Advanced Statistical Analysis of Empirical Performance Scaling

Yasha Pushak
Department of Computer Science
The University of British Columbia
Vancouver, Canada
ypushak@cs.ubc.ca

Holger H. Hoos
LIACS, Universiteit Leiden
Leiden, The Netherlands
hh@liacs.nl

ABSTRACT

Theoretical running time complexity analysis is a widely adopted method for studying the scaling behaviour of algorithms. However, theoretical analysis remains intractable for many high-performance, heuristic algorithms. Recent advances in statistical methods for empirical running time scaling analysis have shown that many state-of-the-art algorithms can achieve significantly better scaling in practice than expected. However, current techniques have only been successfully applied to study algorithms on randomly generated instance sets, since they require instances that can be grouped into "bins", where each instance in a bin has the same size. In practice, real-world instance sets with this property are rarely available. We introduce a novel method that overcomes this limitation. We apply our method to a broad range of scenarios and demonstrate its effectiveness by revealing new insights into the scaling of several prominent algorithms; *e.g.*, the SAT solver lingeling often appears to achieve *sub-polynomial* scaling on prominent bounded model checking instances, and the training times of scikit-learn's implementation of SVMs scale as a lower-degree polynomial than expected (≈ 1.51 instead of 2).

CCS CONCEPTS

• **General and reference → Performance**; **Empirical studies**; **Estimation**; **Evaluation**;

KEYWORDS

Empirical Running Time Scaling, Empirical Scaling Analysis, Empirical Performance Scaling

ACM Reference Format:
Yasha Pushak and Holger H. Hoos. 2020. Advanced Statistical Analysis of Empirical Performance Scaling. In *Genetic and Evolutionary Computation Conference (GECCO '20), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3377930.3390210

1 INTRODUCTION

The scaling of the performance, notably: running time, of algorithms with input size is of great theoretical and practical interest. In most cases, theoretical analysis is limited to asymptotic results with unknown constants and lower-order terms, which can and often do play an important role when using an algorithm in practice – *e.g.*, for deciding whether and at what cost an algorithm can be used in a given use case, which of several algorithms to use, and even to critically assess the impact of improvements in algorithm design and implementation.

Empirical analysis of performance scaling, using principled statistical methods, provides an interesting alternative. Here, we build on a line of previous work that has already succeeded in showing that many state-of-the-art algorithms obtain better-than-expected performance on prominent $\mathcal{NP}$-complete and $\mathcal{NP}$-hard problems, including propositional satisfiability (SAT) and the travelling salesperson problem (see, *e.g.*, Mu and Hoos [26], Mu et al. [27]). In particular, empirical scaling analysis revealed several prominent, incomplete SAT solvers show sub-exponential and perhaps even polynomial scaling on random 3-SAT phase transition instances [26].

The methodology that enables these and similar results, which has been made broadly available in a tool dubbed *empirical scaling analyzer (ESA)* [30], assumes that the inputs or problem instances that form the basis for the empirical scaling analysis are grouped into "bins", where each instance in a given bin has the same size. However, for many practical, "real-world" applications, instances that are grouped in this way are unavailable (see, *e.g.*, the instance sets we study in Section 3.1). As a result, applications of this type of empirical scaling analysis have been inherently limited to algorithms running on randomly generated instance sets.

In this work, we propose a new and enhanced method for empirical scaling analysis that overcomes this limitation, making it applicable to a broad range of interesting performance prediction and modelling tasks in genetic and evolutionary computation, and beyond. At its core, our methodology can be viewed as a dedicated transfer learning procedure specially designed for algorithm running time scaling analysis and prediction. That is, it uses a novel fitting procedure to train candidate scaling models on a set of running time data used as a training set, and then validates these models' extrapolations on a set of test data with larger instance sizes. We also introduce a novel bootstrap sampling procedure, which allows it to perform a statistical analysis of the consistency of the model predictions with the test data.

Our method tests hypotheses about an algorithm's empirical scaling behaviour. For example, "Is the observed empirical running time scaling of algorithm A consistent with the scaling model class $\mathtt{exp}(x) = a \cdot b^x$?". This is unlike theoretical analysis, which can prove bounds on an algorithm's scaling (*e.g.*, $A \in O(n \log(n))$). Nevertheless, a major advantage of our method is that it returns concrete instantiations of models (including their parameters) which can be used to predict A's scaling. In contrast, theoretical analysis

ACM ISBN 978-1-4503-7128-5/20/07…$15.00
https://doi.org/10.1145/3377930.3390210

abstracts away lower order terms and constant values, and therefore cannot make predictions about an algorithm's running times on unseen instances. This advantage is substantial, since small-scale tests can be used to reason about the relative *and* absolute performance of algorithms on larger problem instances.

To the best of our knowledge, very few statistically principled methods for empirical performance scaling analysis exist. Most prior work combines measuring target algorithm performance in terms of some features (*e.g.*, running time or operation counts) and fitting models to explain performance features as a function of an input feature. For example, Goldsmith et al. [14] measure the number of times blocks of code are executed, and they fit models of the form $a \cdot x + b$ and $a \cdot x^b$ to the performance measurements using linear regression (or linear regression on a log-log scale). Some of the more advanced approaches automatically extract notions of instance size or algorithm performance (see, *e.g.*. Zaparanuks and Hauswirth [34] or Coppa et al. [8]). McGeoch et al. [25] introduce several experimental procedures for fitting and bounding polynomial models to performance data. However, none of these methods leverage the power of statistical methods to quantify the degree to which learned models can be trusted in challenging extrapolation scenarios. One of the more advanced methods [13] partially addresses the problem of model trust by automatically determining model complexity (*i.e.*, the number of model parameters) by analysing the source code of the target algorithm. They argue that more complex algorithms give rise to performance scaling that cannot be described without additional scaling model parameters. However, they do not provide any validation of model extrapolations.

