Improving the State of the Art in Inexact TSP Solving using Per-Instance Algorithm Selection

Lars Kotthoff[1], Pascal Kerschke[2], Holger Hoos[3], and Heike Trautmann[2]

[1] Insight Centre for Data Analytics, Ireland, lars.kotthoff@insight-centre.org
[2] University of Münster, Germany, {kerschke,trautmann}@uni-muenster.de
[3] University of British Columbia, Canada, hoos@cs.ubc.ca

Abstract. We investigate per-instance algorithm selection techniques for solving the Travelling Salesman Problem (TSP), based on the two state-of-the-art inexact TSP solvers, LKH and EAX. Our comprehensive experiments demonstrate that the solvers exhibit complementary performance across a diverse set of instances, and the potential for improving the state of the art by selecting between them is significant. Using TSP features from the literature as well as a set of novel features, we show that we can capitalise on this potential by building an efficient selector that achieves significant performance improvements in practice. Our selectors represent a significant improvement in the state-of-the-art in inexact TSP solving, and hence in the ability to find optimal solutions (without proof of optimality) for challenging TSP instances in practice.

1 Introduction

The travelling salesman problem (TSP) is arguably the most prominent NP-hard combinatorial optimisation problem. Given a set of n locations – which, by convention, are called *cities* – and pairwise distances between those cities, the objective in the TSP is to find the shortest round-trip or *tour* through all cities, i.e., a sequence in which every city is visited exactly once, except for the last city, which is the same as the first, and the sum of the distances between successively visited cities along the tour is minimal. Here, we consider the *2D Euclidean TSP* in which the cities correspond to points in the Euclidean plane and the distances between them are simply the Euclidean distances between those points. This is the most commonly studied special case of the TSP, and, like the general TSP, it is known to be NP-hard. The Euclidean TSP has important applications (e.g., in the fabrication of printed circuit boards) and also arises in the context of various transportation and logistics applications.

There are two types of TSP algorithms: exact algorithms, which are guaranteed to find an optimal solution to any TSP instance and, when run to completion, produce a proof of optimality; and inexact algorithms, which cannot guarantee or prove the optimality of the solutions found. Intriguingly, the state of the art for both types of algorithms has been defined by a single solver each for many years: the exact solver Concorde [1] and the inexact solver LKH [4]. Furthermore, LKH typically finds high-quality and even optimal solutions much more quickly than Concorde, and therefore, for the purpose of finding such solutions, per-instance algorithm selection techniques (see, e.g., [8]) were inapplicable to the TSP.

Recently, however, an improvement in the state of the art in inexact TSP solving in the form of a new evolutionary algorithm, EAX, has been reported [13], and from the performance comparison against LKH, it appeared possible that per-instance selection between those two solvers might yield further improvements.

In this work, we pursue this possibility and show, for the first time, that per-instance algorithm selection techniques can be used to improve the state of the art in inexact TSP solving. After providing some preliminary information about the TSP solvers, benchmark instances and algorithm selection techniques we use in our study in Section 2, we report performance results for LKH and EAX that clearly indicate the potential benefit of per-instance algorithm selection (Section 3). Next, we report the performance that can be obtained from actual algorithm selectors, using broad sets of instance features from the literature [6, 12, 15, 19] (Section 4). Finally, we demonstrate how an effective selector can be constructed based on a small number of efficiently computable probing features extracted from the initial phase of EAX runs (Section 5), before concluding with some general observations and directions for future work.

2 Background and Experimental Setup

TSP Solvers. We consider two state-of-the art inexact TSP solvers in this work: LKH [4] and EAX [13].

LKH is a stochastic local search algorithm based on the Lin-Kernighan procedure. It uses an improved variant of the Lin-Kernighan algorithm, based on 5-exchange moves in combination with a construction procedure loosely related to the nearest neighbour heuristic. LKH has defined the state of the art in inexact TSP solving since it was first introduced in 2000.

Besides the reference implementation of LKH, we used a modification of version 1.3, developed in the context of a study of LKH's scaling behaviour [9].[4] This modification adds a simple dynamic restart mechanism to the original LKH algorithm, based on the observation that the performance of the former suffered frequently from stagnation of the underlying stochastic search process. We dub this variant LKH+restart.

EAX is a recently introduced evolutionary algorithm for inexact TSP solving. Its key ingredient is a new edge assembly crossover procedure, which obtains high-quality tours by combining edges from two parent tours with a small number of new, short edges. EAX uses 2-opt local search to determine the initial population, as well as a specific tabu search procedure for generating offspring from very high-quality parent solutions. Furthermore, an entropy-based mechanism is used to preserve diversity in the population of candidate solutions. A rather complex combination of termination criteria is used to determine when a run of EAX is ended, at which point the best tour encountered during the run is returned. Nagata and Kobayashi [13] provide empirical evidence that EAX often, but not always, outperforms LKH on several sets of commonly studied Euclidean TSP instances in terms of the solution qualities reached within similar or shorter running times.

