This is a post-peer-review, pre-copyedit version of a paper published in Liefooghe A & Paquete L (eds.) *Evolutionary Computation in Combinatorial Optimization*. Lecture Notes in Computer Science, 11452. EVOCOP 2019: European Conference on Evolutionary Computation in Combinatorial Optimization, Leipzig, Germany, 24.04.2019-26.04.2019. Cham, Switzerland: Springer International Publishing, pp. 99-114. The final authenticated version is available online at: https://doi.org/10.1007/978-3-030-16711-0_7

Rigorous Performance Analysis of State-of-the-art TSP Heuristic Solvers

Paul McMenemy[1,0000-0002-5280-425X], Nadarajen Veerapen[2,0000-0003-3699-1080], Jason Adair[1,0000-0003-0198-9095], and Gabriela Ochoa[1,0000-0001-7649-5669]

[1] Computing Science and Mathematics, University of Stirling, United Kingdom
[2] Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL, F-59000 Lille, France
paul.mcmenemy@stir.ac.uk

Abstract. Understanding why some problems are better solved by one algorithm rather than another is still an open problem, and the symmetric Travelling Salesperson Problem (TSP) is no exception. We apply three state-of-the-art heuristic solvers to a large set of TSP instances of varying structure and size, identifying which heuristics solve specific instances to optimality faster than others. The first two solvers considered are variants of the multi-trial Helsgaun's Lin-Kernighan Heuristic (a form of iterated local search), with each utilising a different form of Partition Crossover; the third solver is a genetic algorithm (GA) using Edge Assembly Crossover. Our results show that the GA with Edge Assembly Crossover is the best solver, shown to significantly outperform the other algorithms in 73% of the instances analysed. A comprehensive set of features for all instances is also extracted, and decision trees are used to identify main features which could best inform algorithm selection. The most prominent features identified a high proportion of instances where the GA with Edge Assembly Crossover performed significantly better when solving to optimality.

Keywords: TSP, Algorithm Selection, EAX, GPX, Performance Analysis

1 Introduction

Despite decades of intense study, the Travelling Salesperson Problem (TSP) sustains its practical and theoretical interest. It has inspired the design of powerful exact and heuristic solvers, able to tackle TSP problems of increasing size in shorter computing time. The objective in the TSP, given a set of n locations (generally called cities) and pairwise distances between them, is to find the shortest round-trip through all cities such that the total length of the trip (a tour) is minimised. Here we consider the most common case of the problem, the 2D symmetric TSP, where cities correspond to points in the Euclidean plane and distances are also Euclidean. The current TSP state-of-the-art exact solver, Concorde [1], remains unbeaten. Concorde has been used to optimally solve instances of several thousand cities and, for fewer than 1 000 cities, does so in very feasible running times. However, there is interest in developing inexact or heuristic

solvers, as they can provide surprisingly good results for large instances in reasonably short amounts of time when compared to obtaining a solution via Concorde. The scenario of heuristic solvers was previously dominated by a single contender for several years: Helsgaun's Lin-Kernighan Heuristic (LKH+IPT) [2, 3]. However, recent evolutionary algorithms using the Edge Assembly Crossover (EAX) [4], as well as hybrid approaches using the Generalised Partition Crossover in concert with LKH (LKH+GPX2), have been shown to match and improve upon LKH+IPT performance in some instances [5].

An outstanding challenge in heuristic optimisation is to understand how to find the most suitable algorithm for a given problem instance or set of instances. Corne and Reynolds [6] introduce the notion of the 'footprint' of an algorithm to indicate how its performance generalises across different dimensions of instance space. Smith-Miles and Lopes [7] followed this by proposing a methodology to determine the relative performance of optimisation algorithms across various classes of instances. Later works by Pihera and Musliu [8], Kotthoff [9], and Kerschke et al. [10] show that per-instance automated algorithm selection techniques can be used to improve the state-of-the-art in inexact TSP solving.

The main goal of this study is to perform rigorous tests comparing the runtimes to optimality of the three previously mentioned TSP heuristic solvers: (i) LKH+IPT; (ii) an evolutionary algorithm with EAX; and (iii) a hybrid evolutionary algorithm combining LKH with GPX2. A comprehensive set of features is then to be used to characterise TSP instances taken from the available benchmark sets and instance generators in the literature, with the aim of identifying specific instance space features which can provide guidance on which solver is most effective in solving to optimality.

2 Methodology

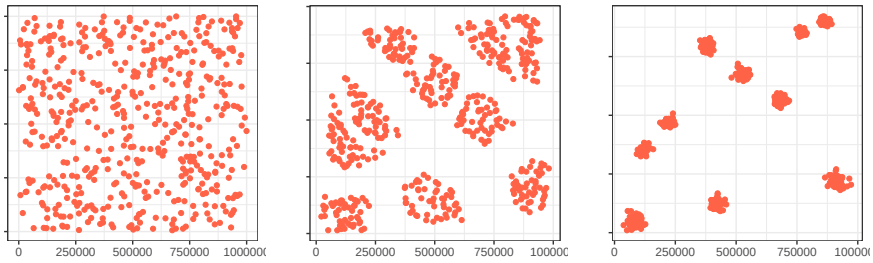
2.1 Instances

For this study 1800 symmetric TSP instances were generated, comprised of varying instance sizes and structures:

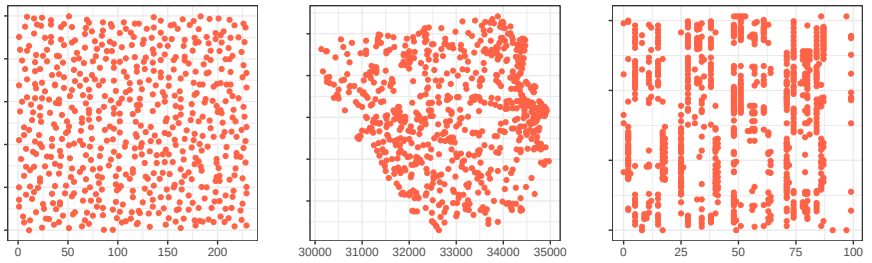
Random Uniform Euclidean (RUE): instances generated by placing a number of points (representing the cities to be visited) randomly within a planar square (e.g., Fig. 1(a)). The distances between the cities are determined as the Euclidean distances between the respective points, where the cost of travel between cities is specified as the Euclidean distance rounded to the nearest whole number. Instances were generated using the DIMACS `Portgen` generator[1] with sizes $n \in \{500, 1\,000, 1\,500, 2\,000\}$. One hundred and fifty seeds of each instance size n were generated for this study.

Random Clustered Euclidean (NETGEN): a set of instances generated with sizes $n \in \{500, 1\,000, 1\,500, 2\,000\}$. Each instance was created with a corresponding parameter, $c \in \{2, 5, 10\}$, which specifies the number of clusters located within the instance by latin hypercube method. City locations

[1] http://archive.dimacs.rutgers.edu/Challenges/TSP/



(a) RUE - 500 (b) MORPH - 500-10 (c) NETGEN - 500-10



(d) LIB - `rat575` (e) World - `uy734` (f) VLSI - `dkg813`

Fig. 1: Instance type problem domains. Plots (a)–(c) are examples of the RUE, MORPH & NETGEN instances, all with $n = 500$. The MORPH example, (b), is constructed from (a) and (c), with $\alpha = 0.5$. Plot (d) shows `rat575`, a rattled grid of 575 locations; plot (e) is the World instance with 734 town/cities in Uruguay (`uy734`); and (f) is the VLSI problem denoted as `dkg813`

are then distributed with respect to the cluster centres, maintaining cluster segregation, Fig. 1(c). Instances were created using the `netgen` software package within the open source software R [11, 12]. Again, 150 instances were generated for each size n, split into 50 instances per combination of c and n.

Morphed Euclidean (MORPH): instances which have been generated from a combination of equal n-sized RUE and NETGEN instances. Pairs of RUE (x_i) and NETGEN (y_i) cities are greedily matched, the first pair (x_1, y_1) being the closest by euclidean distance, with this repeated $\forall i \in \{2, 3, ..., n\}$ until all cities have been matched. Each MORPH city is then generated by relocating a proportional distance (defined by a parameter α) along a straight line from x_i to y_i. Figure 1(b) shows a MORPH instance, generated from a 500-city RUE instance, Fig. 1(a), and 500 city NETGEN instance with 10 clusters, Fig. 1(c). The MORPH examples in this study were generated using the `TSPMETA` package [13] and with $\alpha = 0.5$.

LIB: a widely-used collection of instances with different characteristics and TSP applications [14]. Instances with size $400 \leqslant n < 5\,000$ and edge weight types EUC 2D & CEIL 2D were selected for this study.

World: instances from the World benchmark set[2] which are based on real locations of cities in different countries. Cities with duplicate coordinates were not considered, all instances with size $400 < n < 5\,000$ were selected.

VLSI: a set of instances that originate from an application in very large scale integration (VLSI) circuit design. These instance types are known to be particularly hard for many TSP solvers, including Concorde. Again, all instances with sizes $400 < n < 5\,000$ were included in this study [15].

Only 93 instances of the required size and edge weight types are available, hence the requirement to generate a large, supplemental set of instances. Where appropriate, the combined instances of LIB, World and VLSI will be referred to as TSPLIB.

2.2 Solvers

Our study considers three high-performance heuristic solvers for the symmetric TSP. The three solvers incorporate a form of recombination (crossover) and, according to the literature [10, 16], achieve state-of-the-art performance on Euclidean TSP instances. The first two solvers are modern variants of the powerful Lin-Kernighan-Helsgaun (LKH) heuristic [3].

LKH+IPT & LKH+GPX2: Multi-try LKH variants. LKH is an iterated local search algorithm based on the the Lin-Kernighan heuristic (LK) [17], a variable-depth method that generates complex local search moves by heuristically constructing a sequence of edge exchanges. Over the years, several improvements have been incorporated to LKH. The best results in the literature have been obtained with a version known as multi-trial LKH, where solutions generated by soft restarts of the LK heuristic are recombined using Iterative Partial Transcription (IPT). An alternative version of multi-trial LKH has been recently proposed [16], where the IPT recombination operator is substituted by GPX2, a new generalised partition crossover proposed for the TSP. Both IPT and GPX2 are forms of partition crossover, which are deterministic recombination operators that use features common to the parents to decompose the evaluation function. Following [16] we consider these two versions of multi-try LKH, and name them as LKH+IPT and LKH+GPX2.