This work represents a significant advance in the state of the art of empirical complexity analysis for two reasons: First, our methodology (see Section 2) can be applied to a substantially broader range of application scenarios, thereby allowing it to be applied to many important real-world problems. And second, the most obvious solutions to the problem do not work well in practice. In particular, the custom optimization procedure that we develop in Section 2.1 proved to be critically important to obtain robust, high-quality scaling model parameters. We experimented with over a dozen different pre-existing, state-of-the-art optimization procedures (*e.g.*, IP- or LP-based methods), some of which were chosen in consultation with numerical optimization experts, yet none of which yielded good results. A second contribution, is comprised of several case studies that demonstrate the utility of our method through the discovery of several surprising results (see Section 3). For example: lingeling often appears to obtain *sub-polynomial* scaling on bounded model checking SAT instances (as opposed to the well-known exponential worst-case theoretical bound). And, on the MNIST data set, random forest scaling appears to be slightly sub-linear (indicating that it may need less than one full pass over the data set) and support vector machine scaling is consistent with a polynomial model of degree ≈ 1.51 (much less than expected from theory [20, 21]). We also provide a comparison to previous results by Mu and Hoos [26] (see Section 4). We close with concluding remarks and a discussion of future work (see Section 5). To facilitate broad use of our methodology for empirical scaling analysis, we make it available as version 2 of ESA. [1]

[1]www.cs.ubc.ca/labs/beta/Projects/ESA

2 METHOD

The original method by Mu and Hoos [26] starts from statistics (*e.g.*, median running times) calculated for each instance size "bin", and fits scaling models to these statistics with least squares regression. It uses stratified bootstrap sampling and an extrapolation test to assess the quality of each model. However, these statistics (and therefore the model fitting procedure) and the stratified bootstrap sample all require sets of equally sized instances.

Our method contains three major improvements that overcome these limitations and improve the quality of the results. First, we fit directly to the full training set, using a custom quantile regression procedure (see Section 2.1), which allows us to obtain more robust model fits (see Section 4). Second, we use a new method for calculating observed statistics (see Section 2.2) that does not require bins. And third, we introduce a novel bootstrap sampling procedure (see Section 2.3 that, like stratification, provides a variance-reduction technique for bootstrapping of regression problems without bins.

At a high-level, our method proceeds as follows:

(1) We run an algorithm on a set of problem instances with varying instance sizes in a random order while measuring performance in CPU seconds.[2]
(2) We partition the running time data set by instance size into a training and test set;
(3) We use the training set to fit scaling models that we hypothesize may describe the data well;
(4) We use bootstrap sampling to obtain confidence intervals; and finally,
(5) We evaluate the consistency of the models' predictions with observations on the test set.

2.1 Fitting Models

Our method can be performed on different statistics of the running time distribution: In particular, the mean, median or arbitrary quantiles. One advantage to using median or quantile scaling, is that these statistics are more robust than means. Perhaps most importantly, they are un-affected by a modest number of censored runs of the algorithm (*i.e.*, runs which could not be completed within a pre-defined running time cutoff, T).

Fitting scaling models to the mean of the running time distribution is straightforward and can be easily done using standard least squares regression (*i.e.*, using the L2-norm loss function). To fit medians we minimize the residuals using the L1-norm loss [17] and we generalize this by weighting each "arm" of the loss function with weights $(1 - q)$ and q, where $0 < q < 1$, to obtain quantile regression for quantile q [22]. This yields the objective function

$$\sum_i |\rho_q(y[i] - \hat{y}[i])|, \tag{1}$$

where the function ρ_q is defined as

$$\rho_q(r) = \begin{cases} r \cdot (1 - q) & \text{if } r < 0 \\ r \cdot q & \text{otherwise;} \end{cases} \tag{2}$$

[2]In some cases we have observed changes in running times up to a factor of 2 for extended periods of time due to environmental noise (*e.g.*, clock speed changes designed to improve performance without over-heating the machine). Therefore, identically and independently distributing these effects is critical to obtain robust results.

where $y[i]$ is the observed running time for instance i and where $\hat{y}[i]$ is the running time statistic predicted by the scaling model for training instance i with size $x[i]$, *e.g.*,

$$\hat{y}[i] = \texttt{poly}(x[i]) = a \cdot x[i]^b. \tag{3}$$

In theory, any high quality optimization procedure can be used to fit any scaling model using the objective function in Equation 1. However, in practice, we were unable to find any out-of-the-box method that could reliably obtain high-quality results for all of the scaling models we studied here. One method that we found particularly effective for median regression with *linear* models was iteratively reweighted least squares (IRLS) [17]. IRLS calculates a weight for each instance i in the next iteration as $\frac{1}{|r[i]|}$, where $r[i] = y[i] - \hat{y}[i]$ is the residual for instance i using the model fit in the previous iteration. Hence, the objective function for median regression becomes

$$\sum_i \frac{\left(y[i] - \hat{y}_j[i]\right)^2}{\left|y[i] - \hat{y}_{j-1}[i]\right|}, \tag{4}$$

where $\hat{y}_j[i]$ is the running time statistic predicted by the model fitted in the j^{th} iteration of IRLS for instance size $x[i]$. The sequence of regression problems terminates when the loss stops improving. We heuristically warm-start the procedure by using least squares regression to fit to the observed statistics (see Section 2.2 for how these are obtained) in the first iteration, which yields the initial values for $\hat{y}_0[i]$.