[4] A similar modification can in principle be applied to the current version 2.0.3 of LKH, but as we will see, the performance of version 1.3, for which the modification was made available to us, is sufficient to obtain better performance than EAX in many cases.

We modified the official implementation of EAX to permit setting the random seed (which had previously been fixed to one value) and to terminate when a given solution quality or bound in running time is reached (or exceeded). These modifications were necessary to facilitate our comparative performance analysis and did not compromise performance. During initial experiments, we noticed that EAX often terminates prematurely. We therefore created two variants, which we studied in the following. The first, simply dubbed EAX, disables the original termination criterion and ends a run *only* when a given solution quality or bound in running time is reached (or exceeded). We verified that single runs of this variant performed no worse than the original version of EAX. Our second variant uses the original termination criterion to trigger a restart, by initialising another run; this is done until a given solution quality or bound in running time is reached (or exceeded). We dub this variant EAX+restart.

Benchmark instances. Consistent with other work in this area, we use four types of benchmark instances.

Random uniform Euclidean (RUE) instances are obtained by placing n points uniformly at random in a square, with integer coordinates between 1 and 1 000 000; each point corresponds to a city to be visited. Distances between these cities are defined as Euclidean distances between the respective points, rounded to the nearest integer. We generated instances with 1 000, 1 500, and 2 000 cities, 1 000 each. After filtering the instances that no solver could solve within 1 CPU hour on our reference machine and instances for which features could not be computed because the computation ran out of memory, we were left with 999 instances with 1 000 cities, 1 000 with 1 500 cities, and 998 with 2 000 cities. The RUE instances used in our experiments were generated using the `portgen` generator from the 8th DIMACS Implementation Challenge. Optimal solution qualities for all RUE instances were obtained using Concorde [1].

TSPLIB is a widely used collection of TSP instances with different characteristics, including instances from various applications of the TSP. In our experiments, we used 74 instances with edge types EUC 2D, CEIL 2D and ATT and sizes between 48 and 11 849. Again we excluded instances that no solver was able to solve within 1 CPU hour on our reference machine and instances for which we were unable to compute features.

Finally, we used two sets of instances from the TSP webpage at `http://www.math.uwaterloo.ca/tsp/index.html`. The *National* instances are based on the locations of cities within different countries, and we used 8 National instances with 734 to 9 882 cities. The *VLSI* instances stem from an application in VLSI circuit design, and we used 27 VLSI instances with 662 to 2 924 cities. These instances are known to be particularly hard for many TSP solvers, including Concorde and EAX.

We limited our study to instances for which the optimal solution is known, since we were interested in the ability of our solvers to find optimal solutions and in the time required for doing so. This is the most ambitious goal for any TSP solver, and even though inexact solvers, such as the ones we consider here, cannot prove optimality, they are typically able to find solutions whose optimality is later proven using other methods much more effectively than the best exact solvers.

Automated algorithm selection. The per-instance algorithm selection problem [16] involves selecting from a set of candidate algorithms the one expected to perform best on a given problem instance. It is relevant where algorithm portfolios [3, 5] are employed – instead of tackling a set of problem instances with just a single solver, a set of them is used with the best being selected for each instance.

Algorithm selection systems build performance models of the algorithms or the portfolio they are contained in to forecast which algorithm to use in a particular context. Usually, these models are induced using machine learning. Using the model predictions, one or more algorithms from the portfolio are selected to be run sequentially or in parallel.

Here, we consider the case where exactly one algorithm is selected for solving the problem. One of the most prominent and successful systems that employs this approach is SATzilla [20], which defined the state of the art in SAT solving for a number of years. Since then, additional algorithm selection systems have been developed and proved their worth in the annual SAT competition (e.g. CSHC [10], which has also been applied to MaxSAT). Other successful application areas have been constraint solving [14], continuous black-box optimization [2, 11], mixed integer programming [21], and AI planning [18].

The interested reader is referred to a recent survey [8] for additional information on algorithm selection.

Construction and evaluation of algorithm selectors. In the following, we use the LLAMA algorithm selection toolkit [7], version 0.7.2, to build algorithm selectors for the TSP and consider a range of different approaches to algorithm selection used in the literature. We build models that treat algorithm selection as a classification problem and predict the algorithm to use. We furthermore build models that use regression to predict the performance of the individual algorithms in the portfolio separately and choose the algorithm with the best predicted performance. Finally, we consider models that, for each pair of algorithms, use regression to predict the performance difference between them. The solver with the largest performance improvement over all other algorithms is chosen.

