Journal of Heuristics manuscript No.
(will be inserted by the editor)

On the Empirical Scaling of Running Time for Finding Optimal Solutions to the TSP

Zongxu Mu · Jérémie Dubois-Lacoste · Holger H. Hoos · Thomas Stützle

Received: date / Accepted: date

Abstract We study the empirical scaling of the running time required by state-of-the-art exact and inexact TSP algorithms for finding optimal solutions to Euclidean TSP instances as a function of instance size. In particular, we use a recently introduced statistical approach to obtain scaling models from observed performance data and to assess the accuracy of these models. For Concorde, the long-standing state-of-the-art exact TSP solver, we compare the scaling of the running time until an optimal solution is first encountered (the *finding time*) and that of the overall running time, which adds to the finding time the additional time needed to complete the proof of optimality. For two state-of-the-art inexact TSP solvers, LKH and EAX, we compare the scaling of their running time for finding an optimal solution to a given instance; we also compare the resulting models to that for the scaling of Concorde's finding time, presenting evidence that both inexact TSP solvers show significantly better scaling behaviour than Concorde.

1 Introduction

The travelling salesperson problem (TSP) is a well-known and widely studied $\mathcal{NP}$-hard problem that motivates sustained development of new algorithmic ideas in the domain of combinatorial optimisation. Presently, Concorde (Applegate et al, 2012) represents the long-standing state of the art in exact (or complete) TSP solving, *i.e.*, for finding optimal solutions to a given TSP instance and proving their optimality. In terms of inexact (or incomplete) TSP solvers, which may find optimal solutions but are unable to prove optimality, LKH (Helsgaun, 2000, 2009) had been the best available solver until the recent EAX (Nagata and Kobayashi, 2013), an evolutionary algorithm based on the effective recombination of short tours using a so-called edge assembly crossover operator. Empirical results show that EAX tends to perform better than LKH on a broad range of TSP instances, but does not dominate LKH, which is still more efficient in solving a substantial proportion of the instances (Kotthoff et al, 2015).

Relatively little work has been dedicated to investigating the empirical scaling of the running time of modern TSP solvers on interesting distributions of TSP instances. For exact solvers, an important observation was made in the book by Applegate et al (2006), where a graphical analysis of observed mean running times led to the claim that Concorde may scale exponentially. More recently, Hoos and Stützle (2014), after observing log-normally distributed running times over sets of RUE (random uniform Euclidean) instances of fixed size, found that the scaling of the median running time of Concorde is characterised by a function of the form $a \cdot b^{\sqrt{n}}$. Extending this work, Hoos and Stützle (2015) have investigated the fraction of Concorde's running time spent until an optimal

Zongxu Mu · Holger H. Hoos
Department of Computer Science, University of British Columbia, Vancouver, Canada
E-mail: {zongxumu,hoos}@cs.ubc.ca

Jérémie Dubois-Lacoste · Thomas Stützle
IRIDIA, CoDE, Université libre de Bruxelles, Brussels, Belgium
E-mail: {jeremie.dubois-lacoste,stuetzle}@ulb.ac.be

solution is first encountered and found that fraction to be larger than 0.5 in almost all cases, with a tendency for even larger values as instance size increases.

In this article, we significantly extend recent work, in which we studied the scaling of running times of EAX and LKH (enhanced with performance-improving restart mechanisms) on RUE instances (Dubois-Lacoste et al, 2015) and found that the scaling models obtained for EAX and LKH are of the same form as that of Concorde, namely $a \cdot b^{\sqrt{n}}$. Our overall goal is to obtain a detailed understanding of the state-of-the-art in exact and inexact TSP solving, in terms of statistically sound models of the scaling of the running time required for finding optimal solutions and for completing a proof of optimality.

In particular, we study the empirical performance scaling of EAX, LKH and Concorde, with the goal to characterise differences in the scaling behaviour of these state-of-the-art TSP solvers. Towards this end, we investigate the scaling of median running times required by these solvers for finding optimal solutions (without completing a proof of optimality) to 2-dimensional random uniform Euclidean (RUE) TSP instances. For brevity, in the context of Concord, we call those running times the *finding times*. We focus on 2D RUE instances for the same reasons as in previous work: they represent a widely studied distribution of TSP instances, they have characteristics that also occur in practical applications of the TSP, and they can be easily generated.

In particular, we investigate the following questions:

1. How do the finding times of Concorde, the state-of-the-art exact TSP solver, scale with instance size?
2. How does the scaling model for finding times of Concorde differ from that for overall running time (finding an optimal solution and proving optimality)?
3. Are the finding times for EAX and LKH, as well as the overall running times for Concorde strongly correlated across sets of TSP instances of the same size?
4. How do the running times of EAX and LKH scale with instance size? Are there qualitative differences between the performance scaling of EAX and LKH?
5. How do the scaling models of running times of EAX and LKH compare with that of Concorde? Do these state-of-the-art inexact solvers find optimal solutions (without proof of optimality) substantially faster than the state-of-the-art exact solver, and does the performance gap widen as instance size grows?

Regarding Questions 1 and 2, we will demonstrate that the scaling for the finding times of Concorde is well characterised by a root-exponential function of the form $a \cdot b^{\sqrt{n}}$ with $b \approx 1.250$. The scaling model for Concorde's overall running time is similar, but slightly worse. Regarding Question 3, our results do not indicate strong performance correlation between any of the solvers; even between EAX and LKH, the correlation coefficient in finding times never exceeds 0.5. Regarding Questions 4 and 5, we show that simple exponential scaling of the median running time of EAX and LKH can be ruled out. In particular, we chacterise the scaling of EAX by a root-exponential function of the form $a \cdot b^{\sqrt{n}}$ with $b \approx 1.123$, while we bound the scaling of LKH with a root-exponential function with $b \approx 1.188$ from above and by a polynomial function of the form $a \cdot n^b$ with $b \approx 2.93$ from below. The lower coefficients for the root-exponential scaling models estimated for EAX and LKH indicate a better scaling behaviour than that of Concorde for finding optimal solutions.

