On the Empirical Scaling Behaviour of State-of-the-art Local Search Algorithms for the Euclidean TSP

Jérémie Dubois-Lacoste
Université libre de Bruxelles (ULB), IRIDIA
jduboisl@ulb.ac.be

Holger H. Hoos
University of British Columbia (UBC)
hoos@cs.ubc.ca

Thomas Stützle
Université libre de Bruxelles (ULB), IRIDIA
stuetzle@ulb.ac.be

ABSTRACT

We present a thorough empirical investigation of the scaling behaviour of state-of-the-art local search algorithms for the TSP; in particular, we study the scaling of running time required for finding optimal solutions to Euclidean TSP instances. We use a recently introduced bootstrapping approach to assess the statistical significance of the scaling models thus obtained and contrast these models with those recently reported for the Concorde algorithm. In particular, we answer the question whether the scaling behaviour of state-of-the-art local search algorithms for the TSP differs by more than a constant from that required by Concorde to find the first optimal solution to a given TSP instance.

1. INTRODUCTION

The TSP is one of the best-known combinatorial optimisation problems, and its pivotal role in the development and assessement of new algorithmic ideas could hardly be overestimated. Until recently, the state of the art in exact and inexact TSP solving has been defined by two long-standing incumbents: Concorde [2] and the Lin-Kernighan heuristic (LKH) [4, 5]. In 2013, Nagata and Kobayashi introduced EAX [13] – a new evolutionary algorithm for the TSP that makes use of an improved version of the edge assembly crossover operator [12] to achieve an effective recombination of sub-tours. Their empirical results suggest that EAX tends to perform better than the latest version of LKH and hence represents an improvement in the state of the art in inexact TSP solving. A recent study by Kotthoff *et al.* confirmed that, on aggregate, EAX performs better than LKH across a broad range of Euclidean TSP instances, but that EAX does not dominate LKH, in that there is a substantial number of instances on which LKH reaches tour qualities known to be optimal (from previous runs of Concorde) substantially faster [10].

In what follows, we study the empirical performance of EAX and LKH in more detail. In particular, we investigate the scaling of the median running time required by these state-of-the-art inexact TSP solvers for finding solutions to 2-dimensional Euclidean TSP instances that are known to be optimal (from previous, typically much longer runs of Concorde). We focus on 2D Euclidean TSP instances, since they constitute arguably the most widely studied special case of the general TSP, and because 2D instances occur in many practical applications [1, 3]. Our study is inspired by recent results we obtained on the empirical scaling of Concorde's running time on Euclidean TSP instances with instance size (number of cities), n, according to which median running time closely follows a function of the form $a \cdot b^{\sqrt{n}}$ [8]. Furthermore, in this previous study, we reported log-normal distributions of running time over sets of instances and showed that Concorde's performance on structured 2D Euclidean TSP instances generally agrees well with the scaling model derived from random uniform Euclidean (RUE) instances.

Specifically, in this work, we are interested in answering the following questions:

1. How does the performance of state-of-the-art inexact TSP solvers scale with instance size?
2. How does the performance scaling of these solvers compare to that of the state-of-the-art exact TSP solver, Concorde?
3. Are there qualitative differences between the performance scaling of EAX and LKH?
4. How does the variation of performance across sets of TSP instances of the same size compare between EAX, LKH and Concorde?

Regarding Questions 1 and 2, as we will show, there are interesting qualitative differences between the scaling behaviour of Concorde and that of EAX and LKH. In particular, while the scaling of running time for EAX and LKH is consistent with the same two-parameter root-exponential model, $a \cdot b^{\sqrt{n}}$, previously established for Concorde, there is clear evidence that the scaling parameter b is significantly lower for EAX and LKH (when an appropriate restarting mechanism is used). Furthermore, regarding Question 3, there is weak evidence that the performance of EAX might scale better with instance size than that of the latest version of LKH. Finally, regarding Question 4, unlike in the case of Concorde, the running times of EAX and LKH are not log-normally distributed for fixed instance size, and the distributions for EAX are distinctly multi-modal. Still, we found no evidence for substantially different scaling between the median and higher quantiles of these distributions.

GECCO '15, July 11–15, 2015, Madrid, Spain

DOI: http://dx.doi.org/10.1145/2739480.2754747

The remainder of this article is structured as follows. In Section 2, we introduce the instance sets, the algorithms and the analysis methods we used. We present our experimental results in Section 3. Finally, in Section 4, we discuss our results and their implications before closing with a brief outlook on future work.

2. INSTANCES, ALGORITHMS, METHODS

2.1 Benchmark instances

We performed experiments on Euclidean TSP instances stemming from two primary sources: RUE instances of various sizes and real-world instances from TSPLIB and from http://www.math.uwaterloo.ca/tsp/data/.

The RUE instances were generated by placing uniformly at random n points in a $100\,000 \times 100\,000$ square and computing the Euclidean distances between the points, using the `portgen` generator from the 8th DIMACS implementation challenge on the TSP. We produced 1000 instances for each instance size $n \in \{500, 600, .., 2000\}$ and 100 instances for each $n \in \{2500, 3000, 4000, 4500\}$. In previous studies, we have determined optimal solutions for most of these instances, using Concorde [2], the best performing exact algorithm for the TSP [8]; since these computations are very expensive, we used the same instances and optimal solutions.