In addition to a range of algorithm selection models, we also consider a range of different machine learning techniques. For classification, we use C4.5 decision trees (J48), random forests (RF), and recursive partitioning trees (RPART). For regression, we consider random forests (RF), support vector machines (KSVM), and multivariate adaptive regression spline (MARS) models. All machine learning models were used with their default parameters.

We generally consider the portfolio that contains all four solvers – LKH and EAX as well as their respective restart variants. From our original set of instances, we selected all that at least one of these solvers was able to find the optimal solution for within the specified cutoff time. We also filter instances for which we were unable to compute feature values because the computation ran out of memory or unsupported constructs in the input. This leaves us with a total of 3 106 instances.

We use 10-fold cross-validation to determine the performance of the algorithm selection models. The entire set of instances was randomly partitioned into 10 subsets of approximately equal size. Of the 10 subsets, 9 were combined to form the training set for the algorithm selection models, which were evaluated on the remaining subset. This process was repeated 10 times for all possible combinations of training and test sets. At the end of this process, each problem instance in the original set was used exactly once to evaluate the performance of the algorithm selection models.

Execution environment and performance measurement. All experiments were run on 24-core 2.5 GHz Intel XEON machines with 64 GB of RAM running CentOS 6.4 64 Bit. We measured execution times using the `time` command and limited the CPU time of solvers with the `runsolver` tool [17] where necessary. We set the cutoff time to 3 600 CPU seconds. We ran each solver 10 times on an instance with different random seeds and took the median of the results.

The mean PAR10 score over all instances is 2 062.145 for LKH, 422.477 for LKH+ restart, 11 462.98 for EAX, and 104.014 for EAX+restart. The PAR10 score is the penalized average runtime. That is, if the solver chosen for the respective instance was able to solve it within the cutoff time of 1 hour, the actual runtime is the score. Otherwise, we penalise the solver by multiplying the cutoff time by a factor of 10.

3 Potential for Portfolios

Figure 1 shows scatter plots of the CPU times on our benchmark sets of TSP instances for the four inexact TSP solvers we considered in our study. It is obvious that there is substantial potential for algorithm selection – the solvers show very different behaviour on different sets of instances. There are many instances with large performance differences; in particular, many instances are easily handled by one solver, while the other times out after an hour.

The RUE instances (triangles), which comprise the vast majority of our instance set, are clustered in the centers of the plots – most of them can be solved by all solvers, and often there are only small performance differences. Nevertheless, there are a few instances that at least one of the solvers cannot solve within the time limit of 1 CPU hour.

The TSPLib instances are more varied. While most of them are easily solvable within a few seconds by all solvers, a few are very hard for one solver, but easily solvable by another. The VLSI and National instances are in between very easy and very hard.

The left hand side of Figure 1 shows that there is a large set of instances that EAX is unable to solve within the time limit. However, the right hand side, which compares the restart variants of the solvers, shows that EAX+restart is able to solve the vast majority of these instances within the time limit. This suggests that EAX+restart effectively improves over plain EAX; further analysis of the performance correlation between the two variants indicates potential for automated selection between those. Similar observations apply to LKH *vs* LKH+restart.

While in general solving times tend to increase with instances size, the solver behaviour is not completely consistent with the size of the problem instances. For example, there is a large number of relatively small instances on which EAX times out after

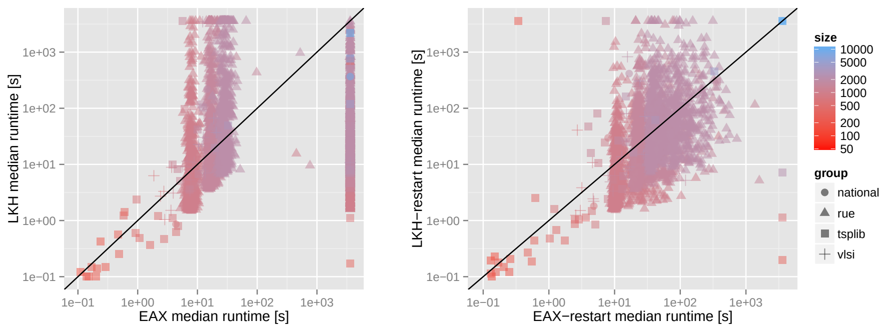

Fig. 1. Performance differences for EAX and LKH (left) and the respective restart variants (right). Each point represents a problem instance. The axes show the CPU time consumed by the respective solver as the median over 10 runs on a log scale. Both solvers exhibit the same performance for points on the diagonal line. The points at the top and right of the plots represent instances on which one of the solvers timed out, the instances in the top right corner could not be solved by either of the solvers.

an hour. Similarly, there are small instances where LKH exhibits the same behaviour. This suggests, consistent with earlier work on performance modelling of TSP solver performance (e.g. [6]) that more information is required to forecast solver behaviour.