We note that our Question 1 differs from that investigated by Hoos and Stützle (2014), in that we consider *finding time* (as defined above) rather than total running time, and, based on the findings by Hoos and Stützle (2015), substantial differences in scaling between the two can be expected. Similarly, our Question 2 is addressed for the first time in this work and complements the earlier observations by Hoos and Stützle (2015). Questions 3 and 4 have been recently investigated in an earlier conference publication (Dubois-Lacoste et al, 2015); we include those results here, extended by a tightened analysis (made possible through the computation of additional, provably optimal solutions for the largest TSP instances considered in our experiments), in order to present a complete picture of the empirical scaling of state-of-the-art exact and inexact TSP solvers. Finally, Question 5 – investigated for the first time in this work, using a sophisticated and statistically sound scaling analysis approach – is of considerable practical and theoretical interest, since it concerns the choice of TSP algorithm when all that matters is to find optimal solutions, whilst a proof of optimality is not required.

By answering these questions, our study provides a complete and detailed picture of the performance scaling of state-of-the-art exact and inexact TSP algorithms with respect to finding optimal solutions and (in the case of exact solvers) proving optimality. It also demonstrates how recently introduced advanced empirical analysis methods can produce such results in a statistically meaningful way.

2 Experimental set-up and methodology

2.1 Benchmark instances

For our analysis, we used the same benchmark sets of RUE instances as in Dubois-Lacoste et al (2015); Hoos and Stützle (2014, 2015). These instances were generated using the portgen generator from the 8th DIMACS implementation challenge for TSP, by placing n points in a 100000×100000 square uniformly at random and computing Euclidean distances between these points. There are 1000 instances for each instance size $n \in \{500, 600, \cdots, 1500, 2000\}$ and 100 instances for each $n \in \{2500, 3000, \cdots, 4500\}$. RUE instances are among the most widely studied classes of Euclidean TSP instances, and it is known that the scaling behaviour of Concorde on these instances agrees well with that observed on other types of TSP instances (Hoos and Stützle, 2014). The instances have been made available at a supplementary data page Mu et al (2017).

2.2 TSP solvers

For our analysis, we chose the (arguably) three most prominent TSP solvers: Concorde (Applegate et al, 2012), EAX (Nagata and Kobayashi, 2013) and LKH (Helsgaun, 2000, 2009).

Concorde represents the long-standing state of the art in exact TSP solving. Based on a branch & cut scheme, it makes use of numerous heuristics and computes initial tours using a chained Lin-Kernighan local search procedure (Applegate et al, 2012). Concorde is the best-performing exact solver of which we are aware; it has been used to solve the largest non-trivial instances for which optimal solutions are currently known. As in previous work, for our experiments, we used Concorde in its default configuration, using QSopt version 1.01 as linear program solver.

The inexact TSP solver LKH, Helsgaun's variant of the Lin-Kernighan TSP heuristic, is a variable-depth search method utilising sophisticated, heuristically guided local search moves based on sequences of five or more edge exchanges. LKH can perform restarts based on perturbations of previously found solutions, using a variety of strategies (Helsgaun, 2000, 2009). For more than a decade, LKH represented the state-of-the-art for inexact TSP solving and has been used to find the best known solutions to many of the largest non-trivial TSP instances. In this work, we used LKH version 2.0.7, keeping parameters at their default values, except for PATCHING_A and PATCHING_C, which we set to 2 and 3, respectively.

Recently, performance improvements over LKH have been reported for EAX, an inexact TSP solver based on an evolutionary algorithm. EAX uses a recombination operator based on so-called edge assembly cross-over; furthermore, it exploits diversity preservation techniques and carefully initialises the population using local optimisation (Nagata and Kobayashi, 2013). In our work, we used the code for EAX that was made available by these authors and evaluated in their paper, including its default parameter settings, with a population size of 100 and a number of offspring generated set to 30.

We have enhanced LKH and EAX with a restart mechanism. It is known that even high-performance local search algorithms can suffer from stagnation behaviour (Stützle and Hoos, 2001), and we have observed that this can have a significant, detrimental effect on the performance of LKH and EAX, as illustrated in Figure 1 .

Based on these observations, we modified the original EAX to perform restarts whenever the termination criterion of the original version (as described in Nagata and Kobayashi (2013)) is met, and to terminate only when (i) a given target solution quality or (ii) a CPU time cutoff is reached. In the following, we simply refer to this variant as EAX; its efficacy can be seen in Figure 1. We also modified LKH with a mechanism that triggers restarts when within n iterations, where n is the size of the TSP instance to be solved, no improvement in tour length has been achieved; from here on, we use LKH to refer to this modified version of the original solver.

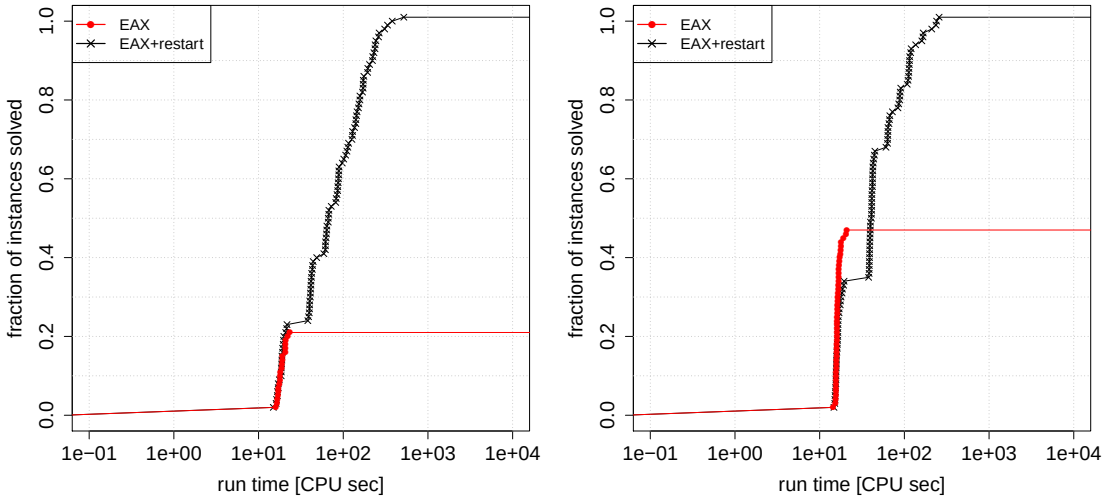

Figure 1 Effect of restarts: Distributions of running times with and without our restart mechanism over multiple independent runs of EAX on two RUE instances of size 1500; running times were measured on the reference machines specified in Section 2.4.