2.2 Helsgaun's Lin-Kernighan heuristic

Helsgaun's variant of the Lin-Kernighan heuristic (LKH) represents a major advance in inexact TSP solving, and for many years, it was the uncontested state-of-the-art method for finding high-quality solutions to a large variety of TSP instances [4, 5]. The Lin-Kernighan (LK) heuristic is a variable-depth search method that generates complex local search moves by heuristically constructing a sequence of edge exchanges. Helsgaun's variant of LK is based on exchange sequences using five [4] or more edge exchanges [5]. The iterated version of the LKH algorithm restarts the local search process from new solutions that are obtained by solution perturbations, which are performed using either a random k-exchange move or a special walk strategy. Furthermore, an approximation of the Held-Karp lower bound is used to obtain small candidate sets for the local search steps.

For our experiments, we used the most recent version 2.0.7 of LKH, as well as the earlier version 1.3. After observing stagnation behaviour in LKH 1.3 and LKH 2.0.7 (which, from here on, we refer to as LKH 2), we enhanced both versions with a restart mechanism (for details see Section 2.4). Apart from the additional restart mechanism, LKH 1.3 was run with its default parameter setting. LKH version 2 comes with example parameter files for tackling TSPLIB instance `pr2392`, where the default parameter settings are modified to include patching of cycles during the search for improving moves (setting `PATCHING_A = 2` and `PATCHING_C = 3`). We compared these recommended settings to the default parameter settings for LKH 2, which would switch this patching of cycles off. As the default settings resulted in worse performance in solving `pr2392`, we used the modified parameters for all further experiments with LKH 2.

2.3 GA with edge assembly crossover

In recent years, several high-performing evolutionary algorithms for the TSP have been proposed [11]. The most successful lines of research mainly dealt with the integration of high-performance local search algorithms into so-called memetic algorithms and the development of specialized recombination operators. The most successful outcome of this work are the evolutionary algorithms integrating variants of edge assembly cross-over – 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 [12].

The first such algorithm known to have reached the performance of LKH, and hence state-of-the-art performance, in finding very high quality solutions to a broad range of Euclidean TSP instances (as considered in our work) is EAX, the genetic algorithm by Nagata & Kobayashi [13]. In a nutshell, EAX exploits improved local and global variants of the edge assembly cross-over operator, specific diversity preservation techniques that uses edge entropy measures in the population replacement scheme, and initialization of the population by local optimization. (For a detailed description of the algorithm and the operators, we refer the reader to the original publication by Nagata & Kobayashi [13].)

Computational results by Nagata & Kobayashi indicate that on many large Euclidean TSP instances, EAX performs better than LKH 2; specifically, they reported that on 50 out of 57 TSPLIB, VLSI and National instances of sizes between 4 461 and 60 000 cities, EAX produced statistically significantly better solutions than LKH in shorter computation times [13]. (Our experiments comparing EAX to LKH 2 shed further light on their relative performance.)

In what follows, we used EAX with its default parameter settings, as specified in the publically available code, using a population size of 100 (which appeared to be sufficient for the instance sizes we tackled) and 30 offspring generated for each attempt of recombining two parent tours (applying the EAX recombination operator repeatedly to two given parent solutions allows to generate several different offspring). The original version of EAX does not support termination upon reaching a given solution quality and used a complex termination criterion to end runs. We modified this version to terminate upon reaching a target solution quality (in our experiments always set to the known optima for the given instances) and to restart whenever the original termination criterion is met, as discussed in more detail in the following.

2.4 Restarts

In this work, we examine the scaling behavior of LKH and EAX for reaching optimal solutions on RUE instances. Ideally, as a pre-requisite, we should ensure that LKH and EAX find an optimal solution in each run. Unfortunately, it is known that even high-performing local search algorithms can be prone to stagnation behaviour [14, 7]. As it happens, this holds for the two high-performance TSP algorithms we consider here. It is clearly seen from the run-time distributions shown in Figure 1 for EAX (all running times were measured on the reference machines specified in Section 2.5).

In fact, especially EAX exhibits a very particular run-time behaviour: It typically requires a relatively high initialisation time until the search process starts producing very high-quality solutions, but once it does, even better solutions are frequently found quite rapidly. However, some runs of EAX are terminated prematurely, based on a built-in, complex criterion. Clearly, the behaviour of EAX can be improved by restarting the search rather than terminating it. Similarly,

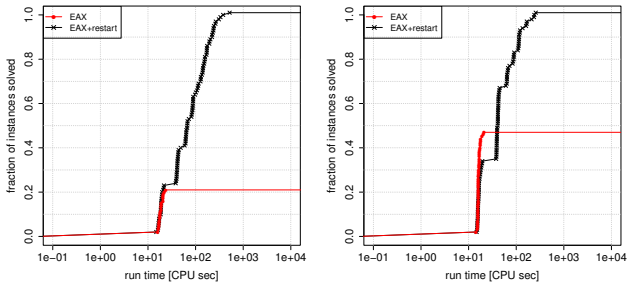

Figure 1: Distributions of running times over multiple independent runs of EAX with and without restart on two RUE instances of size 1500.

we observed stagnation behaviour that could be remedied using a restart mechanism for both versions of LKH.

Based on these observations, we modified the original EAX to perform restarts whenever its original termination criterion is met, and to terminate only when a given target solution quality or CPU time cutoff is reached. In the following, we simply refer to this variant as EAX. Furthermore, we augmented LKH 1.3 and LKH 2 with a mechanism that would trigger a restart when during n iterations, where n is the size of the TSP instance to be solved, no improved solution has been found. From here on, whenever we refer to LKH 1.3 and LKH 2, we mean their versions with restarts.