By noting that $\lim_{j\to\infty} \left|r_{j-1}[i] - r_j[i]\right| = 0$ (provided that the sequence does not diverge) and by switching $\rho_q(\cdot)$ to $\rho_{1-q}(\cdot)$ we can move ρ to the denominator, yielding

$$\sum_i \frac{\left(y[i] - \hat{y}_j[i]\right)^2}{\left|\rho_{1-q}(y[i] - \hat{y}_{j-1}[i])\right|}, \tag{5}$$

which means that ρ can be absorbed into the weights. This allows us to formulate the original non-smooth, quantile regression problem as a sequence of smooth, least squares regression problems, which can be easily solved for any linear model using standard methods.

However, while Equation 5 works well for linear models, we also want to study non-linear models, *e.g.*, $\texttt{exp}(x) = a \cdot b^x$ or $\texttt{poly}(x) = a \cdot x^b$. We transform these into linear problems by taking the log of the running times and the log of the models. For example,

$$\begin{aligned} \log(\texttt{poly}(x)) &= \log(a \cdot x^b) \\ &= \log(b) \cdot x + \log(a) \\ &= a' \cdot x + b' \\ &= \texttt{lin}'(x), \end{aligned} \tag{6}$$

where we have introduced the transformations $a' = \log(b)$ and $b' = \log(a)$. Then, by introducing $y'[i] = \log(y[i])$, we can fit the model $\texttt{lin}'(x) = a' \cdot x + b'$ to y' using the objective function in Equation 5. Finally, we use the inverse transformations to retrieve approximations for the optimal values of a and b from the fitted values for a' and b'.

Unfortunately, these transformations distort the residuals, which effectively increases the weight for smaller running times. To compensate, we use a similar heuristic to Eggensperger et al. [11] and further weight instance i using the predicted running time for instances with size $x[i]$ from the previous iteration of IRLS. The objective function then becomes

$$\sum_i \frac{\left(\log(y[i]) - \log(\hat{y}_j[i])\right)^2 \cdot \hat{y}_{j-1}[i]}{\left|\rho_{1-q}\left(\log(y[i]) - \log(\hat{y}_{j-1}[i])\right)\right|}. \tag{7}$$

For example, putting everything together with the `poly` model yields the sequence of optimization problems: for $j = 1, 2, \ldots$ minimize over a'_j, b'_j the weighted L2 loss function

$$\sum_i \left(y'[i] - a' \cdot x[i] - b'\right)^2 \cdot W_{(a'_{j-1}, b'_{j-1})}[i], \tag{8}$$

where

$$W_{(a'_{j-1}, b'_{j-1})}[i] = \frac{(\exp(b'_{j-1}) \cdot x^{\exp(a'_{j-1})})}{\left|\rho_{1-q}(y'[i] - a'_{j-1} \cdot x[i] - b'_{j-1})\right|} \tag{9}$$

is a constant weight that can be calculated using the fitted values for a'_{j-1} and b'_{j-1} from the previous iteration. We terminate this sequence of problems when the original L1-loss (without taking the log) stops improving.

This sequence of problems can be easily solved using any weighted, linear-least squares regression procedure for any of the models studied in this work. However, the log transformation and heuristic correction of the running times is only required for some of the scaling model classes (*i.e.*, those that are already linear do not require this step). Similar transformations and heuristic corrections can also be performed to extend this method to an even broader range of scaling model classes than those studied here.

2.2 Calculating Observed Statistics

We take inspiration from Yu and Jones [33] and fit a linear model $\texttt{lin}(x) = a \cdot x + b$ (see Section 2.1) to the running times within a local, sliding window. In the first iteration, to obtain a_0 and b_0, we fit the model using un-weighted linear least squares regression. To reduce biases introduced by any curvature in the running time scaling, we weight the training examples using a normal curve centered at the middle of the window. We record the running time predicted by the linear model at the centre of the window as the observed running time statistic. Linear interpolation is used to estimate performance between observations.

2.3 Bootstrap Sampling

We take bootstrap samples of the data sets and repeat the steps outlined above on each of these. We then use the bootstrapped distributions of the fitted model predictions and observed statistics thus obtained to obtain 95 percentile confidence intervals. To reduce variance in fitted models, we perform bootstrap sampling in a novel way: We use a sliding window centered around each instance exactly once. The window is modular so that each window contains the same number of instances. Then, for each position of the window, we sample one instance from within the window, which yields a single, balanced bootstrap sample.

2.4 Quality of Model Fits

We use a heuristic decision model that is firmly grounded in statistics to describe the quality of model fits. That is, we say that a model fits the observed test statistics if at least 90% of the bootstrap confidence intervals for the test instance size observations are consistent (overlap) with the bootstrap confidence intervals for the model's predictions. We say that a model over/under-estimates the observed test statistics if the model does not fit the data well, and if at least 90% of the bootstrap confidence intervals for the test instance sizes are below/above or are consistent with the bootstrap confidence intervals for the model's predictions. We note that the first of these statements (whether or not the model fits) can be viewed analogously to hypothesis testing, where we are checking to see if there is enough evidence to reject a hypothesis given the observations; however, the second statement (whether or not the model over/under-estimates the data), does not correspond to hypothesis testing, but should instead be viewed as a researcher's observation regarding what appears to be a systematic bias in the data. While the second is technically a weaker statement, it nevertheless provides valuable insight into the behaviour of the algorithm. For example, if a $\texttt{poly}(x)$ model over-estimates the test data, then this may imply that the algorithm's underlying scaling is sub-polynomial.