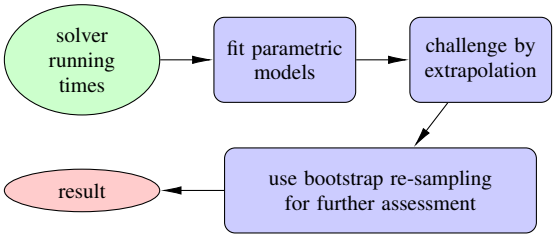

Figure 2 Methodology for scaling analysis

2.3 Empirical scaling analysis

This work, extending the existing line of research works on empirical scaling of TSP algorithms, is largely motivated by advancement in the methodology to study empirical scaling behaviours. The methodology, first introduced by Hoos (2009) and outlined in Figure 2, emphasises the importance of challenging scaling models derived from empirical performance data by extrapolation and makes use of bootstrap re-sampling to statistically assess scaling models.

In a nutshell, performance data are divided into disjoint sets of smaller support instances and larger challenge instances. Parametric scaling models are fitted (in our case, using the Levenberg-Marquardt numerical optimisation algorithm), to performance statistics (in our case, median running times of a given solver) computed over the support sets. Performance predictions for challenge instance sizes are then obtained from the fitted models and compared against the performance statistics observed on the challenge instance sets. To assess the statistical dependence of the scaling models on the given sets of support instances, those sets are re-sampled uniformly at random, with replacement, and model fitting is performed for each such bootstrap sample. From each model thus obtained, a performance prediction for every given challenge size is obtained, yielding a sample of performance predictions for each bootstrap sample and challenge instance size. From each sample P of predictions, a bootstrap percentile confidence interval at confidence level α is determined as $CI = [Q(0.5 - \alpha/2), Q(0.5 + \alpha/2)]$, where $Q(x)$ is the x-quantile of P. A parametric scaling model M is consistent with observed performance data on a challenge instance sets, if the latter fall into the confidence intervals thus obtained from M. Percentile confidence intervals for each model parameter can be derived from the models obtained for each bootstrap sample.

This methodology was used by Hoos and Stützle (2014) to study the scaling of the overall running times of Concorde (for finding an optimal solution and proving its optimality). Later, Mu and Hoos (2015b) adopted it to investigate the performance scaling of several prominent, high-performance SAT solvers. In that work, two useful extensions to the methodology were introduced: 1) computation of bootstrap confidence intervals for performance

data observed on challenge instances, to capture the variability of the performance statistics computed for these; and 2) the use of the using bootstrap confidence intervals for predicted running times, under a given scaling model, for one solver to assess whether the observed performance data of another solver are consistent with the same scaling model. The latter approach allows us to test the hypothesis that the empirical performance scaling of one solver differs significantly from that of another, provided we have a scaling model consistent with the observed performance of one of the solvers.

In the following, we say that a scaling model is fully consistent with the observed performance of a given solver, if the bootstrap confidence interval for predicted running times completely contains that for observed running times; that it is strongly consistent, if the point estimate for the observed performance falls within the confidence interval for predicted running times; and that it is weakly consistent, if the confidence intervals for predicted and observed running times overlap.

We consider the same parametric scaling models as in earlier work (Hoos and Stützle, 2014), namely:

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

We fit these models on median running times for instance sizes $n = 500, 600, \cdots, 1500$ and challenge them with median running times for $n = 2000, 2500, \cdots, 4500$. Furthermore, also as in earlier work, we generate $m = 1000$ bootstrap samples for each set of instances of a given size n and use a confidence level of $\alpha = 0.95$.

We also evaluated a 2-parameter scaling model of the form $a \cdot b^{\sqrt{n} \cdot \log(n)}$. However, this model failed to provide good fits for any of the solvers we considered, and we therefore do not present detailed results for it.

Our scaling analysis approach has recently been implemented as a fully automated tool called *Empirical Scaling Analyser (ESA)* (Mu and Hoos, 2015a). ESA takes sets of performance data as input, performs the scaling analysis automatically and outputs the results as a technical report. We used ESA for a substantial part of the scaling analyses that form the basis of this study.

2.4 Experimental set-up

We performed our experiments on a cluster of computers, each having two 2.0GHz eight-core AMD 6128 processor, 2 × 12MB L2/L3 cache and 16GB RAM, running Cluster Rocks Linux 6.0/CentOS 6.3. We used gcc 4.4.6 with optimisation flag `-O3` to compile all TSP solvers. As all three TSP solvers are fully sequential, our experiments were performed using a single CPU core. For each instance, we ran Concorde using default parameter settings and pseudo-random number seed 23.

As a first step, we tried to solve all instances in our instance set by running Concorde on the reference machines specified above with a CPU time limit of 7 CPU days. Unfortunately, some of our RUE benchmark instances could not be solved by Concorde within this time limit. Because EAX and LKH cannot prove optimality, we need to know provably optimal solution qualities in advance in order to determine the running time required by these inexact solvers to find an optimal solution. Therefore, to deal with the instances not solved by Concorde within 7 CPU days, we used (i) additional longer runs of Concorde we had executed in preliminary experiments on different machines as well as (ii) additional runs of Concorde on the previously unsolved instances with different pseudo-random number seeds[1] and/or additional runs on (a limited number of) faster machines. In addition, we performed multiple runs of EAX and LKH on those instances that still remain unsolved. For a subset of these instances, EAX and LKH reach the same best solution in every single run; we conjecture that these solutions are in fact optimal and refer to them as *pseudo-optimal*.

In the results that follow, we include data for instances for which we have optimal or pseudo-optimal solutions. We note that there are only 31 instances for which our analysis is limited to pseudo-optimal solutions (9 of size

[1] Note that the running times of Concorde follow a random distribution, as different search paths are taken for different pseudo-random number seeds. This variability in running time may be exploited through multiple parallel runs.

4000 and 22 of size 4500). For another 8 instances (2 of size 4000 and 6 of size 4500), LKH and EAX found different solutions after 24 CPU hours, and therefore we do not have solutions we consider to be pseudo-optimal. These instances are treated in the same way as in our earlier work (Dubois-Lacoste et al, 2015), where we use optimistic and pessimistic estimates of solver performance to obtain intervals in which the actual median running times must fall.

The running time data analysed in the following are all based on runs on our reference machines, with a cutoff time of 7 CPU days for Concorde and of 1 CPU day for LKH and EAX. Each instance for which we have an optimal or pseudo-optimal solution was then solved by LKH and EAX, 10 times each, using different pseudo-random number seeds (as these are inherently strongly stochastic algorithms), from which we computed the median running time per instance. Median running times for each set of RUE instances of a given size were then calculated from these data, making sure that missing data was treated in a way that ensures correct calculation (or intervals estimated, as noted above) of the medians.

