The Impact of Automated Algorithm Configuration on the Scaling Behaviour of State-of-the-Art Inexact TSP Solvers

Zongxu Mu[1(✉)], Holger H. Hoos[1(✉)], and Thomas Stützle[2]

[1] Department of Computer Science, University of British Columbia, Vancouver, BC, Canada
{zongxumu,hoos}@cs.ubc.ca
[2] IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
stuetzle@ulb.ac.be

Abstract. Automated algorithm configuration is a powerful and increasingly widely used approach for improving the performance of algorithms for computationally hard problems. In this work, we investigate the impact of automated algorithm configuration on the scaling of the performance of two prominent inexact solvers for the travelling salesman problem (TSP), EAX and LKH. Using a recent approach for analysing the empirical scaling of running time as a function of problem instance size, we demonstrate that automated configuration impacts significantly the scaling behaviour of EAX. Specifically, by automatically configuring the adaptation of a key parameter of EAX with instance size, we reduce the scaling of median running time from root-exponential (of the form $a \cdot b^{\sqrt{n}}$) to polynomial (of the form $a \cdot n^b$), and thus, achieve an improvement in the state of the art in inexact TSP solving. In our experiments with LKH, we noted overfitting on the sets of training instances used for configuration, which demonstrates the need for more sophisticated configuration protocols for scaling behaviour.

1 Introduction

The travelling salesperson problem (TSP) is a well known and widely studied $\mathcal{NP}$-hard problem. Given a set of cities and pair-wise distances between them, the objective of the TSP is to find the shortest round trip that visits each city exactly once. TSP algorithms are usually categorised into two kinds: exact algorithms, which are guaranteed to find an optimal solution to any TSP instance and can prove the optimality of the solution, and inexact algorithms, which may find optimal solutions but cannot prove optimality. Presently, Concorde [2] represents the long-standing state of the art among exact TSP algorithms. In terms of inexact TSP algorithms, LKH [5,6] had been the best available solver until the recent introduction of EAX [20], an evolutionary algorithm that makes uses of an improved edge assembly crossover operator [19] for recombining short tours. Empirical results show that EAX tends to perform better than LKH on a broad range of TSP instances [20]; however, it has been shown recently that

P. Festa et al. (Eds.): LION 2016, LNCS 10079, pp. 157–172, 2016.
DOI: 10.1007/978-3-319-50349-3_11

LKH is not dominated by EAX in that there are many instances for which LKH finds optimal solutions more efficiently than EAX [14].

In theoretical computer science, time complexity is arguably the most important concept for analysing and understanding the difficulty of problems and the performance of algorithms. The time complexity of an algorithm is characterised by the scaling of the time required for solving a problem instance as a function of instance size. In spite of the significant role that theoretical methods play in understanding the complexity of problems and algorithms, many high-performance algorithms are beyond the reach of such methods, and therefore have to be studied using principled empirical approaches.

In this work, we investigate the question whether and to which extent the empirical scaling of the running time of state-of-the-art inexact TSP solvers EAX and LKH changes as the parameter settings of these solvers are optimised. This question is particularly relevant as automated procedures for optimising parameter settings (so-called algorithm configurators) are now readily available and used increasingly frequently in the development of state-of-the-art solvers for computationally challenging problems as well as for customisation of such solvers for particular application contexts (see, *e.g.*, [1,3,10,11]). To study the scaling behaviour of EAX and LKH, we use an advanced empirical scaling analysis approach that challenges automated fitted scaling models by extrapolation and uses bootstrap re-sampling to statistically assess scaling models [7,18]. Our main findings are as follows:

- automated algorithm configuration can significantly improve the scaling behaviour of EAX, and by adapting the population size with instance size, the empirical time complexity of EAX is reduced from root-exponential (of the form $a \cdot b^{\sqrt{n}}$) to polynomial (of the form $a \cdot n^b$);
- the state of the art in inexact TSP solving can thus be improved: for instance, we reduce the median running time of EAX for solving instances of size $n = 4500$ (three times larger than those used for training the configurator) by about a factor of 1.13;
- automated algorithm configuration can significantly impact the scaling of LKH, but configuring LKH suffers from overfitting that leads to improved running times for small instances but worse performance for larger ones.

In the remainder of this work, we first describe the benchmark instances, algorithms and methods that we use in our experiments (Sect. 2); next, we present in detail our results (Sect. 3); and finally, we draw some general conclusions and briefly outline avenues for future work (Sect. 4).

2 Instances, Algorithms and Methods

2.1 Benchmark Instances

2D Euclidean TSP instances, *i.e.*, instances where the locations to be visited correspond to points in the Euclidean plane, often occur in practical applications.

A particularly widely studied type of 2D Euclidean TSP instances are obtained by placing cities uniformly at random in a square. The so-called RUE instances thus obtained are known to have properties similar to a broad range of other 2D Euclidean instances and represent a challenging, widely used benchmark for TSP solvers [12,13]. In the following, we use the benchmark sets of RUE instances generated and studied earlier by Hoos and Stützle [8,9]. These instances were generated using the portgen generator from the 8th DIMACS implementation challenge for TSP, which places n points in a $100\,000 \times 100\,000$ square uniformly at random and computes the Euclidean distances between pairs of points. There are 1000 instances for each instance size $n = 500, 600, \ldots, 1500, 2000$ and 100 instances for each $n = 2500, 3000, \ldots, 4500$.