2.5 Computing environment and experimental setup

We performed all algorithm runs used in our experiments on a cluster of computing nodes equipped with a 2.0 GHz eight-core AMD 6128 processor and 2×12MB L2/L3 cache as well as 16GB RAM each, running Cluster Rocks Linux 6.0/CentOS 6.3. The programmes were compiled with gcc 4.4.6 (using optimization flag `-O3` and run on a single CPU core, as the implementations are all fully sequential. We ran Concorde on those same machines, to ensure comparability of results. We further note that the machines are similar to those used in our previous analysis of the scaling of Concorde's running time [8].

For each of the instances that were solved to optimality by Concorde[1] we performed 10 independent runs of each of LKH 1.3, LKH 2.0.7 and EAX. We chose cutoff times that ensured that all runs of these algorithms were always successful in producing an optimal solution. From those sets of 10 runs per instance and algorithm, we computed median CPU times for each algorithm and instance. All further analysis were performed based on those performance measurements.

2.6 Scaling analysis

To study the scaling of EAX and LKH with instance size n, we considered quantiles (particularly, the median) of the distributions of running times obtained for sets of RUE instances with various fixed values of n. Following [8], we considered the following three 2-parameter scaling models:

- exponential: $Exp[a,b](n) = a \cdot b^n$
- polynomial: $Poly[a,b](n) = a \cdot n^b$
- square-root exponential: $SRExp[a,b](n) = a \cdot b^{\sqrt{n}}$

Our scaling analysis was performed using performance data for EAX, LKH 2 and LKH 1.3 on instance sizes $n = 500, 600, \ldots, 1500$ (the so-called support for our models); specifically, we fitted each scaling model on 11 data points, one for each instance size. The fitting was performed using the nonlinear least-squares Marquardt-Levenberg algorithm implemented by the 'fit' command of the *gnuplot* software.

To asses the degree to which the predictions made by our scaling models are affected by the precise sets of instances in our support, we employed the bootstrapping procedure introduced by Hoos [6] and that we used in our previous study of Concorde [8]: we generated $m = 1000$ samples of algorithm performance data for each instance size in the support (using uniform random sampling with replacement), and for each of these bootstrap samples, we fitted our scaling model to obtain m sets of performance predictions for the large challenge instance sizes. From these sets of predictions, we generated bootstrap confidence intervals for confidence level $\alpha = 0.95$, resulting in an interval $CI = [Q(0.5 - \alpha/2), Q(0.5 + \alpha/2]$, where $Q(x)$ denotes the x-quantile of the distribution of the predicted performance values.

3. EXPERIMENTAL RESULTS

3.1 Performance correlation

As a first step in our analysis, we examined the correlation of the median running times among our three inexact solvers and between these and Concorde. Figure 2 shows scatter plots between running times for pairs of solvers for $n = 1500$; results for other n look quite similar and are summarised by the correlation coefficients shown in Table 1. We note that the correlation of the running times of Concorde and the inexact solvers is very low, as can be seen from the scatter plots for Concorde *vs* EAX (top plot) and LKH 2 (second from top), as well as from the low correlation coefficients of 0.0925 and 0.1716, respectively (*cf.* Table 1). More surprisingly, the performance correlation between EAX and LKH 2 (but also LKH 1.3, not shown here) is also very low, as seen from the third scatter plot in Figure 2 (the corresponding correlation coefficient is 0.0611). Only for the two versions of LKH, performance correlation is somewhat higher (bottom-most plot, correlation coefficient 0.512), but still surprisingly low, considering the overall conceptual similarity between LKH 2 and LKH 1.3.

These results suggest that we have no reason to expect the performance of these TSP algorithms to show the same scaling behaviour (in contrast to a hypothetical situation where performance correlations at different instance sizes had been very tight). Consistent with recent work by Kotthoff *et al.* [10], our results also clearly demonstrate the potential for achieving performance gains using portfolio techniques, such as algorithm selection. On the instance sizes we considered, as well as on structured instances (see Section 4), we found no evidence that the performance of any of our three inexact solvers would be dominated by any of the others, even as n is growing. However, we did see some evidence suggesting that, in terms of simply finding high-quality solutions to Eu-

[1]For instance sizes up to $n = 3000$, all instances were solved to optimality; for larger instance sizes, we mention the details where relevant.

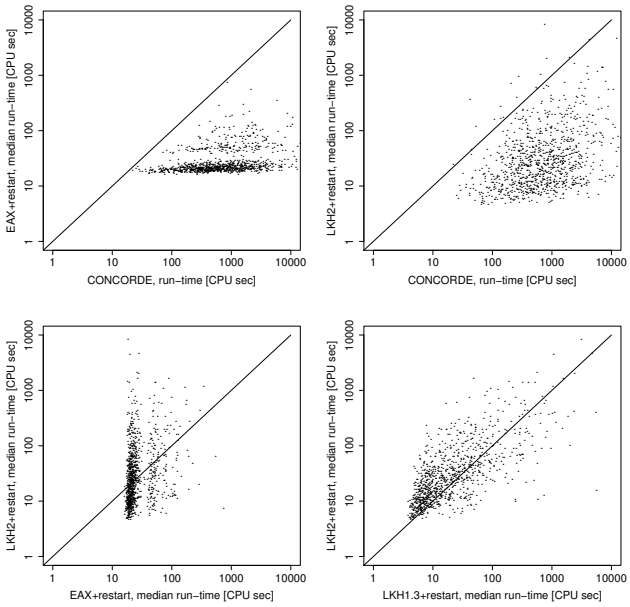

Figure 2: Median running time of Concorde *vs* EAX (top left), Concorde *vs* LKH 2 (top right), EAX *vs* LKH 2 (bottom left) and LKH 1.3 vs LKH 2 (bottom right) on RUE instances, $n = 1500$.

clidean TSP instances, for large n, Concorde performs worse than the inexact solvers (data not shown).