3 Results for Concorde

3.1 Preparation of running time data

When studying the running times for finding optimal solutions (without completing the proof of optimality) compared to overall running times (including a complete proof of optimality), care needs to be taken in the treatment of possible time-out runs. In the case of Concorde, this concern applies to instances for which only or not even pseudo-optimal solutions are available. By comparing the best solution found during a run of Concorde to the pseudo-optimal solution or the best solution known for an instance (from runs of EAX or LKH), we were able to verify for all such instances except for one Concorde was unable to produce a solution of this target quality within cutoff of the 7 CPU days. We therefore know that Concorde's finding times for these instances are necessarily larger than 7 CPU days on our reference machines, and we factored those instances into our median calculations to reflect this fact. For the one remaining instances, for which Concorde found a solution of the best known quality within the cutoff time, we verified that the time required for reaching that solution quality was high enough compared to the finding time on all other instances that the median could not have been affected.

3.2 Scaling of Concord's median finding time

We first fitted parametric models to the median finding times of Concorde on our support instance sets, which resulted in the scaling models shown in Table 1. According to the RMSE (root-mean squared error) on the challenge sets, the root exponential model provides the best fit to the observed data.

To assess the confidence we should have in these models, we further used bootstrap re-sampling, as explained in Section 2.3. The resulting confidence intervals for model parameters are shown in Table 2, while Table 3 provides the confidence intervals for observed and predicted running times on our sets of challenge instances. These results, further illustrated in Figure 3, clearly indicate that only the root-exponential model is consistent with the observed running time data.

Solver	Model		RMSE (support)	RMSE (challenge)
Concorde	Exp.	4.0388×1.0032^{n}	7.7847	2.7852×10^{6}
	RootExp.	$\mathbf{0.083457 \times 1.2503^{\sqrt{n}}}$	**7.0439**	**9169.4**
	Poly.	$1.6989 \times 10^{-10} \times n^{3.9176}$	9.9327	1.038×10^{5}

Table 1 Scaling models for median finding time of Concorde on RUE support instance sets and corresponding RMSE values (in CPU sec) on support and challenge sets. The model yielding the most accurate predictions (as per RMSEs on challenge data) is shown in boldface.

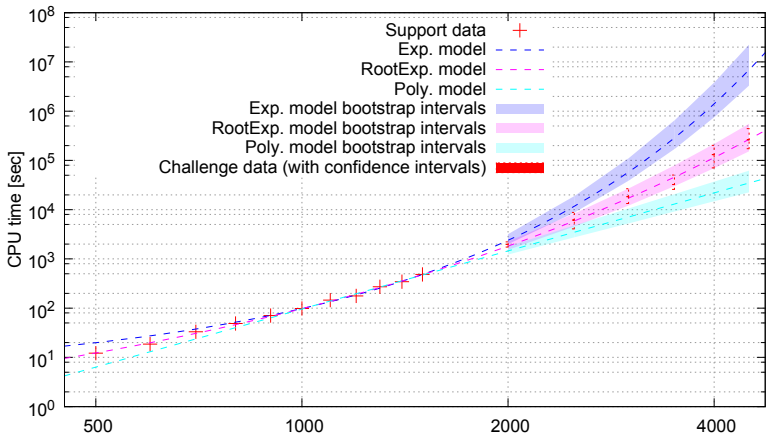

Figure 3 Illustration of scaling models for Concorde's median finding time on RUE instances and observed median finding times.

3.3 Comparison to scaling of Concorde's overall running time

Previously, Hoos & Stützle have demonstrated that the median overall running time of Concorde, measured on a set of reference machines different from that used here, shows root-exponential scaling. Here, we repeated this analysis for the newer overall running times collected along with the finding times considered above. The results we thus obtained confirmed root-exponential scaling of the median overall running time of Concorde, with $a \in [0.11658, 0.35323]$ and $b \in [1.2126, 1.2522]$. Note that the confidence interval for b is very close to that obtained for median finding time, which is $[1.2287, 1.2793]$. Moreover, the predictions made by the root-exponential model of overall running time, as shown in Table 4, are quite consistent with observed finding times.

Overall, we found no evidence that finding time might scale better than overall running time. On the contrary, our models suggest that finding time scales slightly worse than overall running time. A closer look at the data reveals that the observed finding times are usually closer to the lower end of the corresponding bootstrap confidence intervals, while the observed proving times are usually closer to the higher end. These observations are consistent with the earlier findings of Hoos & Stützle that, as n grows, the finding time increasingly dominates Concorde's overall running times (Hoos and Stützle, 2015). To capture this effect more accurately in our scaling models, we would likely have to consider more complex models containing lower-order terms.

Solver	Model	Confidence interval of a	Confidence interval of b
Concorde	Exp.	$[2.6108, 5.2975]$	$[1.0030, 1.0036]$
	RootExp.	$[0.037056, 0.15111]$	$[1.2287, 1.2793]$
	Poly.	$[6.1872 \times 10^{-12}, 1.7351 \times 10^{-9}]$	$[3.5859, 4.3713]$

Table 2 95% bootstrap confidence intervals for the parameters of the scaling models for Concorde's median finding time on RUE instances.

Solver	n	Predicted confidence intervals			Observed median run-time	
		Exp. model	RootExp. model	Poly. model	Point estimates	Confidence intervals
Concorde	2000	$[\mathbf{1988}, \mathbf{3179}]$	$[\mathbf{1528}, \mathbf{2269}]$*	$[\mathbf{1228}, \mathbf{1795}]$	1969	$[1739, 2222]$
	2500	$[\mathbf{8718}, \mathbf{1.884 \times 10^4}]$	$[\mathbf{4536}, \mathbf{8335}]$#	$[\mathbf{2737}, \mathbf{4771}]$	6149	$[4084, 8812]$
	3000	$[3.853 \times 10^4, 1.103 \times 10^5]$	$[\mathbf{1.212 \times 10^4}, \mathbf{2.694 \times 10^4}]$*	$[5252, 1.057 \times 10^4]$	1.84×10^4	$[1.332 \times 10^4, 2.669 \times 10^4]$
	3500	$[1.698 \times 10^5, 6.479 \times 10^5]$	$[\mathbf{3.001 \times 10^4}, \mathbf{7.925 \times 10^4}]$#	$[9149, 2.069 \times 10^4]$	3.246×10^4	$[2.581 \times 10^4, 5.038 \times 10^4]$
	4000	$[7.5 \times 10^5, 3.809 \times 10^6]$	$[\mathbf{6.95 \times 10^4}, \mathbf{2.163 \times 10^5}]$*	$[1.477 \times 10^4, 3.708 \times 10^4]$	1.312×10^5	$[7.073 \times 10^4, 2.024 \times 10^5]$
	4500	$[3.301 \times 10^6, 2.245 \times 10^7]$	$[\mathbf{1.528 \times 10^5}, \mathbf{5.563 \times 10^5}]$*	$[2.248 \times 10^4, 6.205 \times 10^4]$	2.633×10^5	$[1.73 \times 10^5, 4.419 \times 10^5]$