2.2 Inexact Algorithms for TSP

The inexact TSP solvers we selected for our study are the latest versions of LKH and EAX. LKH [5,6], Helsgaun's variant of the Lin-Kernighan TSP heuristic, is a variable-depth search algorithm that performs sophisticated heuristically guided local search moves based on sequences of five or more edge exchanges. It restarts the local search based on perturbations of previously found solutions using various strategies. LKH represents a milestone in the development of inexact TSP solvers and is arguably the most prominent method for finding finding high-quality solutions to challenging TSP instances. In this work, we used LKH version 2.0.7, keeping all parameters at their default values, except for PATCHING_A and PATCHING_C, which we set to 2 and 3, respectively, to include patching of cycles in searching for improving moves. These values were also adopted in the example parameter file for solving TSPLIB instance pr2392 and used in earlier work studying LKH [4].

EAX [20] is a recent evolutionary algorithm that makes use of improved variants of the edge assembly cross-over recombination operator. It also exploits diversity preservation techniques and initialises the initial population by local optimisation. For our experiments, we used EAX with default parameter settings, namely with population size set to 100 and the number of offsprings generated per recombination attempt set to 30.

For our analyses, we used the same modified implementations of LKH and EAX as Dubois-Lacoste *et al.* [4], who enhanced the original solvers with a restart mechanism to achieve improved performance. This type of modification is a simple, yet effective means for overcoming stagnation behaviour often encountered in stochastic local search algorithms [21]; in the case of LKH and EAX, the added restart mechanism helps considerably in finding optimal solutions more efficiently.

2.3 Algorithm Configurator

To automatically configure the parameters of EAX and LKH, we used SMAC, a prominent, state-of-the-art algorithm configurator [10]. SMAC is based on a sequential model-based optimisation procedure that builds and iteratively refines

a statistical model mapping parameter configurations of a given algorithm to performance predictions. This empirical performance model is used to select promising configurations in each iteration of SMAC; these configurations are then run, and the performance values thus observed are used to update the model. The standard version of SMAC, as used in our experiments for configuring EAX and LKH, uses random regression forests to model performance.

2.4 Scaling Analysis

We use a recent boostrap approach for studying the empirical scaling of algorithm performance with input size [7]. It automatically fits scaling models to a set of support data and then challenges these models by testing the performance predictions obtained from them for larger input sizes. Most importantly, it uses a re-sampling approach to assess the models and their predictions in a statistically meaningful way. This approach has been used to characterise the scaling behaviour of state-of-the-art exact and inexact TSP algorithms [4,8] and to study the empirical time complexity of state-of-the-art solvers for the propositional satisfiability problem (SAT) [18]. In this latter work, Mu and Hoos extended the approach to compare scaling models based on bootstrap confidence intervals for predicted and observed running times and to assess differences in the scaling models for two given algorithms. To perform this type of scaling analysis for different configurations of EAX and LKH, we used the ESA system [17].

2.5 Computing Environment and Experimental Setup

For the automatic configuration of EAX and LKH, we used a standard protocol, according to which we performed 25 independent runs of SMAC for each scenario and selected the best parameter configuration according to the performance on the given set of training instances for scaling analysis. The cut-off time for each run of the algorithm being configured and the overall time budget for each run of SMAC differ between our experiments for EAX and LKH, and we report them as we discuss each experiment.

For our scaling analysis, we considered three parametric models:

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

Models were fitted to performance observations in the form of medians of the distributions of running times over sets of instances for given n. Compared to the mean, the median has two advantages: it is statistically more stable and immune to the presence of a certain number of timed-out runs. We performed 10 independent runs per instance and used the median over those 10 running times as the running time for the respective instance. Our approach could be easily extended to other scaling models, but, as we will show in the following, these models jointly characterise the scaling observed in all our experiments, and,

thus, we saw no need to consider different or more complex models. For fitting parametric scaling models to observed data, the ESA system we used for scaling analysis uses the non-linear least-squares Levenberg-Marquardt algorithm.

Following previous work [4,7,8], we computed 95% bootstrap confidence intervals for the performance predictions obtained from our scaling models, based on 1000 bootstrap samples per instance set and 1000 automatically fitted variants of each scaling model. For collecting running time data for our TSP solvers, we used the Compute Canada/Westgrid cluster orcinus (DDR), each node of which is equipped with two 3.0 GHz Intel Xeon E5450 quad-core CPUs and 16 GB of RAM, running 64-bit Red Hat Enterprise Linux Server 5.3.