EAX: GA with Edge Assembly Crossover. A series of high-performing evolutionary algorithms for the TSP integrate variants of edge assembly crossover, a recombination operator that combines the edges of two parent solutions trying to add only few, short edges not found in any of the two parents [18]. We consider the most recent version of EAX [4], which aggregates three key enhancements: (i) initialisation of the population by local

[2] http://www.math.uwaterloo.ca/tsp/world/countries.html

optimisation, (ii) improved local and global variants of the edge assembly crossover operator, and (iii) specific diversity preservation techniques that use edge entropy measures in the population replacement scheme.

2.3 Experimental Setup

Each of the solvers are implemented as single-threaded programs and were run on Intel Xeon Gold 6138 2.0GHz CPUs[3]. The common termination criterion applied is one hour of CPU time. For the analyses in this study, the PAR10 penalised runtime was implemented [10]. This assigns a penalized runtime of 10 times the termination criterion to runs that fail to solve to optimality within that limit. Each solver is run 30 times on each instance and times are recorded.

For EAX, the parameters used are the ones prescribed by Nagata and Kobayashi [4] for "small" instances ($n < 10\,000$) in the `readme` file accompanying their source code. The population size is therefore set to 100 and the number of offspring set to 30. For LKH, version 2.0.9 is used and the default parameters specified within the source code and the example configuration file are considered. The `RECOMBINATION` parameter is set to either `IPT` or `GPX2` in order to select the appropriate crossover operator.

Neither EAX nor LKH implement a time-based termination criterion but use a fixed number of trials instead. The source code was therefore modified to run within a specific amount of time and allow for consistent comparison between EAX and the two LKH variants.

In order to know whether a solver was successful, the result of each run is compared to the known optimal objective function values for library instances. For generated instances, the state-of-the-art exact TSP solver Concorde [1, 19] was used to generate an optimal solution whose objective function value is used for comparison.

2.4 Instance Features

Two sets of features were calculated and combined together for use in this study:

TSPMETA: a set of features described by Mersmann et al. [13] which provide a group of standard geometric features derived from TSP instances. The features were calculated using the `TSPMETA` package within the R software package [12], with 64 features generated in total.

Pihera: a feature set defined by Pihera & Musliu [8] which are based on kNN graph transformations of each instance and the generation of extensive summary statistics of the kNN graphs. This generated 285 features, mainly comprised of kNN-graph transformation metrics and their summary statistics.

Pihera features were calculated for each of the instance type sets described earlier, with no group taking longer than 30 seconds using the openly available C++ software[4]. The `TSPMETA` features took noticeably longer to calculate,

[3] https://www.archie-west.ac.uk [4] https://tspalgsel.github.io/#software

roughly doubling in execution time as sizes incremented by 500 cities. Both the TSPMETA and the Pihera features were combined to generate a comprehensive feature set of 349 features. This combined group of features improved the likelihood of extracting feature(s) which could best inform heuristic selection.

2.5 Statistical Evaluation of Heuristic Performance

Performance analysis was carried out on the three state-of-the-art solvers described earlier (LKH+IPT, EAX, LKH+GPX2). A round-robin set of statistical comparisons was carried out on pairs of heuristics: i.e., EAX versus LKH+IPT, LKH+IPT versus LKH+GPX2, and LKH+GPX2 versus EAX. Mann-Whitney statistical tests were carried out on these pairs for each TSP instance: the thirty runs of each heuristic pair were tested against each other to determine whether their respective runtimes could be from the same distribution of values. This was carried out at the 99% confidence level, thus ensuring a high level of likelihood that runtimes were identified as being from differing distributions if the Mann-Whitney tests' p-values < 0.01. When this occurred, a further comparison of each heuristic's median runtimes was carried out, and the solver with the lower median runtime value was deemed to be the best of the two tested.

The null hypothesis of the Mann-Whitney test posits that the runtimes of the two solvers being tested against one another are from the same distribution of values and so, if p-values > 0.01, then the null hypothesis could not be rejected. In this circumstance it was deemed that neither heuristic was significantly better than the other at solving the instance to optimality, and so neither was designated as the best heuristic for the particular instance being tested. This outcome was labelled as "ANY" in the following results.