3 CASE STUDIES

We applied our method to several new scenarios. In each, we used 500 bootstrap samples with a window size of 101, and we used the first 30% of the running time data as training data. All of our experiments were run on a machine running thirty-two 2.10 GHz Intel Xeon E5-2683 v4 CPUs sharing a 40 960 KB cache and 96 GB RAM. They were running openSUSE Leap 42.1 (x86_64). We used a single core per CPU and limited RAM use to 3 GB in all experiments.

3.1 Bounded Model Checking

A commonly studied benchmark set from the algorithm configuration library (ACLib) [19] is comprised of bounded model checking instances, which are encoded in one of the most prominent combinatorial optimization problem types: SAT. Problem instances are obtained by unrolling loops in hardware circuits to varying depth, in order to verify correctness. We were unable to use the existing instance set, because instances that could not be solved within specified minimum and maximum running time cutoffs were removed to obtain a benchmark set with roughly similar difficulty.[3] Instead, we used aigunroll [5] to unroll the original set of circuits from the 2008 Hardware Model Checking Competition (HWMCC08) [3] to various depths. In particular, we choose 5 circuits from the original set and unrolled them to depths 0, 1, ..., 500 to generate 500 instances for each circuit. Some of the smallest SAT instances were trivial and could be solved by aigunroll; we discarded these.

We then ran lingeling [4], a prominent, state-of-the-art SAT solver, once on each instance and recorded its running times. We measured the size of the SAT instances as the number of variables. Then, we used ESA v2 to fit two models, $\texttt{exp}(x) = a \cdot b^x$ and $\texttt{poly}(x) = a \cdot x^b$, to the median running time of lingeling on each circuit.

[3]Personal communication with the author.

Table 1: Lingeling Scaling Models

Circuit	Best-Fit Model	Fit
139454p5	$4.02 \cdot 10^{-5} \cdot x^{1.00}$	over
csmacdp0neg	$8.20 \cdot 10^{-11} \cdot x^{2.09}$	fits
pdtvisvsar29	$8.67 \cdot 10^{-6} \cdot x^{1.25}$	under
prodconspold4	$1.78 \cdot 10^{-6} \cdot x^{1.33}$	over
texastwoprocp5	$1.14 \cdot 10^{-5} \cdot x^{1.13}$	over
all	$5.15 \cdot 10^{-7} \cdot x^{1.44}$	over

The best-fit models (according to test loss) are shown in rows 1–5 of Table 1. Surprisingly, for all 5 circuits, the exp scaling model was rejected as an over-estimate with 95% confidence. Even more surprisingly, the poly models appeared to over-estimate the test data for three of the five circuits. For the other two, one circuit had test data consistent with and one slightly under-estimated by the poly model, respectively. An example of each is shown in Figures 1a–c (where the dotted black line is the train/test split and the dashed black line is the running time cutoff). Visually, the exp model in Figure 1a appears to provide a good upper bound for the test data. However, we also applied our method to study the 95th quantile of the data, and found that it rejected the exp model with 95% confidence, with the true scaling falling between the two models (data not shown).

These results provide fairly strong evidence that the empirical running time scaling of lingeling on these bounded model checking problems is sub-exponential and perhaps even sub-polynomial. However, in light of our theoretically-motivated expectations, how much confidence should we have that these results will generalize to other, similar circuits? To answer this question, we generated one additional set of bounded model checking instances by unrolling each of the hardware circuits from the HWMCC08 to depths 25, 50, ..., 250. We performed the same analysis on these, and found that, once again, both models continued to over-estimate the test data (see row 6 in Table 1 and Figure 1d).

3.2 Machine Learning

A very popular field of modern AI is machine learning, which often operates on large data sets. The recent success of machine learning in many applications has lead to its widespread adoption for solving problems in all areas of computer science, including genetic and evolutionary computation, see *e.g.*, the use of random forest surrogate models in a gender-based genetic algorithm [1]. We applied ESA v2 to study the training times for three prominent machine learning classification algorithms: random forests (RF) [6], k nearest neighbours (KNN) [12] and support vector machines (SVM) [9]. In this context, a problem instance corresponds to a training set, and the size of an instance is defined as the number of training examples. To obtain a training set, we sub-sampled n training examples from the MNIST [23] data set – one training instance set for each $n \in [2\,500, 2\,519, \ldots, 50\,000]$.