3 Experimental Results

3.1 Treatment of Running Time Data

EAX and LKH cannot prove the optimality of the solutions they find; in our experiments, they therefore need access to the optimal solution qualities of the instances we consider, in order to measure the running time required to reach optimal solutions and to terminate runs once an optimal solution has been found. The optimal solutions to the RUE instance we used have been determined in an earlier study of the scaling of Concorde [8]. However, Concorde did not solve all instances within the allotted time in that study. To make more instances available for EAX and LKH to solve, we ran Concorde on the previously unsolved instances with different seeds and/or on faster machines. In addition, we performed multiple runs of EAX and LKH on those instances that were then still not solved by Concorde. For some of these instances, EAX and LKH found the same best solution in every run; we conjecture these solutions to be optimal and we refer to them as *pseudo-optimal*. For our analysis, we use data for both optimal and pseudo-optimal instances, but we note that qualitatively similar conclusions are obtained when excluding instances with pseudo-optimal solutions.

We took special care in dealing with the eight instances for which we did not succeed in establishing pseudo-optimal solutions (two of size 4000 and six of size 4500). As reported by [4], the pairwise performance correlations between EAX, LKH and Concorde are very low, and the instances for which we were unable to determine even pseudo-optimal solutions may still be easy for one of the inexact solvers. Thus, they are treated using an optimistic/pessimistic estimation, as done by [4]. More precisely, we treat these instances as easy with smaller-than-median running times in the optimistic estimation, and as timed-out instances in the pessimistic treatment. This gives us intervals for the median running times on those instance sizes for which some instances are lacking even pseudo-optimal solutions ($n = 4000$ and $n = 4500$). We note that these intervals are not confidence intervals, but bounds on the median running times, as they must contain the true median running times.

3.2 Scaling of EAX and LKH with Default Parameters

We first repeat the scaling analyses for EAX and LKH of [4,16] with new sets of running time data collected using our machines. This ensures that the comparisons described below are not affected by differences between the machines used. The results we thus obtained are qualitatively similar to those reported in [4,16]. More precisely, we found that the scaling of EAX is reasonably well described by a root-exponential model, while that of LKH is bounded from below and above by a polynomial and a root-exponential model, respectively. We report detailed results for EAX, labelled EAX (default), including the best fitted models and the bootstrap confidence intervals for the model parameters, in Tables 1 and 2, respectively. Analogous results for LKH, labelled LKH (default), can be found in Tables 6 and 7. Comparing our models to those reported in [16], our new results lead to a larger value of b in the root-exponential model for EAX, while large overlaps are seen in the bootstrap confidence intervals for b in the root-exponential and polynomial models of LKH, respectively.

3.3 Impact of Automated Configuration on Scaling of EAX

Next, we automatically configured the two parameters exposed by EAX: N_{pop} (fNumOfPop in the source code), the population size, and N_{ch} (fNumOfKids in the source), the number of offsprings generated per recombination attempt, which also affects the way EAX switches between different search strategies. The default values for these parameters are $N_{pop} = 100$ and $N_{ch} = 30$. In our experiments, we enforced a cut-off of one CPU day for each SMAC run. To ensure that SMAC could perform at least 1000 runs of EAX, we further enforced a cut-off time of 86 s for each EAX run. We note that even though EAX has only two parameters to configure, which makes the use of SMAC seemingly excessive, we still chose to use SMAC, because we saw no harm in doing so and because this allowed us to use the same configuration protocol as for LKH, whose configuration space is much larger. After configuration on a set of RUE instances with $n = 1500$, SMAC determined a parameter setting with N_{pop} (167 *vs* 100) and smaller N_{ch} (20 *vs* 30).

Table 1. Most accurate scaling models (according to RMSE on challenge data) for the median running times required by different variants of EAX for finding optimal solutions to RUE instances and corresponding RMSE values (in CPU sec). EAX (configured) uses a parameter configuration determined by SMAC, and EAX (configured + var pop) additionally determines N_{pop} as a linear function of instance size. All models were fitted using performance data obtained on RUE instances of size 500...1500 and challenged by performance data for instances of size 2000...4500.

Solver	Model		RMSE (support)	RMSE (challenge)
EAX (default)	RootExp.	$0.086254 \times 1.1439^{\sqrt{n}}$	0.12518	[73.961, 116.47]
EAX (configured)	RootExp.	$0.19910 \times 1.1193^{\sqrt{n}}$	0.22625	[22.278, 32.899]
EAX (configured + var pop)	Poly.	$1.6194 \times 10^{-8} \times n^{2.8364}$	0.027288	[37.049, 44.847]

Table 2. 95% bootstrap confidence intervals for the parameters of the scaling models from Table 1.