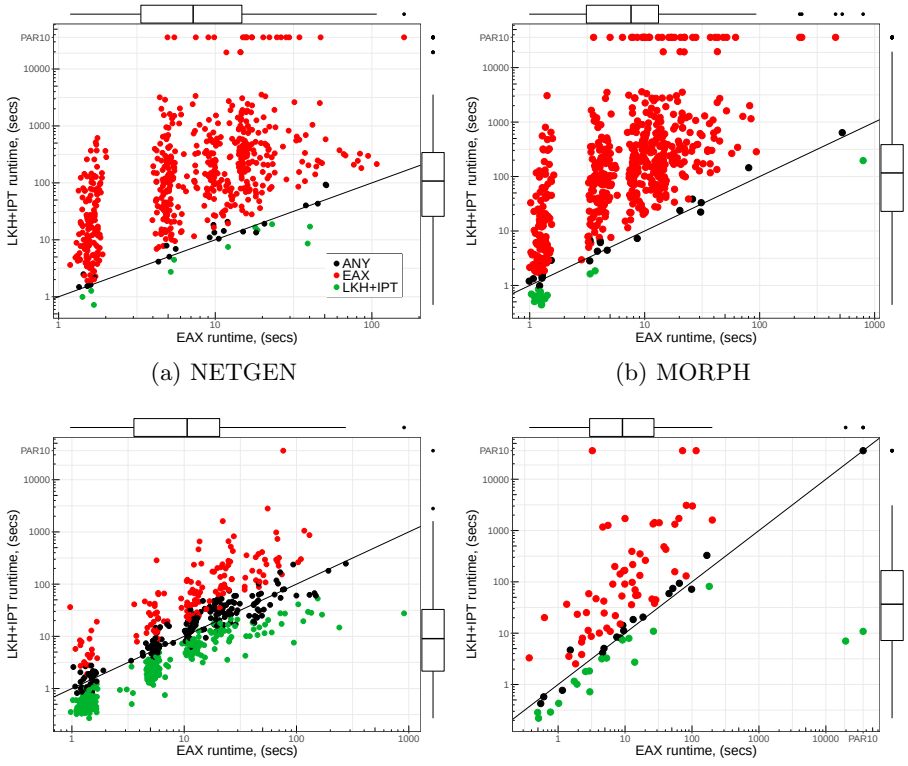
3 Performance Analysis

Runtime performance of each heuristic was carried out using the methodology described in the previous section. Here, results are presented by pairwise combination, i.e., one heuristic is measured against another heuristic's performance. Firstly, scatter plots of the median runtimes per instance are presented, with plots further subdivided by the instance types NETGEN, MORPH, RUE and TSPLIB included in this study. In all plots, red scatter points correspond to when EAX performed significantly better, blue points to LKH+GPX2, and green to LKH+IPT. Black scatter points denote instances where one heuristic was not deemed to be significantly quicker than the other. Diagonal black lines on each plot indicate equality of median runtimes; boxplots of median runtime values are also provided, located adjacent to opposite axes to provide additional comparison of performance.

For each paired comparison, tables showing the numbers and percentages of times of classifications are also presented, with the TSPLIB instances broken down by VLSI, World and LIB instance types. The tables provide further detail that can be occluded in scatter plots by coincidental or clustered data points.

3.1 EAX v LKH+IPT Performance Results

Firstly, the relative performances of EAX versus LKH+IPT are considered, and results are shown in Fig. 2 and Table 1. It can be seen from Fig. 2(a) and (b) that EAX is vastly superior in solving NETGEN and MORPH instance types to optimality when compared with LKH+IPT. The median value of the boxplot for EAX in Fig. 2(a) (which is the overall median value of all EAX runs for NETGEN) is 7.2 seconds (s), whereas the corresponding value for LKH+IPT is 107.7s. Similar values are returned for the MORPH instances, with median EAX runtime = 7.6s and median LKH+IPT = 116.4s.



(a) NETGEN (b) MORPH

(c) RUE (d) TSPLIB

Fig. 2: EAX v LKH+IPT runtime performance results. All plot axes are log-scaled; scatter points are median values of 30 runs of pairwise instances in the form (median(EAX), median(LKH+IPT)); diagonal lines represent equality of median performance; boxplots on axes show distribution of median runtime values

However, for the RUE instances, Fig. 2(c) shows that LKH+IPT greatly improves performance when compared against EAX. Results shown in Table 1 reinforce this result, showing that LKH+IPT solved 41.5% of instances significantly quicker than EAX (median LKH+IPT runtime = 9.0s). Conversely, EAX only solved 23.8% of RUE instances better than LKH+IPT (median EAX runtime 10.6s).

Table 1: EAX versus LKH+IPT classifications

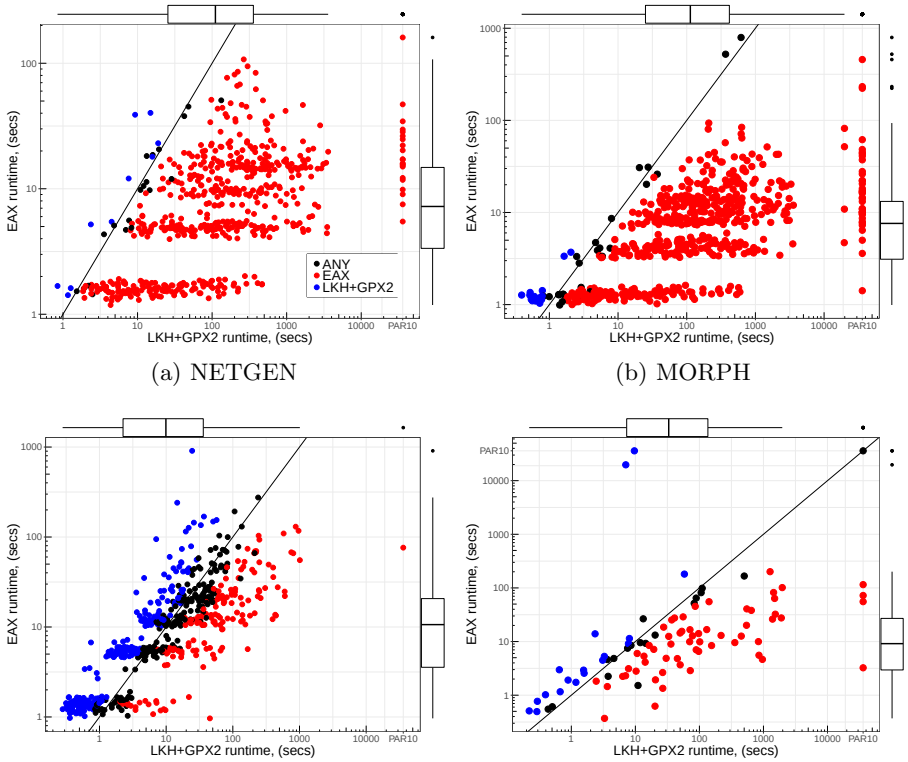
Instance Type	EAX	LKH+IPT	ANY
RUE	143 (23.8%)	249 (41.5%)	208 (34.7%)
MORPH	562 (93.7%)	16 (2.7%)	22 (3.7%)
NETGEN	567 (94.5%)	11 (1.8%)	22 (3.7%)
TSPLIB:			
VLSI	38 (66.7%)	8 (14.0%)	11 (19.3%)
World	3 (100.0%)	0 (0.0%)	0 (0.0%)
LIB	18 (54.5%)	8 (24.2%)	7 (21.2%)
Total	1331 (70.3%)	292 (15.4%)	270 (14.3%)