We used scikit-learn version 0.18.1's [31] implementation of the three learning procedures. Since the default configuration of SVMs did not produce good test performance, we used SMAC [18] to perform hyper-parameter optimisation for all three algorithms

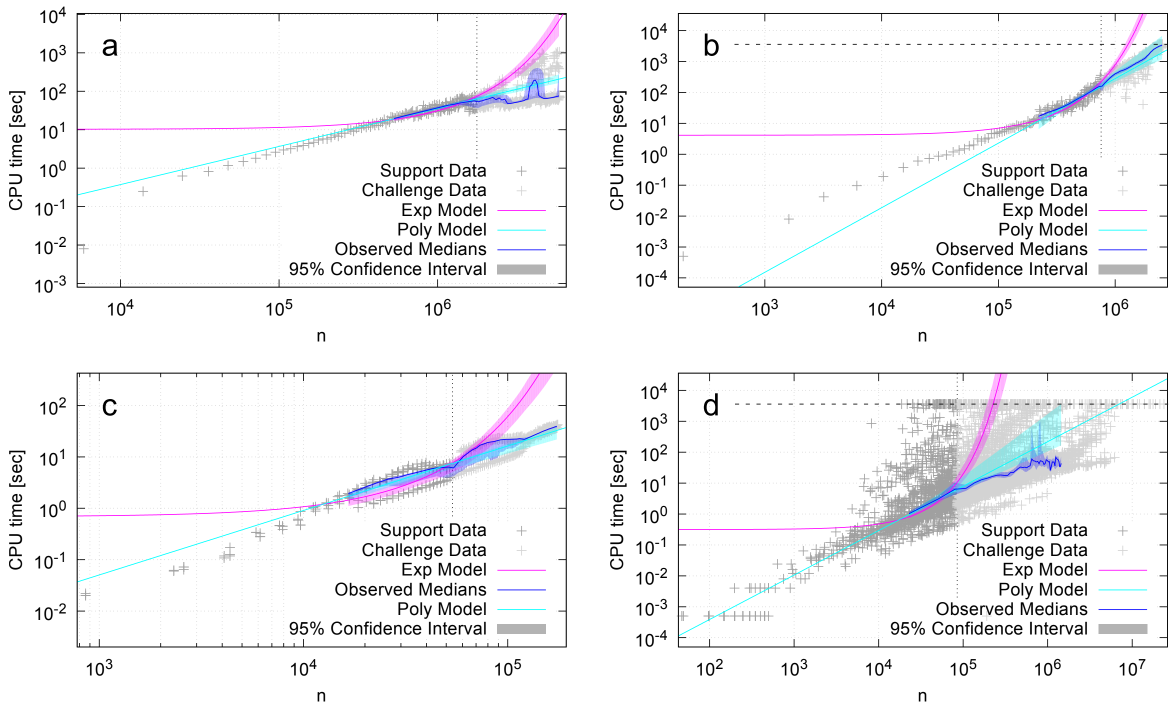

Figure 1: Empirical Scaling Of Lingeling On (a) 139454p5, (b) Csmacdp0neg, (c) Pdtvisvsar29 And (d) All Circuits (For Details, See Text).

by training on instances of size 5 000. In particular, for each machine learning algorithm we performed 25 runs of SMAC, each with budgets of 1 000 target algorithm runs, and we took the hyper-parameter configuration with the best validation performance on a withheld set of training data. The final hyper-parameter configurations had average validation scores of 89% (RF), 94% (KNN) and 95% (SVM) when trained and validated on sub-samples of size 5 000. The best-fit scaling models for the training times of each machine learning model are summarized in Table 2.

3.2.1 *Random Forests.*

We ran ESA v2 with the poly and exp models. Surprisingly, the poly model slightly over-estimated the training times for random forests, *e.g.*, for the largest test instance size the poly model prediction was 1.11 times larger than the observation. The fact that the 95% confidence interval for the degree, b, of the polynomial scaling model was determined as $[1.07, 1.10]$ indicates observed scaling slightly better than linear. We performed the analysis again using a linear model which was fitted as $\texttt{lin}(x) = 3.60 \cdot 10^{-4} \cdot x - 0.28$. The result was consistent with our hypothesis: it also over-estimated the test data, although it did provide a slightly tighter bound, *i.e.* the lin model prediction was 1.03 times larger than the observation for the largest test size. This appears to indicate that random forest training can done with slightly less than one full pass over the training set. One possible alternative explanation is that there are sub-linear start-up costs (see *e.g.*, Section 3.3); however, we observed that several other models with lower-order terms (such as $a \cdot x + b \cdot \log(x)$) over-estimated the test data with 95% confidence, so we did not find support for this hypothesis.

3.2.2 *k-Nearest Neighbours.*

A brute-force implementation of KNN should run in linear time, since it only memorizes the training set [10]; however, the variant we studied is more advanced and builds a k-d tree during training to save computation at test time. Building k-d trees can be done in $O(n \cdot \log(n))$ time [7], so in addition to the exp and poly models, we tried fitting a $\texttt{linlog}(x) = a \cdot x \cdot \log(x) + b$ model. The exp model clearly over-estimated the data, so we do not discuss it further. Surprisingly, both the $\texttt{poly}(x) = 4.71 \cdot 10^{-6} \cdot x^{1.30}$ and the $\texttt{linlog}(x) = 8.93 \cdot 10^{-6} \cdot x \cdot \log(x) - 0.08$ models clearly under-estimated the running times for KNN; *e.g.*, the observed median training time was 1.45 times larger than the poly model prediction for the largest test instance size. KNN had the smallest running times of the three machine learning algorithms (starting at 0.122 CPU seconds), so it is possible that start-up costs or floor-effects affected running times observed for small training instance sizes. However, we were not able to find support for this hypothesis when rerunning our analysis without some of the smaller instance sizes, nor by using different models with additional terms. Even when using only $n \geq 30\,000$, the poly model still under-estimated the data. For $n \geq 35\,000$ the poly model did appear to fit the data; however, there was also so little data left that the exponential model provided a very tight upper bound on the data, hence we could only conclude that there was no longer sufficient data (nor sufficient room for extrapolation) to make powerful

Table 2: ML Training Time Scaling Models

ML Model	Best-Fit Model	Fit
RF	$3.60 \cdot 10^{-4} \cdot x - 0.28$	over
KNN	$4.71 \cdot 10^{-6} \cdot x^{1.30}$	under
SVM	$5.03 \cdot 10^{-5} \cdot x^{1.51}$	fits

statistical statements. To the best of our knowledge, the implementation of KNN we studied does not use dimensionality reduction or any other tricks to reduce the computational cost at test time, leading us to conjecture that there may be a performance bug in the implementation, causing super-polynomial scaling.