Solver	Model	Confidence interval for a	Confidence interval for b
EAX (default)	RootExp.	$[0.08287, 0.08991]$	$[1.1424, 1.1452]$
EAX (configured)	RootExp.	$[0.19024, 0.20586]$	$[1.1182, 1.1208]$
EAX (configured + var pop)	Poly.	$[1.3424 \times 10^{-8}, 1.9355 \times 10^{-8}]$	$[2.8110, 2.8623]$

We then investigated the scaling behaviour of EAX with these optimised parameters. The most accurate scaling model, according to the root mean squared error (RMSE) on challenge data (*i.e.*, instance sets not used for model fitting), is shown in Table 1, where this version of EAX is labelled EAX (configured). Similar to results for the default version of EAX, a root-exponential model characterises the scaling most accurately, while the best exponential and polynomial models can be rejected with 95% confidence. We cross-checked the predictions of the root-exponential model for the default version of EAX against the observed running times for the optimised version of EAX. Our results, illustrated in Fig. 1, clearly show that even though the optimised version loses some performance on small instances, the performance on larger instances is significantly improved. Thus, the original model over-estimates the running times for the optimised version of EAX. In addition, from the confidence intervals of the model parameters, as shown in Table 2, there is evidence that algorithm configuration significantly improves the scaling of EAX, since it reduces the value of b in the 2-parameter root-exponential models from 1.144 to 1.119, with non-overlapping confidence intervals ($[1.1424, 1.1451]$ vs. $[1.1182, 1.1208]$).

In preliminary experiments, we noted that the performance for larger instances could be improved by using larger population sizes, N_{pop}. Furthermore, the README file distributed with the source code of EAX recommends to use $N_{pop} = 300$ for instances with $n > 10\,000$ cities. We therefore considered the possibility of increasing N_{pop} with n and performed an experiment to quantify the impact of varying the population size as a simple linear function of n, *i.e.*,

$$N_{pop}(n) := \alpha \cdot n.$$

To obtain an estimate for α, we divided the optimised value of N_{pop} by the instance size $n = 1500$ at which this setting was obtained, resulting in $\alpha = 0.111$. We then analysed the empirical scaling behaviour of EAX with $N_{pop}(n)$ detemined in this manner and the optimised value of $N_{ch} = 20$ from our previous experiment and refer to this setting as EAX (configured + var pop). The most accurate scaling model is presented in Table 1, with bootstrap confidence intervals for the model parameters shown in Table 2. To our surprise, the best accuracy is now achieved by a polynomial model, while the best exponential and root-exponential models are rejected with 95% confidence, as illustrated in Fig. 2. Furthermore, from Fig. 1, it is obvious that this also stands in contrast with the scaling behaviour of EAX (default).

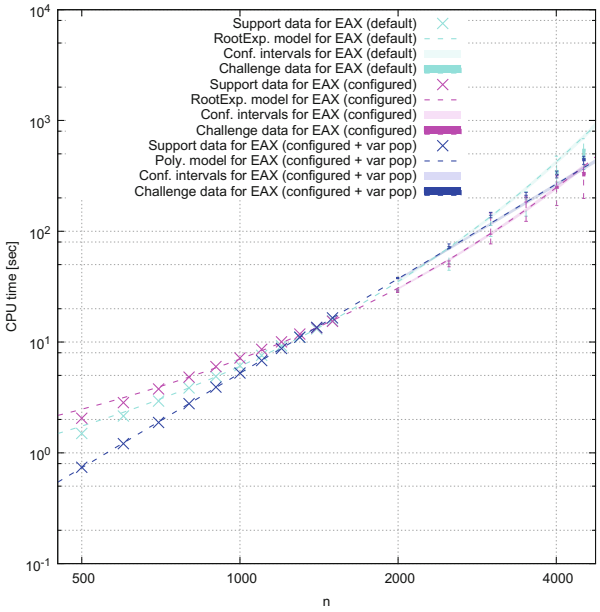

Fig. 1. Most accurate scaling models for the median running times required by three variants of EAX for finding optimal solutions to RUE instances and confidence intervals for model predictions, along with observed performance data; see Tables 1 and 2 for details.

As a result of its improved scaling behaviour, the optimised version of EAX significantly improves the previous state of the art in inexact TSP solving, as represented EAX (default); this can also be seen from the respective observed and predicted running times in Table 3.

We also compared this optimised version of EAX with EAX (default) on solving a set of TSPLIB instances with n between 500 and 4500, based on median running times determined from 10 independent runs per instance. Our results indicate that EAX (configured + var pop) performs better in some cases, but worse in others, and overall does not achieve significantly improved performance. We believe that this is primarily as most TSPLIB instances are actually easier to solve than RUE instances of a similar size, and smaller population size values are sufficient to solve these instances. We note that for the two TSPLIB instances that are harder than similarly-sized RUE instances, EAX (configured + var pop) does perform significantly better than EAX (default). Two other TSPLIB instances were not solved by EAX with any of the two parameter settings.

In addition, we compared LKH with EAX (configured + var pop). As illustrated in Fig. 3, the median running times of EAX (configured + var pop) for $n = 4500$ show substantially less variability than those of LKH (analogous observations hold for other n); furthermore, although EAX performs better than LKH on aggregate, there are many instances on which the converse is true.