For the LIB instances, EAX still performed better than LKH+IPT; however, EAX did not overwhelmingly outperform LKH+IPT as it had for the NETGEN and MORPH instance types. The 3 World instances included in our study (`uy734`, `zi929` and `mu1979`) were solved significantly better by EAX than LKH+IPT. Focusing on VLSI only, the median runtime value for EAX was 12.8s, with LKH+IPT returning a median runtime of 59.2s. When considering these with the results in Table 1, it is apparent that EAX solves VLSI instances to optimality with noticeably greater speed than LKH+IPT.

3.2 LKH+GPX2 v EAX Performance Results

Examining both Fig. 3 and Table 2, it is apparent that EAX provides considerably better results than LKH+GPX2. As with the results for EAX v LKH+IPT in the previous section, EAX performs substantially better than LKH+GPX2, most notably for NETGEN (95.0% EAX) and MORPH (93.5% EAX) instances. Again, however, EAX performs less effectively versus LKH+GPX2 when solving RUE types; EAX drops to 25.8%, whereas LKH+GPX2 rises to 38.8% best effectiveness.

Comparing the classifications for EAX across Tables 1 and 2, there exists very little difference between all results, thus inferring that EAX performs as well against LKH+IPT as it does versus LKH+GPX2. Also, note again that EAX is the dominant solver for VLSI, with 64.9% of all VLSI instances solved best by EAX, and only 7 out of 57 best solved by LKH+GPX2.



(a) NETGEN (b) MORPH

(c) RUE (d) TSPLIB

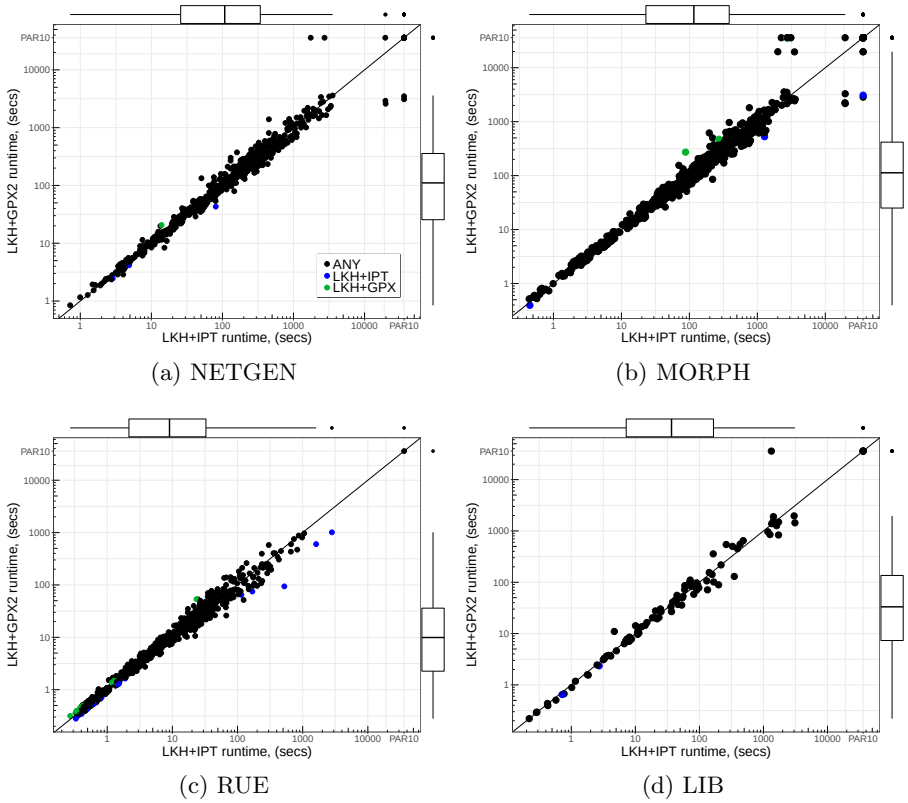
Fig. 3: LKH+GPX2 v EAX runtime performance results. All plot axes are log-scaled; scatter points are median values of 30 runs of pairwise instances (LKH+GPX2, EAX); diagonal lines represent equality of median performance; boxplots on axes show distribution of median runtime values

Table 2: LKH+GPX2 versus EAX classifications

Instance Type	LKH+GPX2	EAX	ANY
RUE	233 (38.8%)	155 (25.8%)	212 (35.3%)
MORPH	14 (2.3%)	561 (93.5%)	25 (4.2%)
NETGEN	10 (1.7%)	570 (95.0%)	20 (3.3%)
TSPLIB:			
VLSI	7 (12.3%)	37 (64.9%)	13 (22.8%)
World	3 (100.0%)	0 (0.0%)	0 (0.0%)
LIB	9 (27.3%)	19 (57.6%)	5 (15.2%)
Total	276 (14.6%)	1342 (70.9%)	275 (14.5%)

3.3 LKH+IPT v LKH+GPX2 Results

As inferred by their relative performances against EAX, the runtime results for LKH+IPT versus LKH+GPX2 exhibit very little variation. Both Fig. 4 and Table 3 show that, for instances of these sizes and types, and with a stopping criterion of one hour, there exists little appreciable difference in their relative performances. The four plots in Fig. 4 show that almost all scatter points lie very closely to the line of median equality. When this did not hold, for example in Fig. 4(d) where a single point is located at (1329, PAR10) (corresponding to VLSI instance `bck2217`), the Mann-Whitney test does not always allow rejection of its null hypothesis and so the instance is labelled as ANY.