3.2.3 Support Vector Machines. As expected, SVM training times were clearly over-estimated by the `exp` model and consistent with the predictions from the `poly` model. However, counter to our expectations, the best-fit `poly` model had degree $b = 1.51$ with a 95% confidence interval of [1.50, 1.52]. There exist many different versions of SVMs, and while some can run faster, common methods are expected to have quadratic or worse running time complexity [20, 21]. Furthermore, the documentation for scikit-learn's SVM implementation explicitly states that the training time is quadratic in the number of training examples.[4] Our analysis indicates that the scaling of SVMs is better than expected, which may be because the implementation we studied exploits structure in the training set.

3.3 Sorting

Sorting large data sets is a common sub-problem that is broadly relevant to all areas of computer science. The scaling of sorting algorithms is also a very interesting choice for empirical running time scaling analysis, as there are many well-known and well-established theoretical results we can use as a baseline. For example, Hoare's QuickSort [16] has worst case scaling in $O(n^2)$, but expected scaling in $O(n \cdot \log(n))$. Furthermore, it is known to have a linear lower-order term [16]. It is also known that the choice of the pivot rule affects QuickSort's performance [2]. To investigate empirical scaling, we implemented a very simple version of Hoare's QuickSort and studied three different pivot rules: a random pivot (R), the median of three random elements (M3) and the ninther (N), *i.e.*, the median of three medians of three. Theoretically, using the median element (of the entire array) is the optimal choice for the pivot; however, this is not done in practice since finding the exact median at each iteration is more expensive than using a random element. Hence, the median of three and ninther pivot rules can trade off between the quality of the pivot and the time required to pick the pivot. As a result, it has been recommended to use the more complicated pivot rules only when sorting large lists [2].

We generated instances with n elements by sampling n integers uniformly from $[0, 100 \cdot n]$. For each $n \in [10\,000, 20\,000, \ldots, 9\,990\,000]$, we generated one instance and ran each configuration of QuickSort once. We considered four candidate scaling models: the `exp` and `poly` models used in Section 3.1, as well as $\texttt{linlog}(x) = a \cdot x \cdot \log(x) + b$ and $\texttt{linlog+lin}(x) = a \cdot x \cdot \log(x) + b \cdot x$, motivated

[4]https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Table 3: Quicksort Scaling Models

Quicksort	Best-Fit Model	Fit
(R)	$2.02 \cdot 10^{-6} \cdot x \cdot \log(x) + 7.91 \cdot 10^{-6} \cdot x$	fits
(M3)	$7.76 \cdot 10^{-7} \cdot x \cdot \log(x) + 2.56 \cdot 10^{-6} \cdot x$	fits
(N)	$9.84 \cdot 10^{-7} \cdot x \cdot \log(x) + 9.90 \cdot 10^{-6} \cdot x$	fits

Table 4: Extended Quicksort Extrapolation Test

Quicksort	95% Interval (CPU Sec.)		Factor
(R)	`linlog`	[4 706.29, 4 786.85]	1.16
	`poly`	[4 611.82, 5 022.09]	1.22
	`linlog+lin`	[4 498.23, 4 797.24]	1.16
	observation	[4 115.58, 4 192.05]	-
(M3)	`linlog`	[1 745.96, 1 766.13]	1.04
	`poly`	**[1 703.52, 1 788.02]**	1.06
	`linlog+lin`	**[1 660.95, 1 727.87]**	1.02
	observation	[1 692.21, 1 722.13]	-
(N)	`linlog`	[3 043.42, 3 092.70]	1.26
	`poly`	[2 728.16, 3 022.21]	1.23
	`linlog+lin`	[2 700.77, 2 930.88]	1.19
	observation	[2 457.71, 2 483.12]	-

by our theoretical expectations [16]. The `exp` model clearly over-estimated the data for all three configurations, so we do not discuss it further. The predictions from the `linlog` and `poly` models over-estimated the test data for some of the configurations, and those from the `linlog+lin` model were consistent with the test data for all of the configurations, as shown in Table 3. However, the `poly` and `linlog` models still provide tight upper-bounds for the test data. To challenge this finding, we ran each version of QuickSort on eleven instances with $n = 100\,000\,000$ (See Table 4, where "factor" is a 95% confidence upper bound on how much the model predictions over-estimate the observations and where we show predictions that are consistent with the observations in boldface). For these tests, the predictions from any of the models over-estimated the observed running times by a factor of at most 1.26, and the `linlog+lin` model continued to provide the best predictions. Considering that these instances are 33 times larger than the largest training instance size, all models are providing rather accurate predictions.