4 Building Algorithm Selectors using Features from the Literature

There are several approaches in the literature that attempt to characterise TSP instances by computing features. We focus on the two presented in [12][5] and [6][6], as they comprise a large set of syntactic and dynamic features and consider them in isolation as well as combined with each other. As mentioned above, the cost of computing the feature values can play a major part in the success of an algorithm selection system. We therefore split the feature set described in [6] further into relatively cheap features and the full set of features that in addition comprises more expensive characteristics and ones that are computed through probing.

We denote the feature set described in [12] **TSPmeta** and the one from [6] **UBC**. Based on these, we use the following four sets of features in our experiments.

UBC (cheap) The feature set from [6] without the more expensive features, in particular, the local search, branch and cut, and clustering distance features (13 features). The mean time of computing this set of features was 0.975 seconds per instance, with the median at 0.97 seconds (standard deviation 0.423).

UBC The full feature set from [6] (50 features). The mean time of computing this set of features was 20.71 seconds per instance, with the median at 16.47 seconds (standard deviation 46.355).

[5] `http://cran.r-project.org/web/packages/tspmeta/index.html`

[6] `http://www.cs.ubc.ca/labs/beta/Projects/EPMs/TSP_features_UBC2012.tar.gz`

		UBC (cheap)	UBC	TSPmeta	UBC ∪ TSPmeta
virtual best		18.521			
single best		104.014			
classification	J48	3077.424	3725.066	3773.809	3542.362
	RF	2676.616	2176.886	2312.164	2252.694
	RPART	1931.505	1580.833	1628.554	1612.98
regression	RF	119.964	126.398	151.198	158.14
	MARS	**95.876**	223.232	204.229	204.974
	KSVM	295.762	911.488	3906.038	2140.112
regression pairs	RF	144.482	139.348	151.503	170.332
	MARS	**95.076**	138.866	208.207	205.855
	KSVM	345.475	850.063	1733.452	1948.489

Table 1. Summary of algorithm selector results using sets of features from the literature. The numbers represent mean PAR10 scores, *including the cost of feature computation*, over the entire set of instances and rounded to three digits. We show the scores for the virtual best and single best solver for comparison. The scores for the models that are better than the single best algorithm are shown in **bold face**.

TSPmeta The full feature set from [12] (64 features). The mean time of computing this set of features was 33.61 seconds per instance, with the median at 28.51 seconds (standard deviation 39.469).

UBC ∪ **TSPmeta** The union of **UBC** and **TSPmeta** (114 features). Some of the features in the constituent sets contain the same information.

An additional set of features based on k-nearest neighbour analysis has been introduced very recently in [15]. These features will be included in future studies.

4.1 Results

The results we achieve with the feature sets described above are detailed in Table 1 (we report PAR10 scores over the union of our four benchmark sets).

We are able to improve upon running the single best solver (EAX+restart) only in two cases overall. All other selectors are (sometimes much) worse than simply choosing the single best solver statically. In particular, the classification-based models exhibit very bad performance. The regression-based models perform much better, in particular, the random forest and MARS models.

To what extent these results are caused by the cost of computing the features becomes clear when examining the results that ignore this cost, presented in Table 2. While the differences for the classification-based models are relatively small, there are major changes for the random forest and MARS regression models.

The cost of computing the probing features can be substantial; this can be seen, e.g., when comparing the performance of the random forest regression model with the TSP-meta feature set without costs (118.565) with the performance including the overhead

		UBC (cheap)	UBC	TSPmeta	UBC ∪ TSPmeta
virtual best		18.521			
single best		104.014			
classification	J48	3076.546	3696.603	3734.9	3495.625
	RF	2675.727	2148.116	2271.907	2194.633
	RPART	1930.594	1551.948	1597.611	1553.285
regression	RF	118.998	106.404	118.565	105.995
	MARS	**94.907**	192.358	171.744	152.899
	KSVM	294.803	892.429	3867.598	2069.934
regression pairs	RF	143.519	119.858	118.87	118.187
	MARS	**94.104**	107.925	175.715	154
	KSVM	344.517	831.006	1702.978	1877.953

Table 2. Summary of algorithm selector results using features from the literature. The numbers represent PAR10 scores over the entire set of instances *without taking the cost for feature computation* into account and rounded to three digits. We show the scores for the virtual best and single best solver for comparison. The scores for the models that are better than the single best algorithm are shown in **bold face**.

(151.198). The average cost of computing this feature set is almost twice as large as the average PAR10 score of the virtual best solver.