3.2 Empirical scaling on small instances

As a first step of our the scaling analysis, we visually examined the scaling of the median and the 0.1 and 0.9 quantiles, $Q(0.1)$ and $Q(0.9)$, of the distributions of (median) running times over RUE instance sets for $n \in \{500, \ldots 1500\}$, which are the same instances used in a previous analysis (support data) [8]. As can be seen in Figure 3, all three solvers show roughly linear median scaling curves in a log-log plot, indicating close to polynomial scaling. For both versions of LKH, the scaling curves for $Q(0.9)$ are roughly parallel to that for the median, which indicates a constant factor between the corresponding running times, and hence suggests that scaling on the hardest instances is not worse than observed for the median. For EAX, on the other hand, there is a strong increase in $Q(0.9)$ between $n = 900$ and $n = 1100$, after which the scaling curves seem to become parallel again. The main reason for this is that for increasing n, EAX requires restarts on an increasing fraction of the instances to solve them to optimality, while small instances are often solved without restarting the search process. The high initialization time incurred by EAX, thus, causes multiple modes to appear in its running time distributions, which in turn induces the observed scaling of $Q(0.9)$ (and other high quantiles). Further insights into this behavior are given in Section 3.5. The 0.1 quantile for EAX is very close to the median, as a large fraction of the instances (especially for small n) are solved without restarting. For LKH 2 and LKH 1.3, $Q(0.1)$ is significantly lower than the median, but seems to be scaling similarly.

Next, we fitted our three two-parameter scaling models to the median running times from this support data. Table 2 shows the best-fit models obtained for each algorithm. The base b of the root-exponential model of LKH 2 is smaller ($b = 1.18264$) than in the case of LKH 1.3 ($b = 1.19592$), indicating better scaling behaviour of LKH 2. The exponent of the polynomial model for EAX is $b = 1.95417$, indicating slightly less than quadratic scaling of median running time with instance size.

Table 2: Best-fit scaling models (*i.e.*, those with minimal RMSE), as specified in Section 2.6.

Algorithm	Best-fit model	a	b
EAX	Poly	1.38225e-05	1.95417
LKH 2	SRexp	0.03481	1.18264
LKH 1.3	SRexp	0.01362	1.19592

Table 3: RMSE of scaling models on support data.

Model	RMSE		
	EAX	LKH 2	LKH 1.3
Poly	0.10306	0.54834	0.37975
Exp	0.78518	0.58083	0.29718
SRexp	0.43369	0.44632	0.27701

Table 3 shows the root mean square error (RMSE) for each of the models we fitted. In all cases, the exponential models give the worst fit, that is, they have the highest RMSE values. For both variants of LKH, the best fit is obtained from the root-exponential model, while for EAX, the polynomial model has the lowest RMSE. Table 2 shows the best-fit models obtained for each algorithm. The base b of the root-exponential model of LKH 2 is smaller ($b = 1.18264$) than in the case of LKH 1.3 ($b = 1.19592$), indicating better scaling behaviour of LKH 2. The exponent of the polynomial model for EAX is $b = 1.95417$, indicating slightly less than quadratic scaling of median running time with instance size.

3.3 Performance scaling on larger instances

Next, we assessed the predictive power of our models by comparing predicted and observed performance on larger instances sizes (challenge data), with $n \in \{2500, \ldots 4500\}$. In this context, we encountered a difficulty, in that for $n > 3000$, we do not have optimal tour lengths for all TSP instances, since Concorde (even with the 7 CPU day running time cutoff that was used in [8]) was unable to solve some instances. Considering the lack of tight performance correlation between Concorde and the solvers considered here (see Section 3.1), some of those instances might be easy for EAX or LKH. Therefore, there is uncertainty about the precise location of the medians of the running time distributions at each such n, and we can only provide bounds on those medians instead. These bounds are based on the best and worst case scenarios, in which all instances unsolved by Concorde are easiest or hardest, respectively, for the algorithm in question. We note that these are not confidence intervals, since we can guarantee the actual median running times to lie within them. (As we will discuss further in Section 4, there may be ways to tighten or collapse those bounds, but they would come at considerable computational cost.)

From the resulting intervals for observed medians, we derived lower and upper bounds on the RMSEs for all models, which are given in Table 4. From these bounds, we observed that, for all three solvers, the exponential scaling model agrees least with the observed performance values, as in all cases it has by far highest RMSE intervals. For EAX and LKH 1.3, the root-exponential scaling model agrees best with observed performance, while for LKH 2, the polynomial model shows the smallest bounds on RMSE. We note how these results differ from the conclusions drawn based

Table 1: Pearson correlation coefficient between running times of EAX, LKH 2, LKH 1.3 and Concorde (Crd) on sets of RUE instances of size n.