We also studied Python 2.7's implementation of TimSort [32] (the default sorting algorithm in Python, callable via `sorted(lst)`) for list `lst`), with running time complexity in $O(n \log(n))$ and $\Omega(n)$ and expected running time in $O(n \log(n))$. Again, the `exp` model clearly over-estimated the data, so we do not discuss it further. The other three models produced better results. According to test loss, the best model for the median scaling was the $\texttt{poly}(x) = 7.77 \cdot 10^{-8} \cdot x^{1.19}$ model, with the $\texttt{linlog+lin}(x) = 2.14 \cdot 10^{-7} \cdot x \cdot \log(x) - 1.85 \cdot 10^{-6} \cdot x$ model as a close second, followed by $\texttt{linlog}(x) = 8.79 \cdot 10^{-8} \cdot x \cdot \log(x) - 0.04$ (the 95% confidence intervals for their test losses are all overlapping). However, while the `poly` and `linlog+lin` model predictions are both consistent with the test data, the `linlog` model was reported to under-estimate it. Again, we further challenged the models by performing eleven runs at instance size $n = 100\,000\,000$, and used these to estimate the observed median running time at 201.05 with a 95% bootstrap confidence interval of [199.30, 233.32]. We found that for this instance size the confidence interval for the `poly` model [256.30, 308.06] no longer fits the data well, but instead provides a relatively tight upper bound. Similarly, the `linlog` model continues to provide a lower bound for the observations [159.86, 166.07], as predicted. However, the confidence interval for the `linlog+lin` model is almost identical to the interval for the observations [205.88, 223.10]. This result agrees very well with what we would expect from theory, since the `linlog+lin` model contains two scaling terms that match theoretical upper and lower bounds on performance, respectively.

4 COMPARISON TO ESA V1

In our case studies in Section 3, we were unable to compare our new method to the earlier methodology for empirical scaling analysis, which is available as a tool called ESA v1[5] (Empirical Scaling Analyzer), because none of the instance sets have instances grouped by size, as required by the underlying approach. Therefore, to compare both approaches, we applied ESA v2 to some of the scenarios previously studied with ESA v1. In particular, we chose two scenarios from the work by Mu and Hoos [26]: BalancedZ [24] and march_hi [15] on random 3-SAT phase transition instances, and one scenario from the work by Mu et al. [27]: (the default configuration of) EAX [28] on RUE TSP instances.

4.1 3-SAT Phase Transition Scenarios

Upon examining the random 3-SAT phase transition instance set by Mu and Hoos [26], we noticed that there was a simple off-by-one error in the code they used to generate these instances. After correcting the error and using their code to generate a new set of instances with the correct clause-to-variable ratios, we repeated their analysis using ESA v1. We observed no qualitative difference from their original results.

In Table 5, we can see the best fit models from running the two methods for march_hi. Both methods picked the same model class and in fact fit almost identical models in this scenario (more precision than shown is required to see the difference). Both methods indicated that the model predictions had the same level of consistency with the test data when using bootstrap analysis. Another metric that we can look at between the two methods is the size of the model prediction's bootstrap intervals. In theory, smaller bootstrap intervals correspond to greater statistical power. We define the size of an interval to be the upper bound divided by the lower bound. The `exp` model for ESA v2 has a 95% confidence interval for the model predictions at instance size 1 000 of [1 651, 2 977], therefore, its interval size is 1.80, which is slightly smaller than that for ESA v1 at 2.20. One possible reason for this is that ESA v2 is able to obtain models that fit the data more robustly. However, the cost for this increase in statistical power is that ESA v2 also requires additional time to perform the analysis (33 695 CPU seconds compared to 665).

In Table 6, we show the best-fitted models for BalancedZ on the random 3-SAT phase transition instances. Rows 1 and 3 provide

[5]https://www.cs.ubc.ca/~w-esa/ESA

Table 5: ESA v1 *vs* ESA v2 for march_hi

Method	Best-Fit Model	Fit
ESA v1	$7.30 \cdot 10^{-5} \cdot 1.032^x$	fits
ESA v2	$7.50 \cdot 10^{-5} \cdot 1.032^x$	fits

Table 6: ESA v1 *vs* ESA v2 for BalanacedZ

Method	Best-Fit Model	Fit
ESA v1 - full	$3.20 \cdot 10^{-9} \cdot x^{2.69}$	fits
ESA v2 - full	$5.21 \cdot 10^{-8} \cdot x^{2.23}$	under
ESA v2 - no small	$1.77 \cdot 10^{-9} \cdot x^{2.79}$	fits

the results of applying ESA v2 and ESA v1 to the full running time data set, respectively. In this case, the results of the analysis do not match. We believe that the reason for this lies in the different methods used to fit the models. Since ESA v1 uses least squares to fit to summary statistics of each instance size, it is implicitly biased towards weighting larger running times more heavily. In contrast, the `poly` model fitted by ESA v2 weights the different instance sizes more evenly and therefore provides more of a "compromise" between the smaller and larger training instance sizes.

Manual inspection of the residuals for the `poly` model fitted by ESA v1 reveals that the observations for the smallest two instance sizes are larger than the fitted model predictions, while the remaining instance sizes have model predictions that are consistent with the observations (data not shown). This observation indicates that the `poly` model fitted by ESA v1 does not completely describe the empirical scaling behaviour of BalancedZ on the training instance set. However, if there are reasons to believe that the running times on the smallest instance sizes are unreliable (*e.g.*, because of floor effects or lower-order terms), ESA v1's model may still provide better predictions for the asymptotic scaling of BalancedZ. Nevertheless, under this assumption, a user-defined weighting scheme can be used with ESA v2 that weights larger instance sizes more heavily. In particular, we re-ran our analysis with ESA v2 using a very simple binary weighting scheme: weight 0 for the first two instance sizes and weight 1 for the remainder. We present the results of this analysis in row 3 of Table 6. In this case, we see that ESA v2 provides a result much more similar to that from ESA v1, and the `poly` model is now reported to fit the test data. We believe that ESA v2's ability to explicitly capture such information offers a more principled approach than ESA v1's implicit heuristic assumption that smaller instance sizes are less important.