(a) NETGEN (b) MORPH

(c) RUE (d) LIB

Fig. 4: LKH+IPT v LKH+GPX2 runtime performance scatter plots

Table 3 further emphasises the similarities of runtimes for both LKH+IPT and LKH+GPX2. 97.8% of the 1893 instances studied were classed as being best solved by either heuristic: only the RUE types returned any notable numbers of instances solved better by either solver.

Table 3: LKH+IPT versus LKH+GPX2 Classifications

Instance Type	LKH+IPT	LKH+GPX2	ANY
RUE	12 (2.0%)	17 (2.8%)	571 (95.2%)
MORPH	4 (0.6%)	3 (0.5%)	593 (98.8%)
NETGEN	1 (0.1%)	3 (0.5%)	596 (99.3%)
TSPLIB:			
VLSI	0 (0.0%)	0 (0.0%)	57 (100.0%)
World	0 (0.0%)	0 (0.0%)	3 (100.0%)
LIB	0 (0.0%)	2 (6.5%)	31 (93.9%)
Total	17 (0.9%)	25 (1.3%)	1851 (97.8%)

3.4 Overall Performance Classification

Due to the overall dominance of EAX in these results, and the similar behaviours of LKH+IPT and LKH+GPX2 (now referred to jointly as SET-LKH), an appropriate question to answer to determine the best solver is "When should we not apply EAX?". If this can be answered, then three courses of action are available: (i) Use EAX only; (ii) use one of SET-LKH; (iii) use any of the above. Due to the close similarity between the SET-LKH results, only the results from the EAX v LKH+IPT and EAX v LKH+GPX2 comparisons were used to determine the overall best heuristic per instance. For overall classification, it is assumed that LKH+IPT and LKH+GPX2 did not perform significantly differently for any instances; only 2.2% of instances were solved significantly more quickly by one than the other (cf. Table 3), and thus is a reasonable simplification to make which does not adversely affect best solver selection.

Table 4 summarises the classification permutations (excluding LKH+IPT v LKH+GPX2) and their outcomes which are used to determine the best overall heuristic per instance. Each permutation of EAX v LKH+IPT and EAX v LKH+GPX2 results in different hierarchical results which are shown as pictograms in Table 4; LKH+IPT denoted by a green "I", EAX (red "E") and LKH+GPX2 (blue "G").

EAX was deemed the best heuristic if it was significantly better than both SET-LKH solvers, or better than one while not defeated by the other (top 3 rows of Table 4). The same reasoning applies for SET-LKH (middle 3 rows), whereas the ANY class was applied only if EAX was not significantly better than both SET-LKH solvers, and vice versa. The permutations where EAX defeated one of the SET-LKH solvers but was defeated by the other did not occur in our study, and so is not shown here. Applying this methodology we can determine whether EAX, one of the SET-LKH heuristics, or ANY of the three would be the best approach.

Table 5 shows that, out of the 1893 instances tested, 1361 were solved significantly better by EAX. EAX also comprises part of the ANY category, and so can be identified as being the most effective heuristic, or at least one of the most effective, for 84.1% of the instances tested.

Table 4: Best Heuristic Classification by Instance

EAX v LKH+IPT	**EAX v LKH+GPX2**	**Pictogram**	**Best Heuristic**
EAX	EAX	E GI	EAX
EAX	ANY	EG I	EAX
ANY	EAX	E G I	EAX
LKH+IPT	ANY	 EG I	SET-LKH
ANY	LKH+GPX2	 E G I	SET-LKH
LKH+IPT	LKH+GPX2	 E GI	SET-LKH
ANY	ANY	EGI	ANY

Excluding the RUE instances, the proportion best solved by the classifications EAX or ANY rises to 96.6%. Conversely, considering only the RUE instances, we see that the SET-LKH heuristics perform effectively 72.5% of the time when combined with the ANY classification. The VLSI instances, often considered the most intractable or difficult to solve to optimality, are best solved by EAX 68.4%; only 14.0% are solved significantly quicker by SET-LKH solvers.

Table 5: Best Overall Heuristic Classifications

Type	EAX	SET-LKH	ANY
RUE	165 (27.5%)	257 (42.8%)	178 (29.7%)
MORPH	563 (93.8%)	16 (2.7%)	21 (3.5%)
NETGEN	572 (95.3%)	11 (1.8%)	17 (2.8%)
TSPLIB:			
VLSI	39 (68.4%)	8 (14.0%)	10 (17.5%)
World	3 (100.%)	0 (0.0%)	0 (0.0%)
LIB	19 (57.5%)	9 (27.3%)	5 (15.2%)
Total	1361 (72.9%)	301 (15.9%)	231 (12.2%)

The marked variations in these results show how important it is to understand the underlying TSP instance itself, and the features of the instance. Doing so can provide important guidance on which solver can be implemented most effectively.