Figure 2 (right) shows the performance of the best overall model, MARS regression on pairs of solvers trained using the UBC (cheap) feature set, compared to the single best solver. There is a large number of instances where the solver the selector chooses is better than the single best (points below the diagonal); in particular, there are 3 instances where the single best solver times out (right margin of plot), while the selector chooses a solver that does not. There are, however, a significant number of instances where the choice made by the selector is incorrect, and EAX+restart exhibits better performance than the chosen solver. In particular, there are two instances that are easy for the single best solver, while the solver chosen by the selector times out (top margin of plot). Unsurprisingly, as can be seen when comparing the left and right plots in Figure 2, the cost of feature computation mainly affects selector performance on easy instances.

Additionally, we performed forward feature selection, where we start with an empty set and repeatedly add the feature that gives most additional information, based on entropy and correlation on the full feature set UBC ∪ TSPmeta to determine the features that are most important for determining the solver to run. No cost-sensitive feature selection strategy was applied (we plan to improve on this approach in future work). The resulting set included eight features from [6] (the mean and standard deviation cluster distances, the average tour cost from the construction heuristic, the skew of the probability of edges in local minima, the time required for the local search probing feature computation, the maximum depth, the median and standard deviation of the distances of the minimum spanning tree) and one from [12] (the fraction of nodes on the convex hull).

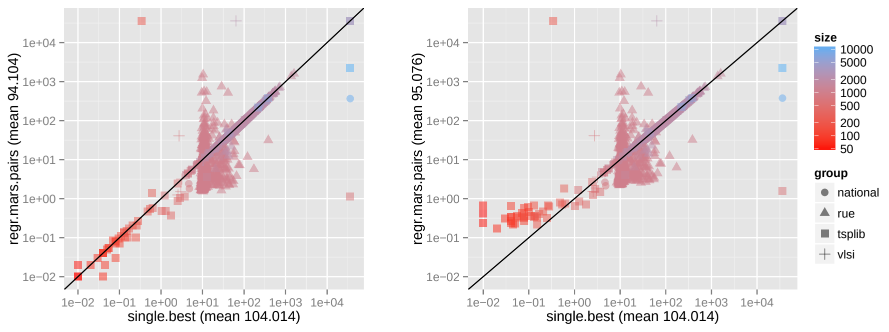

Fig. 2. Algorithm selector performance for the best model trained with features from the literature without and with taking feature costs into account (left and right plot, respectively) – in both cases, the best model was MARS regression on pairs of solvers with the UBC (cheap) feature set. The x-axis shows the log PAR10 score of the single best solver, the y-axis the log PAR10 score of the selector. Each point represents a TSP instance. Points on the diagonal indicate that the selector chose the single best solver, below the diagonal that the selector chose a better solver than the single best.

We also performed feature selection on the features used by the best overall model, MARS on pairs of solvers with the UBC (cheap) feature set. Just a single feature was chosen, the average length of the minimum spanning tree.

While feature selection was able to improve the performance slightly in some cases, selectors trained on the reduced feature set showed worse performance in other cases. There is significant overlap in the type of features computed in the UBC and TSPmeta feature sets, which may explain the inconsistent results we achieved with feature selection. All results reported in this paper are without feature selection, as feature selection does not significantly and consistently improve the results and increases the conceptual complexity of selector construction.

5 Building Algorithm Selectors using EAX Probing Features

In the previous sections, we have shown that there are significant complementarities in performance between the four solvers we consider and therefore significant potential for algorithm selection to improve the current state of the art in TSP solving. Using features described in the literature, we can already achieve a significant performance improvement over the single best solver on our set of instances. In this section, we investigate whether we can improve on this by using a different, novel set of features.

As explained earlier, there is a trade-off between the cost of computing the features characterising a TSP instance and the information obtained through them. In particular, computing the features that cannot be determined directly from the description of the instance itself is expensive, but does help learn better algorithm selection models.

In this section, we propose a new set of features that allows us to investigate the trade-off of cost of feature computation vs. information in a much more fine-grained and principled manner. We harness one of the solvers from our portfolio and analyse its progress when run for a small amount of time. We can control the amount of time directly – the longer the solver is run, the more information we get, but the more expensive the feature computation becomes. This information is then used to derive novel features.

Our single best solver, EAX+restart, provides the user with a trace of its execution as it progresses through the different generations. For each generation, the evolutionary algorithm outputs the best and average tour length found over the individuals of the current population. This gives an indication of how the solver progresses. By comparing the tour lengths of successive generations to the initial one, we get information on how quickly the solver is able to improve on initial solutions.

We consider the information obtained during the first n generations. Each best and average tour length is normalised by the best and average tour lengths of the initial population to obtain the improvement over these. We compute the minimum, maximum, mean, and median of both best and average improvements over the n generations. As the solving trajectory varies between different executions, we compute the median values of these numbers over m runs of EAX+restart with different random seeds.