n	500	600	700	800	900	1000	1100	1200	1300	1400	1500	2000	2500	3000	3500	4000	4500
EAX, LKH 1.3	0.0943	-0.001	0.0805	0.0476	-0.001	0.0766	0.0269	0.0205	0.0716	0.0343	0.0454	0.0646	0.0951	0.1771	0.1724	-0.0226	0.0457
EAX, LKH 2	0.2191	0.0052	0.0788	0.092	0.0022	0.0337	0.0492	0.066	0.0903	0.0258	0.0611	0.0834	0.4793	0.2714	0.1609	0.2761	0.1277
EAX, Crd	0.2038	0.2153	0.2688	0.076	0.0861	0.1364	0.1426	0.1556	0.1687	0.0798	0.0925	0.1582	0.4825	0.1343	0.1545	0.1407	0.0522
LKH 1.3, LKH 2	0.4518	0.6032	0.382	0.1564	0.3777	0.145	0.7311	0.273	0.5551	0.5083	0.512	0.3855	0.405	0.3824	0.6478	0.5681	0.4582
LKH 1.3, Crd	0.1667	0.087	0.1722	0.0464	0.0532	0.0308	0.1097	0.0961	0.0405	0.1821	0.163	0.11	0.2101	0.0349	0.3144	0.0489	-0.0278
LKH 2, Crd	0.4608	0.0577	0.1251	0.0726	-0.0044	0.0235	0.0921	0.2034	0.1135	0.0748	0.1716	0.0934	0.6921	0.2203	0.0569	0.2693	-0.0299

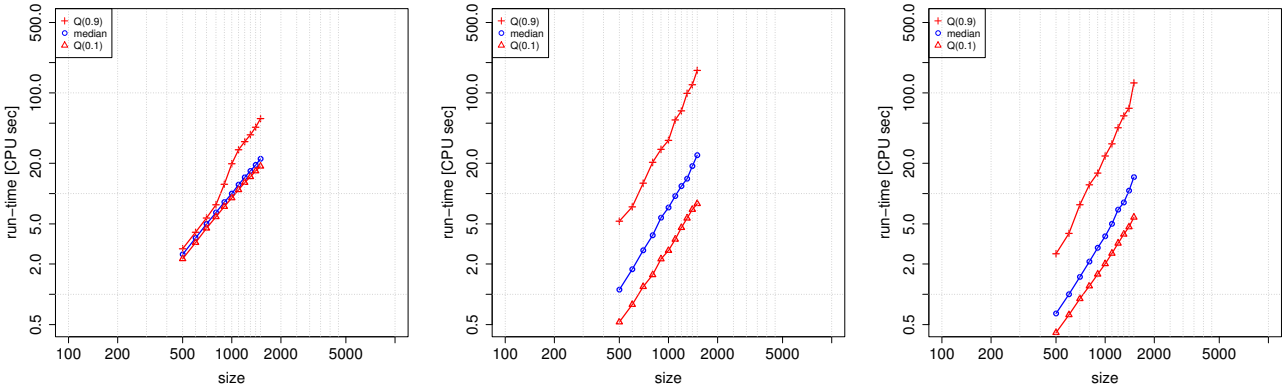

Figure 3: Empirical scaling of median, $Q(0.1)$ and $Q(0.9)$ running time of EAX (left), LKH 2 (middle) and LKH 1.3 (right) on RUE instance sets.

Table 4: RMSE for scaling models from Section 3.2 on challenge data.

Model	RMSE
EAX	
Poly	[103.6738, 328.7718]
Exp	[1738.077, 1979.996]
SRexp	[8.760844, 158.3214]
LKH 2	
Poly	[114.694, 488.5378]
Exp	[13944.76, 14365.05]
SRexp	[516.7933, 930.916]
LKH 1.3	
Poly	[642.3105, 1613.221
Exp	[12046.95, 13052.59]
SRexp	[101.087, 778.7793]

on performance observed on support data only; the reason for this lies likely in somewhat different scaling of algorithm performance for small n.

3.4 Statistical assessment of scaling models

As first observed by Hoos [6], model fits can substantially depend on the samples of instances for each n used as support data. Therefore, we statistically assess model predictions using the bootstrap method outlined in Section 2.6. Because of the uncertainty in our observed median running times discussed earlier, this assessment is slightly more involved. In the following, we say that a scaling model is *inconsistent* with observed data of the interval formed by the bounds on the observed medians described in Section 3.3 if it is disjoint from the bootstrap confidence interval for predicted running times; we say that a scaling model is *consistent* with observed data, if the interval for observed medians overlaps with, but is not fully contained within the bootstrap confidence interval for predicted running times; we say that a scaling model is *strongly consistent* with observed data, if the interval for observed medians is fully contained within the bootstrap confidence interval for predicted running times.

As can be seen in Table 5, for EAX and both versions of LKH, the exponential models are almost all inconsistent with our observations (with the only exception being LKH 2 for $n = 2000$) and should therefore be dismissed. For EAX, the polynomial model is also inconsistent with our observations, while the root-exponential model is consistent for $n \geq 3500$. For LKH 2, the root-exponential model is strongly consistent with the observed performance for $n = 2000$ but inconsistent for larger n, while the polynomial model is inconsistent up to $n = 4000$ and consistent for $n = 4500$, but with relatively small overlap between the intervals. For LKH 1.3, the root-exponential model is consistent with our observations, and even strongly consistent for n up to 4000; the polynomial model, on the other hand, is inconsistent for all n.

