should be determined from the available features is a supervised classification task. In our study, we compare the performance of several classifiers on the datasets. The classifiers are namely Bayes networks, decision tree (J48), k-nearest neighbours (IBk), random forests (RF) and support vector machine (SVM). Let us denote the set of classification algorithms $\mathcal{C}$. We used their implementations in Weka [23], which contains a collection of state-of-the-art machine learning techniques. We also considered multi-layer perceptron from Weka but its performance was inferior to other classifiers during experiments; moreover, it required excessive amount of time to finish the training and was prone to failures.

4.4 Data pre-processing

The usual preprocessing procedure was applied to all the datasets, which are normalization and discretisation. Therefore, each dataset participated in the experiments in three forms – original form, normalized form and discretised form. According to several preliminary experiments, we concluded that discretisation using Kononenko's MDL criterion [24] performs best for our purposes. However, we discretised the original dataset (not the normalized one) because of floating point problems in Weka implementation.

5 Experimental assessment

The run time data were collected for all the benchmark instances on 3 computers, each 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 [25] tool. Each MAOS execution used Java virtual machine with 800MiB heap limit. Features were computed on different machines, however machine dependent features are only those which have their time of computation recorded. Machine dependent features were computed on a station with 8 processors Intel Xeon E5345 at 2.33GHz and 48GiB RAM.

The first experiments solely focused on dataset with two class labels – LKH, MAOS – and no weights. However, this led to biases towards majority class. Following the techniques mentioned in [26], the datasets were refined. We balanced the classes using under-sampling of the majority class; nevertheless, the classifiers still clearly preferred one of the classes depending on the ratio of under-sampling.

To counteract such effects, we used weighting of instances, which on the other hand biased the classifiers towards the instances, where the weights are significantly larger. We considered several weighting metrics; however, any such metric expresses a subjective view of a solution cost difference. We strived to minimize the effect of such subjective quantifications and introduced a third class label ANY whose semantic is that both algorithms performed comparably well. The existence of such a class is well supported by the fact, that in two datasets there are around 70 instances where both algorithms yielded tours with exactly same costs. We relabelled all instances to class ANY if their weight w was smaller than 10^{-4}, i.e. the solution cost differed by 0.01%. Note that the only subjective decision is whether instance belongs to the class ANY or to the original one. However, there is no bias in a sense that one instance would be x-times more important than the other, as is the case of weighted instances.

There is a question what is the actual benefit of class ANY for a user who merely wants to select an algorithm. Let us assume that it is only important to classify correctly the instances which matter, i.e. they retained their class label LKH or MAOS. Therefore, we work with two kinds of datasets in our experiments:

– *3-class* datasets with labels LKH, MAOS and ANY, as described above
– *2-class* datasets with labels LKH, MAOS which are obtained by removing all instances with label ANY

As we also want to study the contribution of our new features, we further split the datasets into several new ones. Let us denote the set of the features taken from literature as O (old) and the new features as N. Note that $\mathcal{F} = O \cup N$. Each dataset D is used in 3 versions - $D_{\mathcal{F}} = D$, D_O, D_N, where the index denotes the subset of features in the dataset. This way, we can compare the classifier performance for old (existing) features, new features only, and all features together.

To sum it up, we have datasets $D_f^{t,q} \in \mathcal{D}$ where $t \in \{60, 1800, firstfinish\}, q \in \{2-class, 3-class\}, f \in \{\mathcal{F}, O, N\}$, which is 18 different datasets. Note, that each such dataset is present in 3 different forms – original, normalized and discretized (see section 4.4) – which results in 54 datasets total.

5.1 Classification

Each classifying algorithm $C \in \mathcal{C}$ (see subsection 4.3) was evaluated on each of the 18 datasets $D \in \mathcal{D}$. All the algorithms, as implemented in Weka, have a set of parameters. In order to evaluate the algorithms systematically, we applied each classifier $C \in \mathcal{C}$ to each dataset $D \in \mathcal{D}$ with multiple different parameter settings $\{Par_1^C, Par_2^C, \ldots, Par_{r_C}^C\}, r_C \in \mathbb{N}$. Note, that in order to simplify the formalism, the 3 forms (original, normalized, discretised) of each dataset $D_f^{t,q}$ are not differentiated in $\mathcal{D}$. We rather think of them as being specified by parameters, i.e. one of the classifier's parameter is which dataset form is used. Accuracy $V(C, Par_i^C)$ of a classifier C for some parameter setting $Par_i^C, 1 \leq i \leq r_C$ is calculated as an average accuracy in 10 runs of 10-fold cross-validation, i.e. the classifier is trained 100-times for each parameter setting. The accuracy of a classifier C is taken as its accuracy for the best parameter setting, i.e. $V(C) = max_{1 \leq i \leq r_C} V(C, Par_i^C)$. We record the average accuracies of each 10-fold cross-validation run, which is 10 values, and use them to perform Welch's statistical test for comparison of both the datasets and the algorithms. All tests use the threshold $p < 0.01$.