This feature computation can be seen as a pre-solving step, during which we are running the actual algorithm used to find a solution. If the solver finds the solution during the first n generations, no further work needs to be done. Presolving is an effective means of quickly solving easy instances without incurring the overhead of feature computation costs. It is used with great success in the SATzilla system [20] for example.

5.1 Determining the Number of Generations and Probing Runs

We first investigated the impact of the parameters n and m on selector performance. The results of these preliminary experiments were somewhat inconclusive, but led us to choose $n = 10$ generations and $m = 1$ algorithm run for computing our probing features. This keeps the cost of feature computation low, while still providing us with valuable information that can be used effectively to decide which solver to use.

The results vary not only with n and m, but also between different probing runs. As our probing algorithm is stochastic, we obtain different feature values for different random seeds. The resulting performance differences can be quite high, especially for easy instances that are solved almost instantaneously if the solver starts its search process with a good set of initial tours. This means that not only the computed feature values, but also the cost of feature computation is different for different runs. This introduces additional stochasticity and noise into our evaluation.

We therefore average feature costs and values over 10 independent algorithm runs with different random seeds, and the results reported below are averages over those runs. In each of these runs, we extract the features as described above and build and evaluate the models. Averaging the results in this manner makes our conclusions statistically more robust.

The mean cost of computing this set of features (mean over all instances that are not solved during feature computation, and median over 10 independent runs per instance) is 2.811 seconds, and the mean number of presolved instances over all independent runs is 26.6, all from TSPLIB.

5.2 Results

In the subsequent evaluation, we focus on the approaches that we have identified as the most promising in the previous experiments, namely random forest and MARS models for regression and regression on pairs of algorithms. Table 3 shows the results we were able to achieve with selectors using only our new features. The overall best model is random forest regression and achieves better performance than the single best solver on average.

We note that the performance of the virtual best solver is very slightly worse than that observed in our experiments from Section 4, although the difference is less than the three significant digits we round to. This happens, because for the instances that are solved during feature computation, we take the runtime of the solver used to compute those features, even though a different solver may be faster.

The selector performance obtained using our new models is worse than the single best solver when taking into account the full cost of feature computation; however, because of the nature of our new probing features, this is not necessary: If the solver used for the feature computation is chosen as the solver to be run on the given TSP instance, the features are obtained at no additional cost, by simply continuing the probing run. The performance results for this 'accelerated' feature computation are shown in the third column of Table 3.

On average over all probing runs with different random seeds, the selectors trained using the new features perform worse than the selectors trained using features from the literature. However, there are clear indications for potential to obtain much better performance. In Table 3, we report, in parentheses, the first quartiles of the distributions of mean PAR10 scores over the 10 independent runs per TSP instance. According to these results, the MARS models for pairs of solvers, using our new probing features, can yield better performance than any of the models we have studied previously, using instance features from the literature.

The performance variation between the 10 independent runs underlying the results in Table 3 is quite high, considering the relatively small difference in performance to the single best algorithm; for our accelerated random forest models and our accelerated MARS models for pairs of solvers, we observe standard deviations of 21.767 and 28.085, respectively. The best performance achieved over the 10 independent runs is up to $\approx$30% better than that of the single best solver. While these results indicate the potential inherent in our new probing features, statistically robust ways to exploit this potential will be investigated in future work.

Figure 3 illustrates the performance of our new algorithm selectors, based on EAX probing features, in more detail. In contrast to the situation when using features from the literature, illustrated in Figure 2, we are now able to match the performance of the single best solver for the vast majority of easy instances. This is in part due to the fact that the very easy instances are now solved during feature computation. Furthermore, there are no more cases where our selector chooses a solver that times out while the single best solver does not. On the contrary, there are three instances where the single best solver times out, but our selector chooses a solver that does not. This fact further illustrates the potential of our new probing features, which enable us to make better predictions, especially in extreme cases, where incorrect decisions are particularly detrimental.

		without costs	with costs	accelerated
virtual best			18.521	
single best			104.014	
regression	RF	**103.415** (**95.041**)	106.238 (**97.859**)	**103.833** (**95.461**)
	MARS	126.204 (116.929)	129.017 (119.733)	126.529 (117.235)
regression pairs	RF	128.739 (119.761)	131.561 (122.584)	129.282 (120.273)
	MARS	107.13 (**85.909**)	109.949 (**88.735**)	107.506 (**86.303**)

Table 3. Summary of algorithm selector results using our new EAX probing features. The numbers represent the mean of the mean PAR10 scores over the entire set of instances (including the ones solved during feature computation) and 10 independent runs per instance, rounded to three digits. The numbers in parentheses represent the first quartiles over ten independent runs. The 'accelerated' column denotes the average PAR10 score where the cost of computing the features was added only if the chosen solver was different from the one used for computing those features. We show the scores for the virtual best and single best solver for comparison. The scores for the models that are better than the single best algorithm are shown in **bold face**.