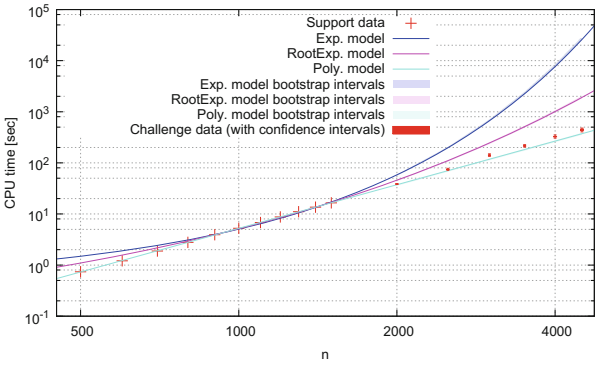

Fig. 2. Scaling models for the median running times required by EAX (configured + var pop) for finding optimal solutions to RUE instances and confidence intervals for predictions obtained from these models, along with observed median running times. All models were fitted using performance data obtained on RUE instances of size 500...1500 and challenged by performance data for instances of size 2000...4500.

Table 3. Improvement of the state of the art in inexact TSP solving, as demonstrated by the bootstrap confidence intervals for observed (for $n = 4500$) and predicted (for $n = 6000, 10000$) median running times for the default and optimised (configured + var pop) versions of EAX.

n	Median running time for EAX			75th percentile of running time for EAX		
	Default	Configured + var pop	Speedup	Default	Configured + var pop	Speedup
4500	[364.1, 685.6]	[404.8, 478.5]	≈1.2×	[685.6, 2350.3]	[482.8, 573.8]	≈ 2.9×
6000	[2711, 3026]	[809, 880]	≈3.4×	[5052, 9908]	[873, 1090]	≈7.6×
10000	[54511, 64275]	[3401, 3800]	≈16.5×	[132120, 364836]	[3613, 4826]	≈58.9×

We performed an analogous performance comparison for the previously mentioned set of TSPLIB instances. Both EAX and LKH fail to solve two of the 28 instances, and LKH fails to solve one instance solved by EAX. As seen in Fig. 3, once again, EAX (configured + var pop) performs better on aggregate, but LKH is considerably faster in solving several of these instances and therefore still contributes substantially to the state of the art in inexact TSP solving.

Analysing the performance results for EAX (configured + var pop) in more detail, we found that by increasing the population size for large instances, the success probability of each restart segment (*i.e.*, the part of a run between two restarts, between initialisation and the first restart and between the last restart and termination) is significantly increased. Figure 4, showing the distributions of the success probabilities of restart segments from all runs of EAX on RUE instances with $n = 1500$ and 4500, clearly illustrates this finding. Increasing population size, however, also increases the running time of the restart segments. The two effects can be clearly seen from medians of success probabilities and

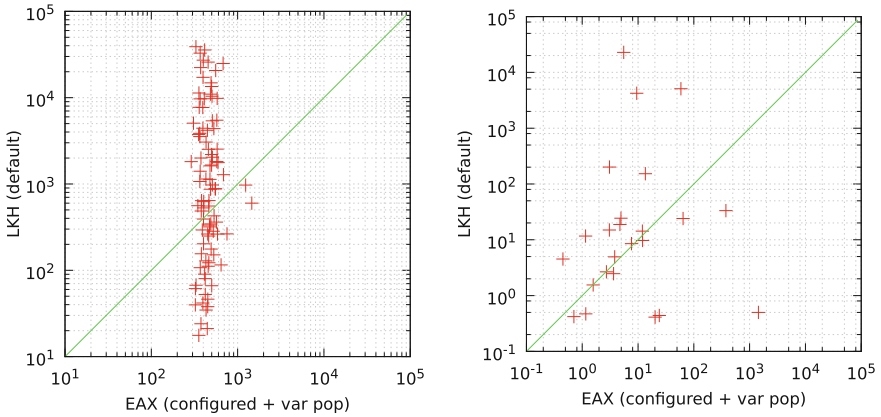

Fig. 3. Running time required by EAX (configured + var pop) *vs* the default version of LKH for solving RUE instances of size $n = 4500$ (left) and a set of TSPLIB instances (right); all running times are reported in CPU seconds. For RUE instances, the median running time for EAX and LKH lies within $[430.8, 454.8]$ and $[870.3, 1131.1]$, respectively. For TSPLIB instances, the median running time for EAX and LKH is 6.6 and 13.0, respectively.

Table 4. Median success probability and running time per restart segment of EAX (default) and EAX (configured + var pop) on sets of RUE instances.

n	Success probability per restart segment		Running time per restart segment	
	Default	Configured + var pop	Default	Configured + var pop
500	1.00	0.77	1.524	0.930
1500	0.77	0.91	16.265	17.991
2500	0.53	0.91	37.986	77.950
3500	0.30	0.91	70.406	218.735
4500	0.16	1.0	119.231	450.940

times to restart shown in Table 4. Hence, there is a tradeoff between the two effects. To achieve better scaling for EAX, it is critical to set the population size at the right point balancing the two effects.