4 Heuristic Selection by Minimal Feature Extraction

It is important to be able to identify instance features which can inform specific algorithm selection. To this end, a decision tree model was trained using the Pihera and TSPMETA feature sets described earlier. A decision tree is a simple but useful technique for creating "rules of thumb "that can be applied to identify instance traits that may hamper the performance of specific solvers. This process identified key features (and their associated values) which can be used to select the most effective instance-specific algorithms. From the head of the decision tree shown in Fig. 5, two key features were extracted which can be used to identify a significant variance in performance between the EAX and SET-LKH solvers: `nn5.sc.max.n`, from the Pihera feature set; and `mst_dists_median`, from the TSPMETA feature set.

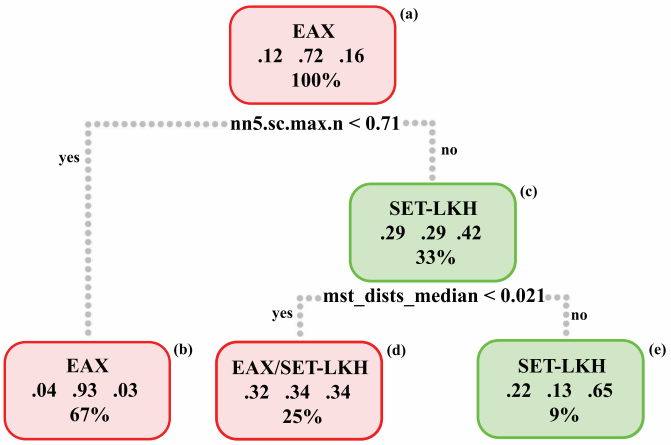
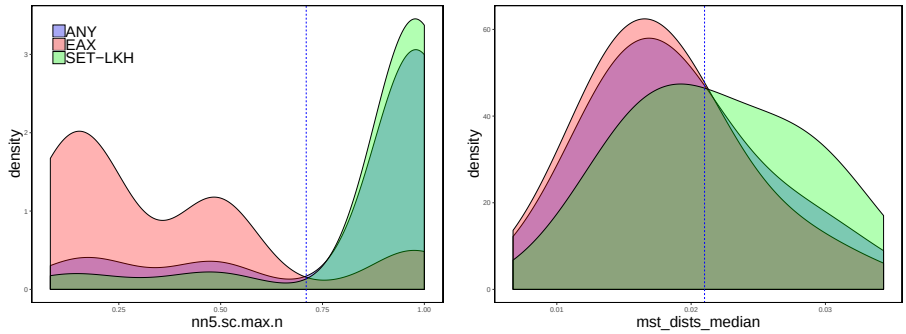


Fig. 5: Decision tree trained on all instance features (tree depth = 2)

It can be seen that 72% of instances were solved significantly faster by EAX than either of the SET-LKH heuristics and, in a further 16%, no significant difference was found (cf. Table 5 & Fig. 5(a)). The intuitive choice, therefore, would be to use EAX by default; however, note that 12% of all instances tested were best solved by the SET-LKH heuristics. It was found that 67% of all instances exhibited `nn5.sc.max.n` < 0.71; of these, 93% were solved best by EAX (cf. Figures 5(b) and 6(a)). When `nn5.sc.max.n` $>= 0.71$, 71% were solved by SET-LKH heuristics in comparable (29%) or less (42%) time, Fig. 5(c).



(a) Kernels of `nn5.sc.max.n` (b) Kernels of `mst_dists_median`

Fig. 6: Plots show kernels of ANY, EAX, SET-LKH classed instances for features `nn5.sc.max.n` (a) and `mst_dists_median` (b). Vertical lines indicate the threshold values obtained from the decision tree in Fig. 5

This branch of the decision tree can be further broken down when feature `mst_dists_median` $>= 0.021$, with 87% of instances on that branch solved in similar or less time by SET-LKH, Fig. 5(e). In cases where `mst_dists_median` < 0.021, we see a less distinctive split in performance, with EAX and SET-LKH performing best in a similar number of cases, Fig. 5(d).

Figures 6(a) & (b) show that the ANY kernels, labelled as being blue in colour, are completely masked by the EAX and SET-LKH kernels either side of the decision tree threshold values. This infers that the instances classified as ANY may defy simple identification using decision tree methods.

5 Conclusions

We have carried out an empirical runtime analysis of three state-of-the-art TSP heuristic solvers (LKH+IPT, LKH+GPX2, and EAX), ranking their respective performances using robust statistical methods. The algorithms were run on 1893 distinct instances, made up of 600 uniformly distributed, 600 strongly clustered, 600 loosely clustered, and 93 TSP problems selected from available benchmark datasets. Our results show that EAX performs significantly better than the others, especially for the NETGEN and MORPH instances with almost all of those types solved significantly better by EAX. However, for the RUE types, EAX does not perform as well as the SET-LKH solvers: EAX was only classed as the best solver $\sim 25\%$ of the time.

EAX was ranked the best overall solver per instance, identified as being significantly better in 72.9% of instances. Including the ANY category (instances where no heuristic proved to be significantly better), EAX could be applied to 85% of all instances tested to return the quickest time to optimality. Even though we observed EAX as being strongly dominant, these results further support the

"No Free Lunch" theorem [20], i.e., that no heuristic outperforms any other solver for all problems.