When comparing the left- and right-hand plots in Figure 3, we see the impact of the feature computation costs. There is no difference in the top and right-hand parts of the plots, as the instances in these areas take longer to solve, and the time for feature computation is insignificant. In the centre part, however, a small shift of points towards the top of the plot can be observed – there is no shift to the right, as the single best solver can be determined statically and does not require features. Easy instances are not affected as much, as the solver used to compute the probing features is also chosen as the solver to continue solving.

Since our use of EAX probing features effectively combines feature computation with presolving, we see considerable benefits over other algorithm selection approaches for relatively easy instances. With further optimised feature computation and presolving strategies, it should be possible for the selector to focus on improving performance on difficult instances and thus to obtain additional overall performance improvements.

5.3 Combining with Features from the Literature

As we have seen above, our new features have the potential to give rise to better selectors than those obtained by using only features from the literature. For our final set of experiments, we combined the feature sets from the literature with our new EAX probing features to assess whether this could result in even better selectors.

Table 4 shows the performance results for selectors using the combined set of features. Overall, when accounting for the cost of determining the features, performance is worse than for the individual sets in isolation. This is mostly caused by the high cost of feature computation. However, even when ignoring this cost, the selectors do not perform better than before. In particular, while the best selector achieved performance similar to the single best algorithm on average, it appears to be unable to capitalise on

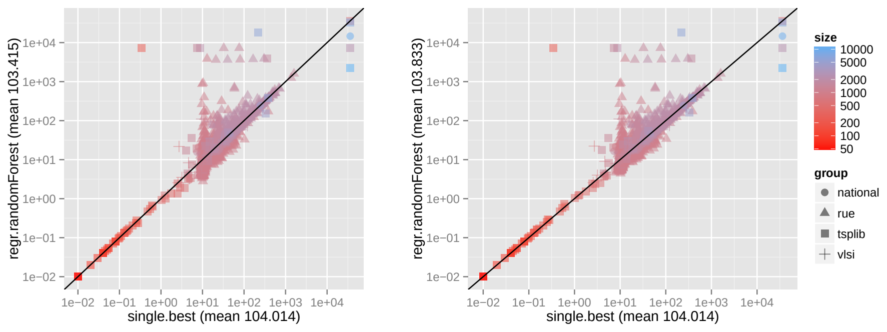

Fig. 3. Algorithm selector performance for the best model trained with the new EAX probing features, random forest regression, without (left) and with (right) feature cost, where in the latter case, the 'accelerated' feature computation method was used. The x- and y- axes show the log PAR10 scores of the single best solver and the selector, respectively. Each point represents one TSP instance. Points on the diagonal correspond to cases where the selector chooses the single best solver, and points below the diagonal to cases where the selector chooses a solver with even better performance for that instance.

the additional information contained in the larger feature set. We believe that the redundant information contained in the set of all features has a detrimental effect on selector performance.

6 Conclusions

The Travelling Salesman Problem is one of the most iconic NP-hard optimisation problems. It has been extensively studied over the years, and many approaches for solving it have been developed. Until recently, a single solver, LKH, has defined the state of the art for inexact TSP solving. With the recent introduction of a new state-of-the-art inexact TSP algorithm, EAX, this picture has changed.

In this work, we have extensively studied the empirical performance of LKH, EAX, and improved variants of these base solvers on a large set of TSP instances ranging from trivial to hard. We have demonstrated the huge potential for algorithm selection in this context. We then successfully applied algorithm selection techniques to improve the state of the art in inexact TSP solving.

On the large set of instances we consider in this paper, we have computed features defined in the literature. We empirically investigated how informative these features are with respect to choosing the best solver for a specific instance. The initial results are very encouraging. Even with features that are relatively cheap to compute, we are able to build algorithm selection models that outperform the current state of the art – the single best solver over the entire set of instances, EAX+restart.

Motivated by this observation, we proposed a new set of features based on information gleaned from the execution trace of one of the solvers in our portfolio. Controlling the trade-off between the amount of information and the cost of computing it, we were

		without costs	with costs	accelerated
virtual best		18.521		
single best		104.014		
regression	RF	**103.894** (**95.928**)	163.162 (161.727)	160.761 (159.32)
	MARS	216.892 (190.686)	277.835 (260.146)	275.732 (257.834)
regression pairs	RF	125.729 (119.585)	180.626 (174.478)	178.291 (172.132)
	MARS	159.126 (145.451)	221.648 (210.952)	219.412 (208.644)

Table 4. Summary of algorithm selector results using the combined set of all features from the literature and our own. The numbers represent the mean of the mean PAR10 scores over the entire set of instances (including the ones solved during feature computation) and all 10 random seeds rounded to three digits. The numbers in parentheses are the first quartiles over 10 independent runs. The 'accelerated' column denotes the average PAR10 score where the full cost of computing the features was added only if the chosen solver was different from the one used for computing those features. If the same solver was chosen, only the cost for the features not derived during the probing run was added. We show the scores for the virtual best and single best solver for comparison. The scores for the models that are better than the single best algorithm are shown in **bold face**.

able to show that the quality of the selector can improve significantly over selectors that use existing features. Our approach to feature computation combines the extraction of instance characteristics with presolving, which has the additional benefit that trivial instances are solved during this phase and the selector does not have to consider them.