Table 3 95% bootstrap confidence intervals for Concorde's predicted and observed median finding times on RUE instances. The instance sizes shown here are larger than those used for fitting the models. Bootstrap intervals for predictions that are weakly consistent with the observed data are shown in boldface, those that are strongly consistent are marked by sharps (#), and those that fully contain the confidence intervals for observations are marked by asterisks (*).

Solver	n	Predicted confidence intervals RootExp. model for proving time	Observed median proving time Point estimates	Observed median proving time Confidence intervals	Observed median finding time Point estimates	Observed median finding time Confidence intervals
Concorde	2000	$[\mathbf{1962, 2736}]$#	2508	$[2197, 2760]$	1969	$[1739, 2222]$
	2500	$[\mathbf{5431, 8914}]$#	7899	$[4886, 9789]$	6149	$[4084, 8812]$
	3000	$[\mathbf{1.366 \times 10^4, 2.612 \times 10^4}]$#	2.064×10^4	$[1.492 \times 10^4, 2.795 \times 10^4]$	1.84×10^4	$[1.332 \times 10^4, 2.669 \times 10^4]$
	3500	$[\mathbf{3.181 \times 10^4, 6.994 \times 10^4}]$#	4.057×10^4	$[2.586 \times 10^4, 5.719 \times 10^4]$	3.246×10^4	$[2.581 \times 10^4, 5.038 \times 10^4]$
	4000	$[\mathbf{6.987 \times 10^4, 1.753 \times 10^5}]$#	1.377×10^5	$[8.236 \times 10^4, 2.108 \times 10^5]$	1.312×10^5	$[7.073 \times 10^4, 2.024 \times 10^5]$
	4500	$[\mathbf{1.462 \times 10^5, 4.154 \times 10^5}]$#	3.264×10^5	$[1.953 \times 10^5, 5.087 \times 10^5]$	2.633×10^5	$[1.73 \times 10^5, 4.419 \times 10^5]$

Table 4 95% bootstrap confidence intervals for predicted and observed median overall running times of Concorde on RUE instances. The instance sizes shown here are larger than those used for fitting the models. Bootstrap intervals for predictions that are weakly consistent with the observed finding times are shown in boldface, those that are strongly consistent are marked by sharps (#), and those that fully contain the confidence intervals for observations are marked by asterisks (*).

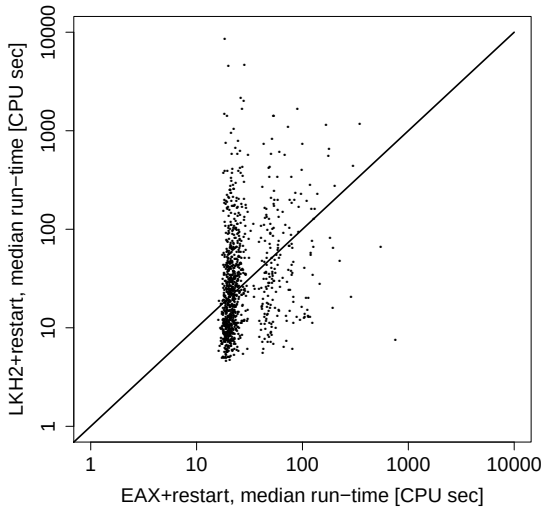

Figure 4 Median running time of EAX *vs* LKH 2.0.7 on RUE instances of size 1500.

n	500	600	700	800	900	1000	1100	1200	1300	1400	1500	2000	2500	3000	3500	4000	4500
EAX *vs* Concorde	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 *vs* Concorde	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
EAX *vs* LKH	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

Table 5 Pearson correlation coefficients between median running times of EAX, LKH, and Concorde on sets of RUE instances of size n.

4 Results for EAX and LKH

4.1 Performance correlations

As a next step in our analysis, we examined the correlation of the median running times between our two inexact solvers and between these and the overall running time of Concorde. We note that the correlation of the median running times of Concorde and the inexact solvers is typically very low, as can be noted from low correlation coefficients for all instance sizes reported in Table 4.1.

More surprisingly, the performance correlation between EAX and LKH also tends to be very low, as can be also seen in Figure 4, and in additional plots presented in Dubois-Lacoste et al (2015). This suggests that, *a priori*, there is no reason to expect similar scaling of the performance of these TSP solvers.

4.2 Scaling of median running time for EAX and LKH

We first fitted parametric models to the median running times of EAX and LKH, resulting in the models shown in Table 6. Based on RMSE values on challenge instance sets, the root-exponential and the polynomial model yield

Solver	Model		RMSE (support)	RMSE (challenge)
EAX	Exp.	1.6512×1.0017^{n}	0.80329	$[1513.3, 1566.2]$
	RootExp.	$\mathbf{0.24795 \times 1.123^{\sqrt{n}}}$	**0.45614**	$[\mathbf{44.77, 88.739}]$
	Poly.	$1.9196 \times 10^{-5} \times n^{1.9055}$	0.10699	$[235.79, 287.6]$
LKH	Exp.	$0.56147 \times 1.0025^{n}$	0.51265	$[18213, 18330]$
	RootExp.	$0.030075 \times 1.1879^{\sqrt{n}}$	0.38383	$[797.7, 909.51]$
	Poly.	$\mathbf{1.15 \times 10^{-8} \times n^{2.9297}}$	**0.50193**	$[\mathbf{282.12, 378.53}]$

Table 6 Scaling models for median running times of EAX and LKH and corresponding RMSE values on predicted and observed performance data (in CPU sec). The models yielding the most accurate predictions (as per RMSEs on challenge data) are shown in boldface.