In summary, among the models considered here, the root-exponential model best explains scaling behaviour observed for EAX and LKH 1.3, while the scaling for LKH 2 seems to lie somewhere between the polynomial and the root-exponential model. Specifically, LKH 1.3 most likely shows root-exponential scaling of median running time with instance size n, while LKH 2 seems to perform slightly better than suggested by the root-exponential model with some possibility of polynomial scaling for sufficiently large n; EAX likely

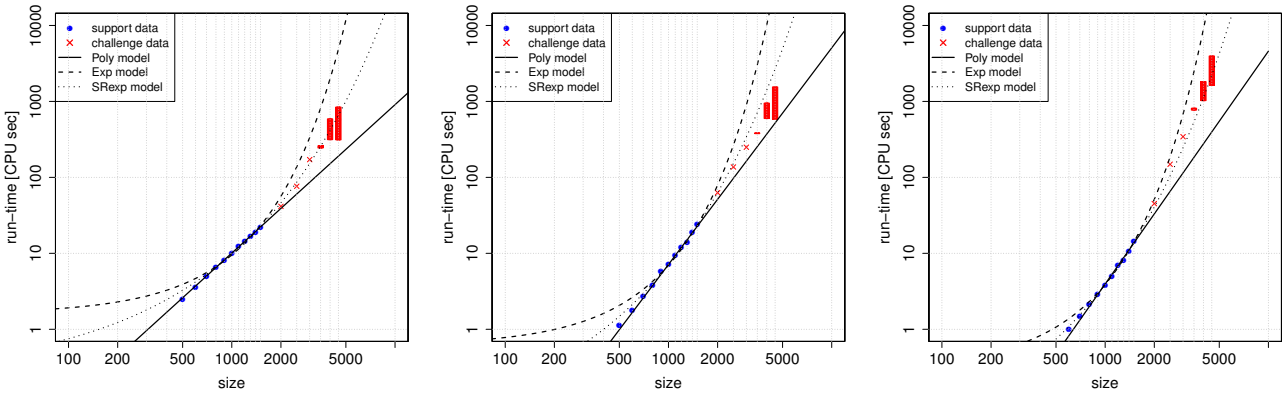

Figure 4: Scaling models of median running time (fitted on support data) and observed data for sets of larger challenge instance for EAX (left), LKH 2 (middle) and LKH 1.3 (right).

Table 6: Confidence intervals for parameter values for scaling models fitted on bootstrapped support data.

Best-fit model	a	b
	EAX	
Poly	[1.27095e-05, 1.60351e-05]	[1.93245, 1.96595]
Exp	[1.58004, 1.62804]	[1.00176, 1.00180]
SRexp	[0.22357, 0.23809]	[1.12461, 1.12681]
	LKH 2	
Poly	[1.10147e-08, 9.25764e-09]	[2.74510, 3.09874]
Exp	[0.47657, 1.00000]	[1.00206, 1.00265]
SRexp	[0.02201, 0.04205]	[1.17617, 1.19873]
	LKH 1.3	
Poly	[1.07808e-09, 9.25297e-10]	[3.01844, 3.54762]
Exp	[0.18548, 0.29762]	[1.00255, 1.00296]
SRexp	[0.00560, 0.01455]	[1.19302, 1.22686]
	Concorde	
Poly	[8.69e-11, 8.44e-9]	[3.46, 4.11]
Exp	[6.71, 11.21]	[1.00292, 1.00334]
SRexp	[0.115, 0.373]	[1.2212, 1.2630]

scales root-exponentially for sufficiently large n, with a small possibility of polynomial scaling.

Our bootstrap scaling analysis also produces confidence intervals for the model parameters. As seen in Table 6, EAX scales substantially better than LKH 2, which, in turn, scales somewhat better than LKH 1.3 (as one would expect). Compared to the confidence intervals for the parameters of the root-exponential scaling model for Concorde reported in [8] and also shown in the table, EAX and LKH 2 show significantly better scaling than Concorde, and LKH 1.3 seems to scale slightly better (although the respective confidence intervals for the scaling parameter b overlap minimally).

3.5 Performance variation for fixed instance size

In Figure 5, we show distributions of median running time per instance over sets of RUE instances of a fixed size. We clearly observe that distributions for EAX are multi-modal. Closer examination revealed that this is a consequence of the fact that the complex restart mechanism fails to completely avoid stagnation behaviour. As n grows, the fraction of instances that can be solved without restart drops, while the fraction of instances that do require restarts increases. Each mode (visible as a 'hump' in the empirical CDF) corresponds to a set of instances for which a specific number of restarts has been performed. Still, from examining the right tails of the CFDs, there is no clear evidence that the running time for the most difficult instances increases faster than observed at median difficulty, once restarts are required. This is consistent with what we observed earlier, in Section 3.2.

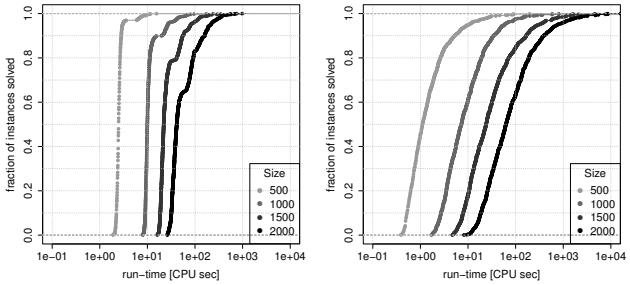

Figure 5: Cumulative distribution functions of running time for EAX (left) and LKH 2 (right) over sets of RUE instances. The plot for LKH 1.3 (not shown here) looks qualitatively similar to that for LKH 2.

The situation for LKH is qualitatively different: Here, the distributions of median running time across instances for fixed n show no sign of multi-modality, indicating that the restart mechanism we added to the original implementations of LKH 1.3 and LKH 2 does not suffer from the type of stagnation found in EAX. Again, there is no evidence that the higher quantiles of these distributions scale worse than the median. However, different from the situation from Concorde, where the distributions were characterised as log-normal [8], here, we find distributions that deviate from log-normality, as clearly evident from the fact that the CDFs, shown in the semi-log plot, are not point-symmetrical (as all log-normal distributions would be).