We present the comparison of classifiers' accuracy in Figure 8, 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

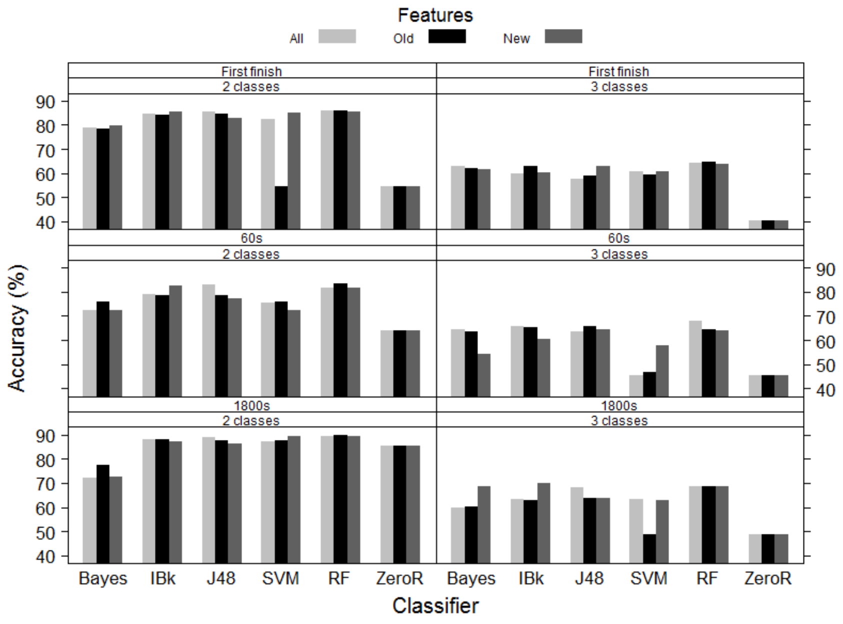


Fig. 8. Performance summary of all classifiers

which always predicts the most common class (ZeroR). We will now analyse and compare how particular classifiers performed. Then the difference between the hardness of prediction for the tree time points will be discussed.

In case of 1800s time constraint (bottom row in Figure 8), 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; again the difference is statistically significant. Similar scenario repeats for the $firstfinish$ dataset, which is in the top row of Figure 8. 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.

We can see that there is a substantial difference in obtained accuracy when 2 or 3 classes are considered. Let us first focus on the 3-class datasets, which are in the right column of Figure 8. In all cases, the best achieved prediction is better than simple choice of the most common class (ZeroR). For 1800s (bottom row), the best accuracy is around 69%; ∼64% for $firstfinish$ (top row) and 64%-67% for 60s dataset. It seems that for 3-class datasets, the level of hardness of prediction is relatively the same for all time points. Even though the 1800s time point was predicted with the best accuracy, we must take into account that

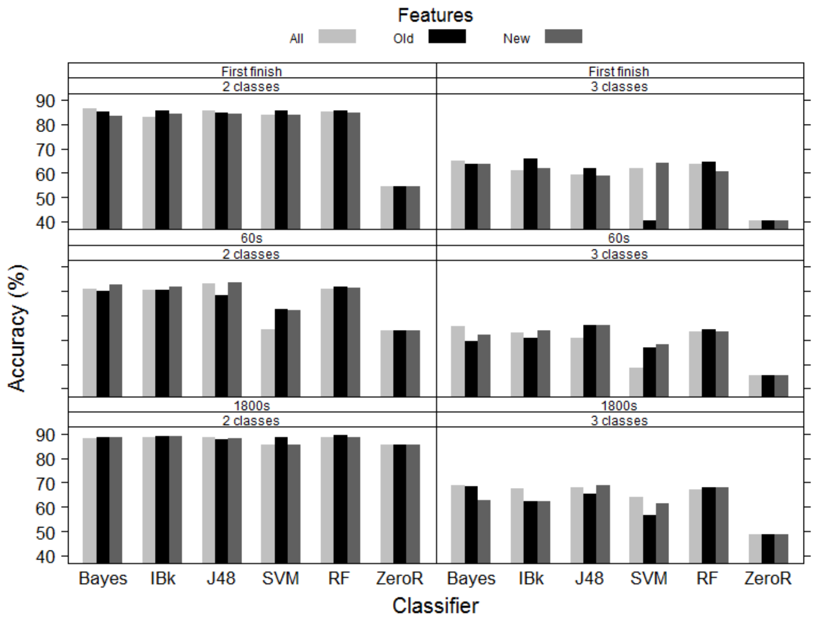


Fig. 9. Performance summary of all classifiers applied to datasets after feature selection

the ZeroR classifier, which can be seen as a base line, also performed better than in other cases.