Solver	Model	Confidence interval of a	Confidence interval of b
EAX	Exp.	$[1.6234, 1.6764]$	$[1.0017, 1.0018]$
	RootExp.	$[0.23938, 0.25592]$	$[1.1219, 1.1242]$
	Poly.	$[1.6803 \times 10^{-5}, 2.1556 \times 10^{-5}]$	$[1.8887, 1.9245]$
LKH	Exp.	$[0.46665, 1]$	$[1.0021, 1.0027]$
	RootExp.	$[0.020678, 0.043847]$	$[1.1749, 1.2006]$
	Poly.	$[2.769 \times 10^{-9}, 4.8245 \times 10^{-8}]$	$[2.7229, 3.1287]$

Table 7 95% bootstrap confidence intervals for the model parameters of the scaling models for median running time of EAX and LKH on RUE instances.

Solver	n	Predicted confidence intervals			Observed median run-time	
		Exp. model	RootExp. model	Poly. model	Point estimates	Confidence intervals
EAX	2000	[53.08, 54.75]	[43.77, 45.01]	[37.02, 37.98]	41.24	[40.03, 42.26]
	2500	[125.9, 131.9]	[**80.33, 83.51**]	[56.43, 58.35]	73.19	[61.11, 118]
	3000	[298.7, 317.8]	[139, 146]	[79.63, 82.87]	172.2	[155.7, 223.2]
	3500	[709.2, 765.6]	[**230.3, 244**]#	[106.5, 111.5]	239.9	[220, 357.7]
	4000	[1682, 1845]	[**368.4, 393.6**]	[137.1, 144.1]	[483.2, 547.4]	[370.2, 649.8]
	4500	[3991, 4445]	[**572.8, 616.7**]+	[171.3, 180.8]	[611.7, 727.7]	[520, 877.6]
LKH	2000	[**62.15, 95.52**]#	[**58.8, 73.94**]#	[**48.05, 59.71**]	62.64	[58.03, 69.61]
	2500	[**174.5, 360.2**]	[**138.1, 193.6**]	[**88.54, 119.8**]	137	[108.5, 199.7]
	3000	[490, 1361]	[**299.3, 462.5**]	[**145.9, 211.4**]	249.4	[201.3, 372.2]
	3500	[1376, 5154]	[**609.3, 1030**]	[**222.4, 342.4**]	382.8	[260.8, 648.8]
	4000	$[3863, 1.954 \times 10^{4}]$	[**1176, 2178**]	[320.7, 519.9]	[891, 907.3]	[551.5, 1207]
	4500	$[1.085 \times 10^{4}, 7.412 \times 10^{4}]$	[**2173, 4391**]	[442.8, 751.6]	[1059, 1352]	[808.2, 2203]

Table 8 95% bootstrap confidence intervals for predicted and observed median running times of EAX and LKH on RUE instances. The instance sizes shown here are larger than those used for fitting the models. Bootstrap intervals for predictions that are weakly consistent with the observed data are shown in boldface, those that are consistent are marked by plus signs (+), those that are strongly consistent are marked by sharps (#), and those that fully contain the confidence intervals for observations are marked by asterisks (*).

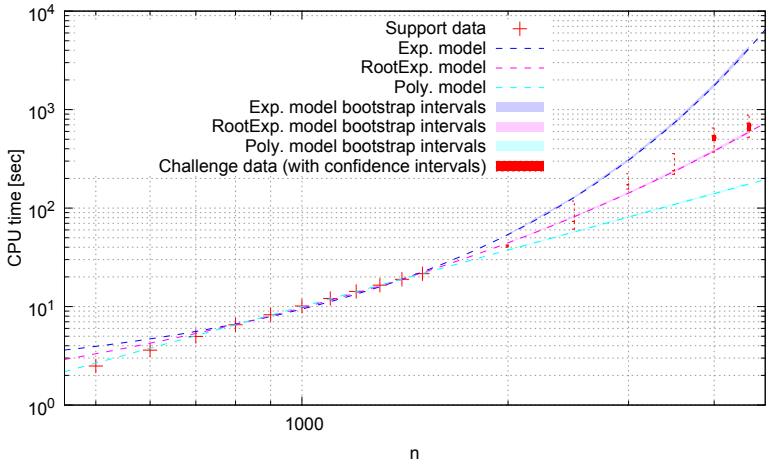

Figure 5 Illustration of scaling models for median running time of EAX on RUE instances and observed median running times.

the most accurate performance predictions for EAX and LKH, respectively. To further assess these models, we determined bootstrap confidence intervals for model parameters (Table 7) as well as for predicted and observed running times (Table 8). The results for EAX, also presented graphically in Figure 5, indicate that the root-exponential model is fairly consistent with the observed performance data, while the other two models should be rejected. The performance scaling of LKH, on the other hand, falls between the polynomial and root-exponential models (see also Figure 6), which bound it from below and above, respectively.

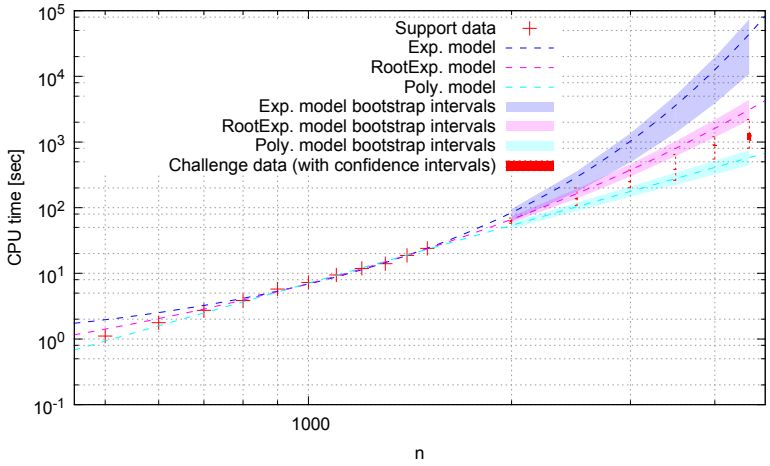

Figure 6 Illustration of scaling models for median running time of LKH on RUE instances and observed median running times.