In future work, we will further investigate our new EAX probing features, with the goal of obtaining additional, statistically robust performance improvements. We will also endeavour to add additional features and investigate the impact of cost-sensitive feature selection methods.

Acknowledgements. We thank Thomas Stützle for letting us use the restart version of LKH 1.3 he implemented in the context of a different project and for helpful comments on earlier versions of this work. Holger Hoos acknowledges support from an NSERC Discovery Grant. Lars Kotthoff is supported by EU FP7 FET project 284715 (ICON). Pascal Kerschke and Heike Trautmann acknowledge support from the European Center of Information Systems (ERCIS).

References

1. Applegate, D.L., Bixby, R.E., Chvatal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study. Princeton University Press, Princeton, NJ, USA (2007)
2. Bischl, B., Mersmann, O., Trautmann, H., Preuss, M.: Algorithm selection based on exploratory landscape analysis and cost-sensitive learning. In: Proceedings of the 14th annual conference on Genetic and evolutionary computation. GECCO '12, ACM, New York, NY, USA (2012)
3. Gomes, C.P., Selman, B.: Algorithm portfolios. Artificial Intelligence 126(1-2), 43–62 (2001)

4. Helsgaun, K.: General k-opt submoves for the LinKernighan TSP heuristic. Mathematical Programming Computation 1(2-3), 119–163 (2009)
5. Huberman, B.A., Lukose, R.M., Hogg, T.: An economics approach to hard computational problems. Science 275(5296), 51–54 (1997)
6. Hutter, F., Xu, L., Hoos, H.H., Leyton-Brown, K.: Algorithm runtime prediction: Methods & evaluation. Artificial Intelligence 206(0), 79 – 111 (2014)
7. Kotthoff, L.: LLAMA: Leveraging learning to automatically manage algorithms. Tech. Rep. arXiv:1306.1031, arXiv (Jun 2013), `http://arxiv.org/abs/1306.1031`
8. Kotthoff, L.: Algorithm selection for combinatorial search problems: A survey. AI Magazine 35(3), 48–60 (2014)
9. Lacoste, J.D., Hoos, H.H., Stützle, T.: On the empirical time complexity of state-of-the-art inexact tsp solvers, manuscript in preparation
10. Malitsky, Y., Sabharwal, A., Samulowitz, H., Sellmann, M.: Algorithm portfolios based on cost-sensitive hierarchical clustering. In: IJCAI (Aug 2013)
11. Mersmann, O., Bischl, B., Trautmann, H., Preuss, M., Weihs, C., Rudolph, G.: Exploratory landscape analysis. In: Proceedings of the 13th annual conference on Genetic and evolutionary computation. pp. 829–836. GECCO '11, ACM, New York, NY, USA (2011), `http://doi.acm.org/10.1145/2001576.2001690`
12. 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 69(2), 151–182 (2013)
13. Nagata, Y., Kobayashi, S.: A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. INFORMS Journal on Computing 25(2), 346–363 (2013)
14. O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., O'Sullivan, B.: Using case-based reasoning in an algorithm portfolio for constraint solving. In: Proceedings of the 19th Irish Conference on Artificial Intelligence and Cognitive Science (Jan 2008)
15. Pihera, J., Musliu, N.: Application of machine learning to algorithm selection for TSP. In: Fogel, D., et al. (eds.) Proceedings of the IEEE 26th International Conference on Tools with Artificial Intelligence (ICTAI). IEEE press (2014)
16. Rice, J.R.: The algorithm selection problem. Advances in Computers 15, 65–118 (1976)
17. Roussel, O.: Controlling a solver execution with the runsolver tool. JSAT 7(4), 139–144 (2011)
18. Seipp, J., Braun, M., Garimort, J., Helmert, M.: Learning portfolios of automatically tuned planners. In: ICAPS (2012)
19. 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), 87–104 (2011)
20. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla: Portfolio-based algorithm selection for SAT. J. Artif. Intell. Res. (JAIR) 32, 565–606 (2008)
21. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In: RCRA Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion at the International Joint Conference on Artificial Intelligence (IJCAI). pp. 16–30 (2011)