The 2-class datasets were predicted with much higher accuracy. Note, that 2-class dataset is obtained by removal of the instances whose class was ANY, i.e. their final selection of algorithm was considered unimportant. In case of 1800s time constraint, the best classifier (RF) achieved ∼89.5% over ∼85.4% of ZeroR. For 60s, the performance is ∼83% and for $firstfinish$ ∼86% but in both cases it is big improvement over ZeroR. Analogically 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. The classification task is the hardest for 60s time point.

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: feature selection was applied to each dataset $D \in \mathcal{D}$ and then the classifiers were trained and evaluated. The bi-directional *LinearForwardSearch* algorithm inside Weka with 10 back-tracking steps and *CfsSubsetEval* as evaluator was used for the feature selection process. We report the average accuracy of classifiers on these datasets in Figure 9.

Feature selection caused that the RF and IBk are no longer the dominating classifiers. All classifiers are now best in some case, though SVM only once and not statistically significantly. However, it is mostly J48 (decision tree) and Bayes network that perform best. Especially when all features are given, Bayes network dominates for $firstfinish$ datasets for both 2 and 3 classes, for 60 and 1800s time point with 3 classes. On the other hand, J48 dominates for the 60s time point in most cases. We observe that when a lot of features are present (before feature selection), RF and IBk are really convenient. But in case of fewer features, the J48 and Bayes network are the classifiers of choice for our task.

The impact of feature selection was mostly negative when applied to the set of old features. It holds for the new features as well, except for the case of 60s time point where the accuracy was improved. Feature selection improved the average accuracy when applied to the set of all features in 4 from 6 cases. However, in case of 60s time point and 3 classes, the accuracy deteriorated by more than 2%. Therefore, we can recommend to perform feature selection only when all features are used.

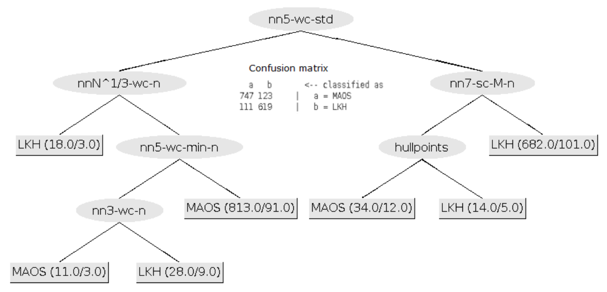
5.2 Feature evaluation

Our next step is to evaluate the contribution of new features. Firstly, we want to compare the accuracy of best classifiers for the different feature sets. The datasets $D_f^{t,q} \in \mathcal{D}$ enable us to compare the different feature sets for each fixed time point t and number of classes q. Two feature sets $f_1, f_2 \in \{\mathcal{F}, B, N\}$ are compared using Welch's statistical test with $p < 0.01$ and hypothesis that average accuracy of best $C_{D,f_1,t,q}$ on $D_{f_1}^{t,q}$ is greater than accuracy of best $C_{D,f_2,t,q}$ on $D_{f_2}^{t,q}$.

Firstly, we analyse the datasets before feature selection. For case of $firstfinish$ datasets, there 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 only performed significantly better than the best performing classifiers (both were RF) on the all features and the old features. The 60s time constraint breaks again the pattern, as for 2 classes the old features perform best. For 3 classes and time point 60s, set of all features performs best.

Feature selection changed which classifiers now 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 there is no significant statistical difference between the old features and the new features only. It seems that 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; 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 and new features performed better than all features together, we see that addition of new features can statistically significantly improve the results. The fact, which we deem surprising, is that new features alone were competitive with the set of the all features and the old features. Note, that the new features didn't contain a lot of very basic facts which are captured by the old features. The only intersection of the old and the new features is the number of vertices in the TSP problem instance.



```
  a   b      <-- classified as
747 123   |   a = MAOS
111 619   |   b = LKH
```

Fig. 10. 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. Particularly, nn5-wc-std stands for the number of weakly connected components after kNN undirected transformation with $k = 5$. The second level feature to the left is a normalized number of weakly connected components for $k = |V|^{1/3}$ (where $|V|$ is the number of vertices). The second level feature to the right is the median size of strongly connected components for $k = 7$. Note, that actual values on the edges are left out for brewity.