4.3 Comparison to scaling of Concorde's finding time

To compare the scaling of the median finding time required by Concorde on RUE instances to that required by EAX and LKH, we first compared the bootstrap intervals for the values of b in the respective root-exponential models. For Concorde, we have $b \in [1.2255, 1.2747]$, which is significantly larger than for EAX ($b \in [1.1219, 1.1242]$) and LKH ($b \in [1.1749, 1.2006]$), where in the case of LKH, the root-exponential model bounds the median running time from above. These observations suggest that the performance of Concorde scales significantly worse than that of EAX and LKH.

To further test this hypothesis, we fitted a one-parameter model of the form $a \cdot b_{Concorde}^{\sqrt{n}}$, where $b_{Concorde} = 1.2503$, to the performance data for EAX and LKH, respectively. The resulting models, shown in Figure 7, clearly over-estimate the running times of EAX and LKH, which support the claim that the finding times of Concorde scale significantly worse than those of EAX and LKH – in other words, the inexact solvers find optimal solutions to RUE instances not only faster than Concorde, but their performance advantage increases with instance size.

We also compared the scaling of EAX with that of LKH. As illustrated in Figure 8, LKH clearly scales worse than EAX, in that LKH performs better than EAX for small instances, but clearly worse for $n \geq 1500$.

5 Conclusions

In this article, we studied the empirical scaling of several state-of-the-art TSP solvers for finding optimal solutions of RUE instances.

For Concorde, a root-exponential model of the form $a \cdot b^{\sqrt{n}}$, with b being around 1.25, is believed to best describe the scaling behaviour. Comparing with the scaling models for proving times, we found that the finding times of Concorde scale slightly worse than the proving times, which can be seen from the fact that b is slightly larger for the model of finding time (1.25032 *vs* 1.24194), and so is the bootstrap interval ($[1.22803, 1.276]$ *vs* $[1.2212; 1.2630]$). Moreover, it can be seen as consistent with the conclusion that the fraction of finding time approaches 1 as n increases (Hoos and Stützle, 2015). However, we note that better capturing this property may require more sophisticated models with second-order terms.

EAX also scales root-exponentially, best described by a model with b around 1.123. For LKH, the scaling seems also to be root-exponential, but we cannot rule out a polynomial model with high confidence. For both solvers, there is evidence that they scale better than Concorde for finding the optimal solutions of RUE instances. In other words, if we treat Concorde as an inexact solver, then it scales significantly worse than EAX and LKH. We are

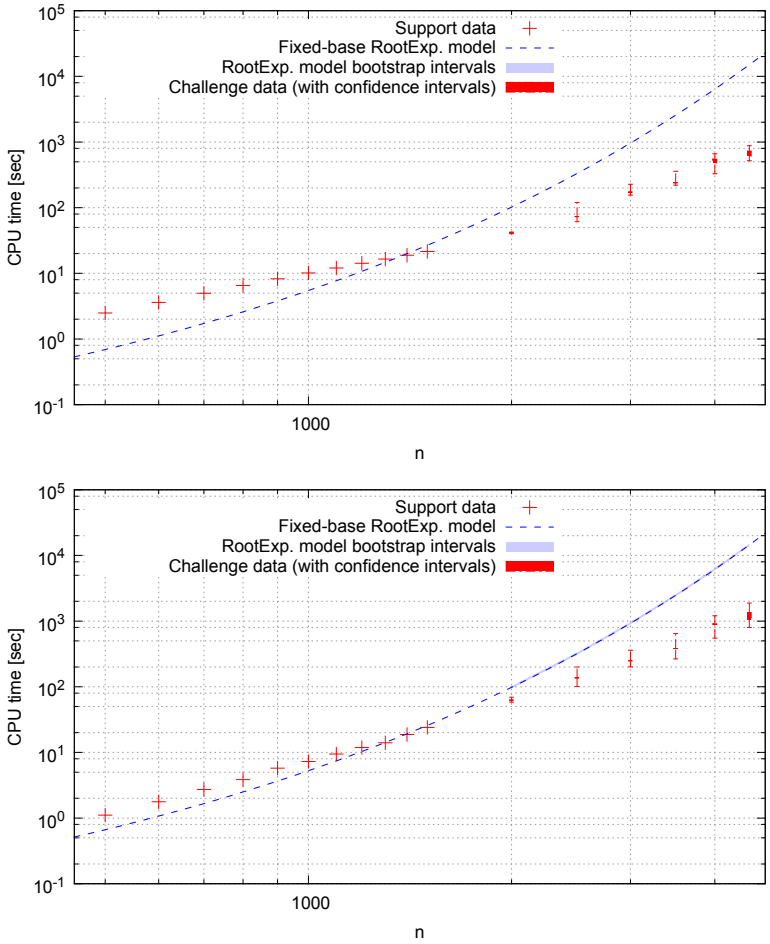

Figure 7 One-parameter models (of the form $a \cdot b^{\sqrt{n}}_{Concorde}$) for the median running time of EAX (top) and LKH (bottom) on RUE instances.

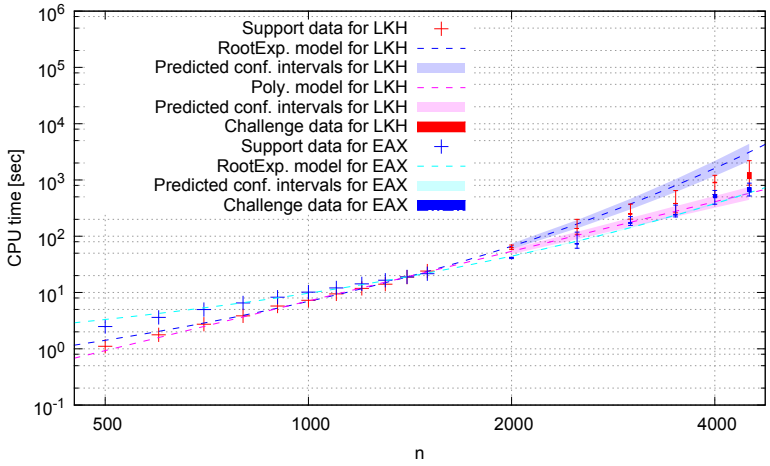

Figure 8 Comparison of the best fitted models for the median running time of EAX and LKH on RUE instances.