To summarise, our experiments indicate that adapting the population size, N_{pop}, with instance size can significantly improve the scaling behaviour of EAX on RUE instances. After optimising the adaptation mechanism using automated configuration of α (together with N_{ch}), the scaling of EAX is best captured by a polynomial model, and the resulting version of EAX represents a significant improvement in the state of the art for inexact TSP solving. In particular, we observed an $\approx$1.13$\times$ improvement in the median running time for EAX and even more substantial improvements for higher percentiles when solving instances

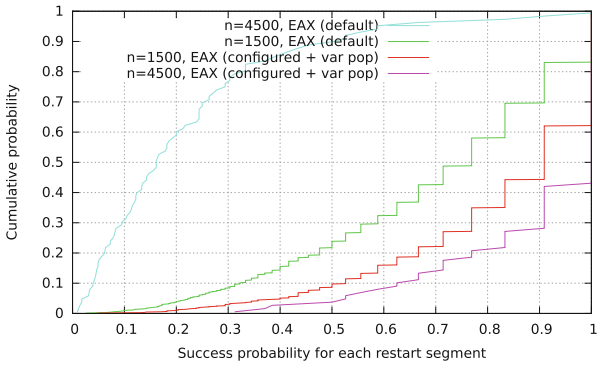

Fig. 4. Distributions of the success probability of each restart segment within EAX solving RUE instances with $n = 1500, 4500$. (For details, see text.)

of size $n = 4500$. Based on our scaling analysis, we expect the performance advantage of EAX (configured + var pop) over EAX (default) to grow further with instance size.

3.4 Impact of Automated Configuration on Scaling of LKH

We also attempted to configure LKH for improved scaling of running time. With over 40 parameters, LKH is arguably much more configurable than EAX. Out of these parameters, some require additional information, such as initial tours or sub-division of a given TSP instance, which are not available in our case. Thus, we selected the 21 parameters (12 numerical and 9 categorical) listed in Table 5, which we could configure without additional information or code modification. We used the same default values as specified in the user guide, except for PATCHING_A and PATCHING_C, as mentioned in Sect. 2.2. The ranges or sets of settings for all parameters were determined based on the user guide; when in doubt, we used large ranges to create a large configuration space for SMAC to explore. We enforced a cut-off of 2 CPU days for each SMAC run. To ensure that SMAC could perform at least 1000 runs of LKH, we further enforced a cut-off time of 172 s for each LKH run.

We first configured LKH analogously to EAX, that is, we performed 25 SMAC runs using a set of 100 RUE instances of size $n = 1500$ randomly selected from the full set, and selected from the 25 configurations the one with the best performance across the 100 training instances. The resulting configuration achieved improved performance only for instances up to size 2000, but did not scale to larger instance sizes, suggesting overfitting on smaller instance sizes.

Next, we attempted to improve the scaling performance by following a protocol proposed by Styles *et al.* [23]. More precisely, we performed 25 SMAC runs to configure LKH using a set of 100 instances with instance size $n = 1000$ and selected the best configuration based on performance on a set of 50 validation instances selected uniformly at random from the the set of RUE instances of

Table 5. List of numerical (N) and categorical (C) parameters for LKH considered in our configuration experiments.

Parameter name	Type	Domain
ASCENT_CANDIDATES	N	$[10, 500]$
BACKBONE_TRIALS	N	$[0, 5]$
BACKTRACKING	C	{YES, NO}
CANDIDATE_SET_TYPE	C	{ALPHA, DELAUNAY, NEAREST-NEIGHBOR, QUADRANT}
EXTRA_CANDIDATES	N	$[0, 20]$
EXTRA_CANDIDATE_SET_TYPE	C	{NEAREST-NEIGHBOR, QUADRANT}
GAIN23	C	{YES, NO}
GAIN_CRITERION	C	{YES, NO}
INITIAL_STEP_SIZE	N	$[1, 5]$
INITIAL_TOUR_ALGORITHM	C	{BORUVKA, GREEDY, MOORE, NEAREST-NEIGHBOR, QUICK-BORUVKA, SIERPINSKI, WALK}
KICK_TYPE	N	$\{0\} \cup [4, 20]$
KICKS	N	$[0, 5]$
MAX_CANDIDATES	N	$[3, 20]$
MOVE_TYPE	N	$[2, 20]$
PATCHING_A	N	$[1, 5]$
PATCHING_C	N	$[1, 5]$
POPULATION_SIZE	N	$[0, 1000]$
RESTRICTED_SEARCH	C	{YES, NO}
SUBGRADIENT	C	{YES, NO}
SUBSEQUENT_MOVE_TYPE	N	$\{0\} \cup [2, 20]$
SUBSEQUENT_PATCHING	C	{YES, NO}

size $n = 1500$. We then analysed the scaling of LKH with the configuration such obtained and found a root-exponential model to fit best, with a value of b very similar to that obtained for the default configuration (1.2077 *vs* 1.2067). Based on these scaling models and observed running times, we determined that this configuration performs better than the default configuration of LKH up to $n = 3500$, but not beyond. This indicates that using the modified configuration protocol reduces, but does not completely eliminate overfitting on smaller instance sizes.