Table 5: Bootstrap confidence intervals for predicted median running times, obtained from scaling models fitted on bootstrapped support data.

size	Poly model	Exp model	Root-exp model	observed (corrected) median
		EAX		
2000	[38.36, 39.26]	[55.28, 56.87]	[45.46, 46.62]	41.45
2500	[59.05, 60.87]	[133.48, 139.32]	[84.51, 87.57]	76.34
3000	[84.01, 87.12]	[322.12, 341.19]	[148.03, 154.79]	170.79
3500	[113.18, 117.97]	[777.61, 835.37]	[247.87, 261.36]	[243.59, 260.79]
4000	[146.48, 153.38]	[1877.36, 2045.64]	[400.44, 425.62]	[313.49, 589.76]
4500	[183.92, 193.35]	[4532.43, 5009.62]	[628.24, 672.96]	[311.17, 844.59]
		LKH 2		
2000	[48.75, 58.89]	[62.43, 93.83]	[59.59, 72.93]	62.64
2500	[90.08, 117.75]	[175.50, 351.54]	[140.75, 189.98]	137.02
3000	[148.70, 207.19]	[493.34, 1312.54]	[305.19, 451.79]	249.37
3500	[226.89, 334.09]	[1386.78, 4914.56]	[621.91, 1001.17]	[379.97, 382.77]
4000	[327.52, 505.51]	[3898.22, 18466.95]	[1208.54, 2098.30]	[594.94, 951.26]
4500	[451.91, 728.46]	[10957.83, 69105.15]	[2255.56, 4205.25]	[581.69, 1553.86]
		LKH 1.3		
2000	[31.40, 41.92]	[49.05, 68.61]	[38.70, 52.34]	45.1002
2500	[61.70, 92.38]	[175.60, 299.54]	[98.44, 153.35]	142.2693
3000	[106.99, 175.92]	[629.27, 1315.31]	[229.61, 405.85]	328.0472
3500	[170.76, 302.96]	[2255.04, 5777.80]	[497.01, 996.14]	[768.9041, 814.4542]
4000	[256.17, 486.24]	[8081.02, 25262.33]	[1019.59, 2296.28]	[1023.6704, 1822.6779]
4500	[366.26, 738.27]	[28958.66, 110454.73]	[2003.86, 5031.37]	[1630.5511, 3970.6894]

4. DISCUSSION

The results from our scaling analysis clearly rule out pure exponential scaling of the median running time required by any of the three high-performance state-of-the-art TSP solvers we studied, EAX, LKH 2 and LKH 1.3 (all equipped with restart mechanisms) to find optimal solutions of RUE instances. This is not surprising, since Concorde, which also proves optimality when run to completion, has also been shown to have more benign scaling behaviour than a simple exponential function on this widely studied type of TSP instances [8]. In fact, Concorde's median scaling of running time with instance size appears to closely follow $a \cdot b^{\sqrt{n}}$ with $a \in [0.115, 0.373]$ and $b \in [1.2212, 1.2630]$. Therefore, one would expect the scaling behaviour of high-performance inexact TSP solvers to be no worse.

In fact, our study provided evidence consistent with the hypothesis that EAX and LKH 1.3 show scaling of the same fundamental type as Concorde, albeit with smaller constants b. Our analysis is complicated by the fact that for $n > 3000$, provably optimal solutions to some RUE instances could not be obtained. This limitation also applies to an earlier study [8], but has less impact there, where the missing data points do not affect the medians used in the scaling analysis. Because the algorithms we studied here are inexact, and their performance does not correlate tightly with Concorde, we can merely bound the median running times for any set that contains instances unsolved by Concorde. As a result, even though the bootstrap confidence intervals obtained for the predictions made using our three families of models do not overlap for the most part, our observations are not entirely inconsistent with the hypothesis of polynomial scaling of the LKH 2 variant. However, considering the relatively small overlap of the interval between our bounds for the median running times on our challenge data and the respective bootstrap confidence intervals, we believe that polynomial scaling may not be very likely.

Assuming root-exponential scaling for all three inexact solvers (note that for LKH 2 the root-exponential scaling seems to overestimate actual media run-times), it becomes interesting to compare the parameters of the respective scaling models, and in particular, the parameter b, which mostly determines the scaling behaviour. We say that an algorithm A scales better than an algorithm B if, and only if, we have reason to believe that the b value of the scaling model for A is lower than that for B. Our analysis clearly suggests that, in this sense, EAX scales better than LKH 2 ($b \in [1.1246, 1.1268]$ *vs* $b \in [1.1762, 1.1987]$), which in turn scales likely better than LKH 1.3 ($b \in [1.1930, 1.2268]$). As one would expect, EAX and the current version of LKH scale better than Concorde ($b \in [1.2212, 1.2630]$), and the same is likely the case for LKH 1.3. This provides solid evidence in support of common experience suggesting that when tackling very large TSP instances, inexact algorithms are the only feasible means to obtain optimal or near-optimal solutions.

Our readers may object that comparing the scaling of an exact solver with that of an inexact solver is misleading, since after all, the exact solver will find optimal solutions possibly long before being able to complete a proof of optimality; however, recent empirical results suggest that, at least for Concorde on Euclidean TSP instances like the ones studied here, this tends to not be the case, as the fraction of Concorde's overall running time spent before an optimal solution is first encountered approaches 1 as n increases [9]. This implies that the median running time required by Concorde for first finding an optimal solution (without completing a proof of optimality) scales somewhat *worse* than its overall running time (although asymptotically, the scaling would become indistinguishable).