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 10. 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 convenient algorithm.

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. The number of tour intersections in plane was selected in datasets for $firstfinish$ time point. Distance to convex hull contour and number of (best) improving steps appeared in the datasets as well. Interesting observation is that the portfolio of selected features changes with the considered time point. To sum it up, the feature selection process indicates that the highest contribution comes from the kNN transformation, lengths of convex hull edges and features related to tour segments.

6 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 technique which characterizes the problem instance by features and determines the best solver by machine-learning algorithms. We used two solvers – LKH [5] and MAOS [6] –, which, to best of our knowledge, are the state-of-the-art algorithms 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 two thousands 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 kNN graph transformation features and tour segments' features (see Section 3) to be especially helpful for improvements of algorithm selection accuracy. These two categories of features are the most important ones 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 researchers.

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 Bayes networks outperform the other classifiers. The feature selection was beneficial mainly for datasets with almost 400 features.

We also studied our problem with regard to different points in time. Despite some differences in prediction accuracy, the experiments indicate that the hardness of prediction at chosen time points is relatively similar. However, the importance of features depends on the time point where they are used for prediction.

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. We also want to assess our approach on a larger set of instances. 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.

7 Acknowledgements

This research was supported by the Austrian Science Fund (FWF): P24814-N23. The first author of this paper is financially supported by the Vienna PhD School of informatics [3].

References

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

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

2. Smith-Miles, K.A.: Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Computing Surveys **41**(1) (Dec 2008) 1–25
3. 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)
4. Laporte, G.: A concise guide to the traveling salesman problem. Journal of the Operational Research Society **61**(1) (January 2010) 35–40
5. Helsgaun, K.: An effective implementation of the LinKernighan traveling salesman heuristic. European Journal of Operational Research **126**(1) (October 2000) 106–130
6. 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
7. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling-salesman problem. Operations Research **21**(2) (March 1973) 498–516
8. Cook, W.: National traveling salesman problems (April 2009)
9. Hutter, F., Xu, L., Hoos, H.H., Leyton-Brown, K.: Algorithm runtime prediction: Methods & evaluation. Artificial Intelligence **206** (2014) 79–111
10. Smith-Miles, K., van Hemert, J., Lim, X.Y.: Understanding TSP difficulty by learning from evolved instances. In: Learning and intelligent optimization. Springer (2010) 266–280
11. 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 Mathematics and Artificial Intelligence (March 2013)
12. Fukunaga, A.S.: Genetic algorithm portfolios. In: 2000. Proceedings of the 2000 Congress on Evolutionary Computation. Volume 2., IEEE (2000) 1304–1311
13. Kanda, J., Carvalho, A., Hruschka, E., Soares, C.: Using meta-learning to classify traveling salesman problems. In: 2010 Eleventh Brazilian Symposium on Neural Networks, IEEE (2010) 73–78
14. 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
15. Kotthoff, L.: Algorithm selection for combinatorial search problems: A survey. CoRR **abs/1210.7959** (2012)
16. Karp, R.: Reducibility among combinatorial problems. In Miller, R., Thatcher, J., eds.: Complexity of Computer Computations. Plenum Press (1972) 85–103
17. Papadimitriou, C.H.: The euclidean travelling salesman problem is NP-complete. Theoretical Computer Science **4**(3) (June 1977) 237–244
18. Reinelt, G.: Tsplib. `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/` (January 2003) Acessed: 2014-01-21.
19. Baghel, M., Agrawal, S., Silakari, S.: Survey of metaheuristic algorithms for combinatorial optimization. International Journal of Computer Applications **58**(19) (November 2012) 21–31
20. Hutter, F., Xu, L., Hoos, H.H., Leyton-Brown, K.: Feature computation code for TSP. `http://www.cs.ubc.ca/labs/beta/Projects/EPMs/TSP_features_UBC2012.tar.gz` Accessed: 2014-01-13.
21. Rohe, A.: VLSI data sets. `http://www.math.uwaterloo.ca/tsp/vlsi/` (May 2013) Accessed: 2014-01-21.
22. McGeoch, L.: Instance generating code for DIMACS TSP challenge,. `http://dimacs.rutgers.edu/Challenges/TSP/codes.zip` Accessed: 2014-01-13.

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. Kononenko, I.: On biases in estimating multi-valued attributes. In: IJCAI. Volume 95. (1995) 1034–1040
25. Tange, O.: Gnu parallel-the command-line power tool. login: The USENIX Magazine (2011) 42–47
26. He, H., Garcia, E.A.: Learning from imbalanced data. Knowledge and Data Engineering, IEEE Transactions on **21**(9) (2009) 1263–1284