After inspecting the parameter settings obtained from the two configuration experiments described so far in more detail, we concluded that they help LKH to more carefully check possible local search steps in a way that is effective only for smaller TSP instances. Based on this observation, we fixed KICKS, MOVE_TYPE, POPULATION_SIZE and RESTRICTED_SEARCH to their default values and performed another round of configuration following the protocol by Styles *et al.*. Results from the analysis of the scaling behaviour of

Table 6. Most accurate (bounding) scaling models for the median running times required by variants of LKH for finding optimal solutions to RUE instances and corresponding RMSE values (in CPU sec). LKH (scaling configuration) uses parameter settings obtained using SMAC and the protocol by Styles *et al.*; LKH (scaling configuration w/fewer para) uses a configuration obtained analogously using a reduced parameter space (for details, see text). All models were fitted using performance data obtained on RUE instances of size 500...1500 and challenged by performance data for instances of size 2000...4500.

Solver	Model		RMSE (support)	RMSE (challenge)
LKH (default)	RootExp.	$0.010186 \times 1.2067^{\sqrt{n}}$	0.34977	[859.21, 968.83]
	Poly.	$8.2328 \times 10^{-10} \times n^{3.2255}$	0.30801	[190.79, 293.36]
LKH (scaling configuration)	RootExp.	$0.0066414 \times 1.2077^{\sqrt{n}}$	0.13341	[533.13, 1165.9]
LKH (scaling configuration w/fewer para)	RootExp.	$0.0048097 \times 1.2010^{\sqrt{n}}$	0.15081	[1394.8, 2090.6]

Table 7. 95 % bootstrap confidence intervals for the parameters of the scaling models from Table 6.

Solver	Model	Confidence interval for a	Confidence interval for b
LKH (default)	RootExp.	$[0.0047391, 0.01955]$	$[1.1836, 1.2333]$
	Poly.	$[4.3312 \times 10^{-11}, 1.0215 \times 10^{-8}]$	$[2.8684, 3.6341]$
LKH (scaling configuration)	RootExp.	$[0.0040961, 0.010936]$	$[1.1908, 1.2245]$
LKH (scaling configuration w/fewer para)	RootExp.	$[0.0027107, 0.008814]$	$[1.1802, 1.2210]$

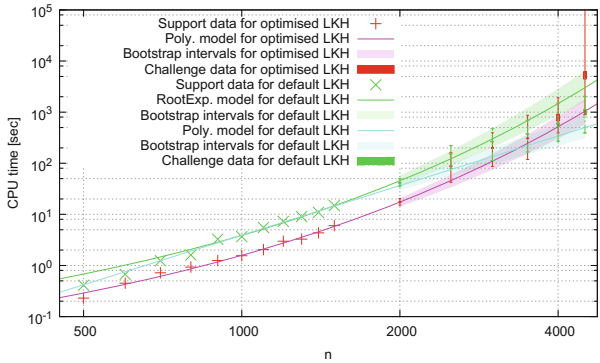

Fig. 5. Scaling models for default LKH and optimised LKH (scaling configuration w/fewer para) from Table 6, along with observed performance data and bootstrap confidence intervals for predicted performance obtained from the scaling models.

the configuration of LKH thus obtained are shown in Tables 6 and 7. We note that the confidence interval for b in the root-exponential model largely overlaps with that for LKH with default parameters ($[1.1802, 1.2210]$ *vs* $[1.1836, 1.2333]$). Again, the root-exponential model gives the best fit (according to RMSE on challenge data), and fits the running times well up to $n = 4000$, as seen in Fig. 5. The value of $b = 1.2010$ in this model is very similar to that for the default version

of LKH ($b = 1.2077$), with a large overlap in the respective confidence intervals ($[1.1802, 1.2210]$ *vs* $[1.1908, 1.2245]$). Comparing the running times and scaling models between these two configurations of LKH, as illustrated in Figure 5, we noticed that the new configuration decreases the running times for $n \leq 4000$, but performs worse for $n = 4500$. In other words, the Styles *et al.* protocol applied to our reduced parameter space seems to overfit less, but still suffers from some overfitting to the instance sizes used for configuration.

4 Discussion

In this work, we investigated the impact of parameter settings and automated configuration on the scaling of inexact TSP algorithms. For EAX, algorithm configuration helps improve the scaling, which can be further improved by adapting the population size with instance size. In particular, we achieved an improvement in the median running time for EAX on RUE instances of size $n = 4500$ of a factor of $\approx$1.13 and of a factor of $\approx$1.87 for the 75th percentile of the distribution of running times over sets of RUE instances; based on our scaling models, we expect the improvement to be even more significant for larger instances.

Surprisingly, when adapting the population size with instance size, we obtain polynomial scaling of the median running time with instance size, compared to root-exponential scaling for the default configuration of EAX. To the best of our knowledge, this is the first time, polynomial scaling of the empirical median running time for an inexact TSP solver has been reported for a widely used set of challenging 2D Euclidean TSP instances.

Overall, our work complements earlier work on the scaling of state-of-the-art TSP solvers [4,8,9] and indicates potential for improvements in scaling behaviour through automated algorithm configuration and through setting certain parameters in dependence of features of the problem instance to be solved.

We see significant potential in developing automated configuration procedures for better scaling behaviour. Such procedures can make algorithm configuration more applicable to real-world situations, as problem instances from the target distribution may take a long time to solve. This poses a substantial challenge to existing algorithm configuration techniques, which require many runs of the target algorithms with different parameter settings. There is some previous work on configuration protocols addressing this challenge [15,22,23], but based on the findings for LKH reported in our study here, we believe that there is the need (and, indeed much room) for further improvements. In particular, we see the tight integration of empirical scaling analysis into the configuration process as a promising avenue for future research in this area.