Naturally, we were interested to see whether the predictions obtained from the scaling models for RUE instances were consistent with the running times observed by the inexact solvers we considered on structured TSP instances, such as the ones in TSPLIB or the VLSI and National sets from http://www.math.uwaterloo.ca/tsp/data/. We found that LKH 2 and EAX found optimal solutions for most of these instances in the size range we considered for RUE within

running times quite consistent with our predictions. However, in a few cases, both solvers had substantial difficulties in finding the optimal solutions. This concerns in particular the TSPLIB instances `fl3795`, `u2319` and, to a lesser extent, `fl1577`. For these instances, either optimal solutions could not be found at all (notably, for `fl3795`), or solving times where orders of magnitude larger as would be expected from the confidence intervals obtained from our root-exponential models. One reason for these discrepancies lies in the fact that the structure of those instances is markedly different from those found in RUE and many other types of 2D Euclidean TSP instances, in that they contain a small number of well-separated, large clusters of cities or cities that are arranged in a grid pattern. We note that, although these instances are difficult for EAX or LKH, they are not necessarily hard to solve for other TSP solvers based on stochastic local search [14].

Our analysis could be extended in three directions. Firstly, it would be nice to reduce the uncertainty in analysis arising from the instances for which optimal solutions are not known. This could be done by attempting even longer runs for Concorde, although, considering the cutoff of 7 CPU days used to determine the optimal solutions used here, it is unclear how effective this would be. Another possibility would be to perform multiple long runs of Concorde on these instances using different pseudo-random number seeds, since it is known that running times can vary significantly with seeds. Yet another option would be run extensive experiments with cutting-edge inexact solvers to create pseudo-optimal solutions, and to then use these as a basis for scaling analysis. Although this may introduce errors, as these solution may in fact not be optimal, the very high performance of EAX and LKH 2 with restarts we observed when solving large RUE instances suggests that the incidence of such errors should be very small.

Secondly, we may build scaling models from support sets consisting of larger instances sizes, to reduce the possible impact of changes in the scaling behaviour as the instances become reasonably challenging. To asses the extent to which such impact may have coloured our findings, we have repeated our scaling analysis using the instance sets for $n = \{1000, 1100, 1200, 1400, 1500, 2000\}$ as the support data for fitting scaling models and found that the qualitative results remain similar, although the confidence intervals for model predictions and model parameters tend to be smaller.

Finally, we note that the inexact solvers we studied all have parameters that impact their performance and may affect scaling behaviour. To investigate to which extent optimising these parameters (or the heuristic mechanisms they control) may not only improve performance, but also the scaling of performance with instance size, is an interesting direction for future work.

Acknowledgments. This research and its results have been made possible through funding from the COMEX project (P7/36) within the Interuniversity Attraction Poles Programme of the Belgian Science Policy Office. HH acknowledges funding through an NSERC Discovery Grant. TS acknowledges support from the Belgian F.R.S.-FNRS, of which he is a Senior Research Associate.

5. REFERENCES

[1] D. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, Princeton, NJ, 2006.

[2] D. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. Concorde TSP solver. http://www.math.uwaterloo.ca/tsp/concorde.html, 2014. Version visited last on 15 April 2014.

[3] W. J. Cook. *In Pursuit of the Traveling Salesman*. Princeton University Press, Princeton, NJ, 2012.

[4] K. Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126:106–130, 2000.

[5] K. Helsgaun. General k-opt submoves for the Lin-Kernighan TSP heuristic. *Mathematical Programming Computation*, 1(2–3):119–163, 2009.

[6] H. H. Hoos. A bootstrap approach to analysing the scaling of empirical run-time data with problem size. Technical Report TR-2009-16, University of British Columbia, June 2009. Available on-line at http://www.cs.ubc.ca/ hoos/Publ/Hoos09.pdf.

[7] H. H. Hoos and T. Stützle. *Stochastic Local Search—Foundations and Applications*. Morgan Kaufmann Publishers, San Francisco, CA, 2005.

[8] H. H. Hoos and T. Stützle. On the empirical scaling of run-time for finding optimal solutions to the traveling salesman problem. *European Journal of Operational Research*, 238(1):87–94, 2014.

[9] H. H. Hoos and T. Stützle. On the empirical time complexity of finding optimal solutions *vs* proving optimality for Euclidean TSP instances. *Optimization Letters*, 2015, DOI: 0.1007/s11590-014-0828-5.

[10] L. Kotthoff, P. Kerschke, H. H. Hoos, and H. Trautmann. Improving the state of the art in inexact TSP solving using per-instance algorithm selection. In C. Dhaenens, L. Jourdan, and M.-E. Marmion, editors, *Proceedings of Learning and Intelligent OptimizatioN Conference, LION 9, Lille, France*, January 2015.

[11] P. Merz and B. Freisleben. Memetic algorithms for the traveling salesman problem. *Complex Systems*, 13(4):297–345, 2001.

[12] Y. Nagata and S. Kobayashi. Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem. In T. Bäck, editor, *ICGA*, pages 450–457. Morgan Kaufmann Publishers, San Francisco, CA, 1997.

[13] Y. Nagata and S. Kobayashi. A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. *INFORMS Journal on Computing*, 25(2):346–363, 2013.

[14] T. Stützle and H. H. Hoos. Analysing the run-time behaviour of iterated local search for the travelling salesman problem. In P. Hansen and C. Ribeiro, editors, *Essays and Surveys on Metaheuristics*, Operations Research/Computer Science Interfaces Series, pages 589–611. Kluwer Academic Publishers, Boston, MA, 2001.