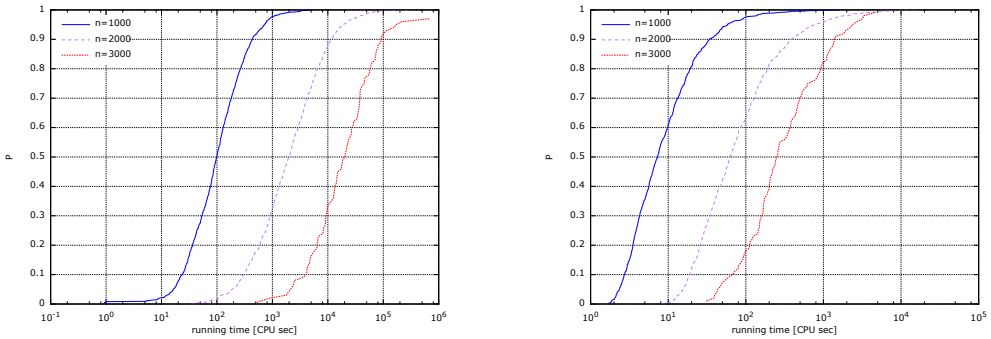

Figure 9 Distributions of running times for finding optimal solutions over RUE instances of fixed size for Concorde (left) and for LKH (right); in the case of LKH, running times are medians over 10 independent runs per TSP instance.

hopeful that by exploiting solutions found by high-performance inexact solvers, better exact algorithms can be constructed for the TSP problem.

For our scaling analyses, we have focussed on median running time over sets of RUE instances, since the median, as a robust measure of location of the underlying probability distribution, is a standard statistic considered routinely in theoretical and practical contexts. One might wonder whether quantiles other than the median, and, in particular, higher quantiles exhibit qualitatively different scaling behaviour. Unfortunately, reasonably stable estimates for those higher quantiles require substantially larger numbers of instances to be considered, which, in light of the computation times involved, would have far exceeded the computational resources we could muster for this study. Still, the empirical distributions of running times over the complete sets of RUE instances should reflect any substantial, qualitative differences in scaling between the median and higher quantiles. As shown in Figure 9 for Concorde (finding time) and LKH, there is no evidence for such differences. In fact, these CDFs are consistent with the hypothesis that the difference in scaling between the median and higher quantiles, such as $Q_{0.75}$ and $Q_{0.9}$, is characterised by a multiplicative constant – *i.e.*, that the ratio between those quantiles and the median is constant, regardless of instance size. The same holds for Concorde's overall running time (see supplementary material). For EAX, the situation is slightly different, since the efficacy of the restart mechanism changes with instance size, which leads to multi-modal distributions of running times over instance sets, with higher modes becoming more prominent with growing instance size. As a result, the ratio between higher quantiles and the median is not a simple constant; yet, there is no conclusive evidence for qualitatively different scaling. Despite the fact that empirical estimates for the higher quantiles are necessarily less stable, we also performed the same detailed scaling analysis for $Q_{0.75}$ and $Q_{0.9}$ as we reported earlier for the median running times, and again, the results did not suggest substantially worse scaling of those higher quantiles. (Detailed results of these analyses can be found in our supplementary material Mu et al (2017)).

Finally, we have considered the default parameter settings of Concorde, LKH and EAX, as recommended by their respective authors. We note that all three solvers might benefit automatically configuring their parameters, and indeed, it has recently been shown that this is the case for EAX Mu et al (2016). So far, automatic configuration has not produced similar improvements for LKH, and for Concorde, to the best of our knowledge, similar work has not yet been undertaken. Hence, in future work, it may be worthwhile to examine more carefully how automatic configuration, maybe in dependence of instance size, may impact the scaling behaviour of the TSP solvers we have considered here.

Acknowledgements This work received support from the COMEX project within the Interuniversity Attraction Poles Programme of the Belgian Science Policy Office. Thomas Stützle acknowledges support from the Belgian F.R.S.-FNRS, of which he is a senior research associate.

References

Applegate DL, Bixby RE, Chvátal V, Cook WJ (2006) The Traveling Salesman Problem: A Computational Study. Princeton University Press

Applegate DL, Bixby RE, Chvátal V, Cook WJ (2012) The traveling salesman problem, concorde TSP solver. http://www tsp gatech edu/concorde

Dubois-Lacoste J, Hoos HH, Stützle T (2015) On the empirical scaling behaviour of state-of-the-art local search algorithms for the Euclidean TSP. In: Proceedings of GECCO 2015, ACM Press, pp 377–384

Helsgaun K (2000) An effective implementation of the Lin–Kernighan traveling salesman heuristic. European Journal of Operational Research 126(1):106–130

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

Hoos HH (2009) A bootstrap approach to analysing the scaling of empirical run-time data with problem size. Tech. rep., TR-2009-16, Department of Computer Science, University of British Columbia

Hoos HH, Stützle T (2014) On the empirical scaling of run-time for finding optimal solutions to the travelling salesman problem. European Journal of Operational Research 238(1):87–94

Hoos HH, Stützle T (2015) On the empirical time complexity of finding optimal solutions vs proving optimality for Euclidean TSP instances. Optimization Letters

Kotthoff L, Kerschke P, Hoos H, Trautmann H (2015) Improving the state of the art in inexact tsp solving using per-instance algorithm selection. In: Proceedings of LION 2015, Springer, pp 202–217

Mu Z, Hoos HH (2015a) Empirical scaling analyser: An automated system for empirical analysis of performance scaling. In: GECCO 2015, Companion, pp 771–772

Mu Z, Hoos HH (2015b) On the empirical time complexity of random 3-SAT at the phase transition. In: Proceedings of IJCAI 2015, pp 367–373

Mu Z, Hoos HH, Stützle T (2016) The impact of automated algorithm configuration on the scaling behaviour of state-of-the-art inexact TSP solvers. In: Festa P, Sellmann M, Vanschoren J (eds) Learning and Intelligent Optimization, 10th International Conference, LION 10, Lecture Notes in Computer Science, vol 10079, Springer Verlag, Heidelberg, Germany, pp 157–172

Mu Z, Dubois-Lacoste J, Hoos HH, Stützle T (2017) On the empirical scaling of running time for finding optimal solutions to the tsp: Supplementary material. http://iridia.ulb.ac.be/supp/IridiaSupp2017-010/

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

Stützle T, Hoos HH (2001) Analysing the run-time behaviour of iterated local search for the travelling salesman problem. In: Hansen P, Ribeiro C (eds) Essays and Surveys on Metaheuristics, Kluwer Academic Publishers, pp 589–611