Acknowledgements. HH and ZM acknowledge support through an NSERC Discovery Grant. TS acknowledges support from the Belgian F.R.S.-FNRS, of which he is a senior research associate. This work received support from Compute Canada/Westgrid and from the COMEX project within the Interuniversity Attraction Poles Programme of the Belgian Science Policy Office.

References

1. Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 142–157. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04244-7_14
2. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The traveling salesman problem, concorde TSP solver (2012). http://www.tsp.gatech.edu/concorde
3. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: F-race and iterated F-race: an overview. In: Bartz-Beielstein, T., Chiarandini, M., Paquete, L., Preuss, M. (eds.) Experimental Methods for the Analysis of Optimization Algorithms, pp. 311–336. Springer, Heidelberg (2010)
4. Dubois-Lacoste, J., Hoos, H.H., Stützle, T.: On the empirical scaling behaviour of state-of-the-art local search algorithms for the Euclidean TSP. In: Proceedings of GECCO 2015, pp. 377–384. ACM Press (2015)
5. Helsgaun, K.: An effective implementation of the Lin-Kernighan traveling salesman heuristic. Eur. J. Oper. Res. **126**(1), 106–130 (2000)
6. Helsgaun, K.: General k-opt submoves for the Lin-Kernighan TSP heuristic. Math. Program. Comput. **1**(2–3), 119–163 (2009)
7. Hoos, H.H.: A bootstrap approach to analysing the scaling of empirical run-time data with problem size. Technical report, TR-2009-16, Department of Computer Science, University of British Columbia (2009)
8. Hoos, H.H., Stützle, T.: On the empirical scaling of run-time for finding optimal solutions to the travelling salesman problem. Eur. J. Oper. Res. **238**(1), 87–94 (2014)
9. Hoos, H.H., Stützle, T.: On the empirical time complexity of finding optimal solutions vs proving optimality for Euclidean TSP instances. Optim. Lett. **9**, 1247–1257 (2015)
10. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C.A.C. (ed.) LION 2011. LNCS, vol. 6683, pp. 507–523. Springer, Heidelberg (2011). doi:10.1007/978-3-642-25566-3_40
11. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. J. Artif. Intell. Res. **36**(1), 267–306 (2009)
12. Johnson, D.S., McGeoch, L.A.: The traveling salesman problem: a case study in local optimization. In: Aarts, E.H.L., Lenstra, J.K. (eds.) Local Search in Combinatorial Optimization, pp. 215–310. Wiley, Chichester (1997)
13. Johnson, D.S., McGeoch, L.A.: Experimental analysis of heuristics for the STSP. In: Gutin, G., Punnen, A. (eds.) The Traveling Salesman Problem and Its Variations, pp. 369–443. Kluwer Academic Publishers, New York (2002)
14. Kotthoff, L., Kerschke, P., Hoos, H., Trautmann, H.: Improving the state of the art in inexact TSP solving using per-instance algorithm selection. In: Dhaenens, C., Jourdan, L., Marmion, M.-E. (eds.) LION 2015. LNCS, vol. 8994, pp. 202–217. Springer, Heidelberg (2015). doi:10.1007/978-3-319-19084-6_18
15. Mascia, F., Birattari, M., Stützle, T.: Tuning algorithms for tackling large instances: an experimental protocol. In: Nicosia, G., Pardalos, P. (eds.) LION 2013. LNCS, vol. 7997, pp. 410–422. Springer, Heidelberg (2013). doi:10.1007/978-3-642-44973-4_44
16. Mu, Z.: Analysing the empirical time complexity of high-performance algorithms for SAT and TSP. Master's thesis, University of British Columbia, Vancouver, Canada (2015)

17. Mu, Z., Hoos, H.H.: Empirical scaling analyser: an automated system for empirical analysis of performance scaling. In: GECCO 2015, Companion, pp. 771–772 (2015)
18. Mu, Z., Hoos, H.H.: On the empirical time complexity of random 3-SAT at the phase transition. In: Proceedings of IJCAI 2015, pp. 367–373 (2015)
19. Nagata, Y.: Edge assembly crossover: a high-power genetic algorithm for the traveling salesman problem. In: Proceedings of ICGA 1997, pp. 450–457 (1997)
20. 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)
21. Stützle, T., Hoos, H.H.: 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, pp. 589–611. Kluwer Academic Publishers, New York (2001)
22. Styles, J., Hoos, H.: Ordered racing protocols for automatically configuring algorithms for scaling performance. In: Proceedings of GECCO 2013, pp. 551–558. ACM (2013)
23. Styles, J., Hoos, H.H., Müller, M.: Automatically configuring algorithms for scaling performance. In: Hamadi, Y., Schoenaur, M. (eds.) LION 6. LNCS, vol. 7219, pp. 205–219. Springer, Heidelberg (2012). doi:10.1007/978-3-642-34413-8_15