Best algorithm selection is a topic of ongoing research across all combinatorial optimisation problems [8, 9, 21]. For the TSP instance set analysed here, we have shown that the simple use of a decision tree can allow identification of a few discriminatory features of the instance space that pinpoint which algorithm is best applied. Our analysis identified one feature from the Pihera set (`nn5.sc.max.n`) whose value strongly predicts when to use EAX. When this is used in concert with another feature from the TSPMETA group (`mst_dists_median`), we can quickly identify when to apply one of the SET-LKH solvers. Thus, calculating both TSPMETA and Pihera features sets of the TSP instances can provide strong guidance to inform which solver to use.

There is no evidence to assume that EAX would continue to remain dominant over the SET-LKH solvers for increased instance sizes, or that the similar results for both LKH+IPT and LKH+GPX2 would also hold. Therefore an extension of the work presented here would be to carry out similar analyses for symmetric TSP instances of size $> 2\,000$. It is anticipated that, due to the effects of combinatorial explosion, a similar study of increased instance size would need to implement a timeout criterion greater than the one hour used here. Renato et al. [16] indicate that LKH+GPX2 does outperform LKH+IPT for most instances they studied which had sizes in the range $[3056, 115475]$; however, their study was not limited by a timeout criterion, and instead enforced a minimum number of trials per run. Thus, increasing the timeout criterion for larger instances may allow the LKH+GPX2 heuristic the capacity to realise any advantage it may have over LKH+IPT and EAX for larger instances. We must also consider that LKH+GPX2 is a newly developed algorithm which may not have fully optimised implementation methods, and any refinements may significantly improve its future performance.

Acknowledgements. This work was supported by the Leverhulme Trust [award number RPG-2015-395] and by the UK's Engineering and Physical Sciences Research Council [grant number EP/J017515/1]. Results were obtained using the EPSRC-funded ARCHIE-WeSt High Performance Computer (www.archie-west.ac.uk, EPSRC grant EP/K000586/1).

Data Access. All data generated for this research are openly available from the Stirling Online Repository for Research Data (http://hdl.handle.net/11667/127).

References

1. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study. Princeton University Press (2007)
2. Helsgaun, K.: Effective implementation of the Lin-Kernighan traveling salesman heuristic. Eur. J. Oper. Res. 126(1), 106–130 (2000)

3. Helsgaun, K.: An effective implementation of k-opt moves for the lin-kernighan tsp heuristic. Tech. Rep. 109, Roskilde University (2007)
4. Nagata, Y., Kobayashi, S.: A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. INFORMS J. Comput. 25(2), 346–363 (2013)
5. Whitley, D., Hains, D., Howe, A.: A hybrid genetic algorithm for the traveling salesman problem using generalized partition crossover. In: PPSN XI. LNCS, vol. 6238, pp. 566–575. Springer (2010)
6. Corne, D.W., Reynolds, A.P.: Optimisation and generalisation: Footprints in instance space. In: PPSN XI. LNCS, vol. 6238, pp. 22–31 (2010)
7. Smith-Miles, K., Lopes, L.: Measuring instance difficulty for combinatorial optimization problems. Comput. Oper. Res. 39(5), 875–889 (2012)
8. Pihera, J., Musliu, N.: Application of machine learning to algorithm selection for TSP. In: IEEE 26th International Conference on Tools with Artificial Intelligence. pp. 47–54 (2014)
9. Kotthoff, L., Kerschke, Pascaland Hoos, H., Trautmann, H.: Improving the state of the art in inexact tsp solving using per-instance algorithm selection. In: Learning and Intelligent Optimization. pp. 202–217. Springer, Cham (2015)
10. Kerschke, P., Kotthoff, L., Bossek, J., Hoos, H.H.: Leveraging tsp solver complementarity through machine learning. Evol. Comput. 26(4), 597–620 (2018)
11. Bossek, J.: `netgen`: Network Generator for Combinatorial Graph Problems (2016), https://CRAN.R-project.org/package=netgen, R package version 1.3
12. R Core Team: R: A Language and Environment for Statistical Computing. Vienna, Austria (2018), https://www.R-project.org/
13. 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. Ann. Math. Artif. Intell. 69(2), 151–182 (2013)
14. Reinelt, G.: Tsplib– a traveling salesman problem library. ORSA Journal on Computing 3(4), 376–384 (1991)
15. Rohe, A.: VLSI data sets. http://www.math.uwaterloo.ca/tsp/vlsi/ (2017), online; accessed 5 November 2018
16. Tinós, R., Helsgaun, K., Whitley, D.: Efficient recombination in the Lin-Kernighan-Helsgaun traveling salesman heuristic. In: PPSN XV. pp. 95–107. LNCS, Springer (2018)
17. Lin, S., Kernighan, B.W.: An Effective Heuristic Algorithm for the Traveling-Salesman Problem. Operations Research 21, 498–516 (1973)
18. Nagata, Y., Kobayashi, S.: Edge assembly crossover: A high-power genetic algorithm for the Travelling Salesman Problem. In: ICGA. pp. 450–457 (1997)
19. Applegate, D., Bixby, R., Chvátal, V., Cook, W.: Concorde TSP solver (2003), http://www.math.uwaterloo.ca/tsp/concorde.html
20. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Trans. Evol. Comput. 1(1), 67–82 (1997)
21. Bischl, B., Mersmann, O., Trautmann, H., Preuß, M.: Algorithm selection based on exploratory landscape analysis and cost-sensitive learning. p. 313 (2012)