For the largest test instance size ($n = 1\,000$), the size of the confidence interval is 1.74 for ESA v1's `poly` model predictions. For ESA v2 with the full data set, the size of the interval is 1.66, compared to 1.33 when omitting the smallest two instance sizes, providing further evidence that ESA v2 has more statistical power than ESA v1. However, once again ESA v2 required more running time than ESA v1 (6 748 CPU seconds with the full training set and 321 CPU seconds with the reduced training set compared to 57 CPU seconds for ESA v1). It is also interesting to note that ESA v2 without the smallest two instance sizes took substantially less time to run

Table 7: ESA v1 *vs* ESA v2 for EAX

Method	Best-Fit Model	Fit
ESA v1	$0.25 \cdot 1.12^{\sqrt{x}}$	fits
ESA v2	$0.20 \cdot 1.13^{\sqrt{x}}$	fits

than with the full training set; we believe there are two reasons for this: First, the training time is approximately cubic in the size of the training set (since it performs matrix multiplication to solve the linear least squares regression problem); and second, much fewer iterations of the iteratively reweighted least squares algorithm were needed to fit the models, since the initial mean regression problem provided a better initial estimate for the median scaling when the two smallest (outlying) instance sizes were removed.

4.2 EAX on TSP RUE Instances

We also studied the running time data set used by Mu et al. [27] to study the empirical running time complexity of EAX [28], a state-of-the-art genetic algorithm for the TSP. We downloaded and ran the latest version of ESA v1 using the same three models from their original analysis: ($\texttt{exp}(x) = a \cdot b^x$, $\texttt{sqrtexp}(x) = a \cdot b^{\sqrt{x}}$ and $\texttt{poly}(x) = a \cdot x^b$) on the data and compared it to the analysis using ESA v2. The resulting best-fit models are shown in Table 7. We note that the best-fit models are very similar, and the automated analysis produces the same qualitative statements regarding the consistency of model predictions with observed test data.

Again, the bootstrap intervals for the model predictions are slightly smaller for ESA v2 than for ESA v1 at 1.075 *vs* 1.081, respectively. As in previous experiments, ESA v2 took substantially longer to run than ESA v1 (227 243 CPU seconds compared to 52 CPU seconds for ESA v1). We note that this was by far the largest data set we studied (with 11 000 training instances); for most of our other scenarios (all of which had less than 1 500 training instances), we were able to run ESA v2 in well under an hour.

5 CONCLUSION

The most significant contribution of this work is the introduction of a statistically principled method for performing empirical running time scaling analysis on instance sets that are not grouped by bins. This procedure allows advanced statistical analysis of empirical performance scaling to be applied to a substantially broader range of interesting problem instances sets. As a second contribution, we have implemented our method in an easy-to-use tool: ESA v2 [6].

We have demonstrated the effectiveness of ESA v2 by using it to reveal new insights about the empirical scaling of several prominent algorithms. Specifically, we showed that the well-known SAT solver lingeling [4] appears to obtain not only sub-exponential scaling, but even *sub-polynomial* scaling on bounded model checking SAT instances. Furthermore, ESA v2 was able to accurately discriminate between models differing by only a $\log(x)$ lower-order term when applied to QuickSort. When we further tested our method on even larger test instance sizes (see Table 4), the correct scaling model class was no longer always consistent with the observations, but it still nevertheless provided very accurate predictions considering the magnitude of the difference between the test instance size and the largest training instance size (a factor of 33). We also applied ESA v2 to the training times for several prominent machine learning algorithms on the MNIST digits data set [23]. We showed several surprising results: First, random forests [6] appeared to obtain sub-linear scaling (*i.e.*, it probably did not need the full training set). Second, k nearest neighbours [12] exhibited super-polynomial scaling (perhaps indicating a bug). And third, support vector machine [9] scaling was consistent with a lower-degree polynomial model than expected (≈ 1.51 instead of 2). Finally, we demonstrated that ESA v2 can obtain similar (but more robust) results to ESA v1 by applying it to three previously studied applications with instances grouped into bins.

There are two common pitfalls that are important to avoid when performing empirical scaling analysis. First, the target algorithm must be run on the instances in a random order, otherwise environmental noise from background processes is not identically and independently distributed, which can lead to erroneous conclusions. In some of our preliminary work, we were quite surprised by the results until we realized we had fallen subject to this pitfall. Second, it is essential to ensure that only one property controlling instance difficulty is varied, otherwise the scaling behaviour of algorithms may abruptly change in unpredictable ways. In fact, we discovered precisely such a scenario working with FCC SAT instances [29] where the scaling of SAT solvers first appeared to be benign, before abruptly becoming super-exponential. Upon further study, we found evidence for a previously undiscovered phase transition in the underlying instance distribution.

Perhaps the biggest open challenge for future work lies in the accurate detection and modelling of lower-order terms. If prior information is available, our method can facilitate user-defined weights to specify which training data should be trusted the most (to effectively ignore lower-order terms) or to explicitly add additional scaling terms (so that lower-order terms can be included without additional model complexity). However, it may also be possible to add lower-order terms by developing a grey-box approach (see, *e.g.*, Goldsmith [13]) that uses additional profiling information about the target algorithm runs to build more complex models, *i.e.*, by measuring, modelling and then combining the performance of different loops or function calls within an algorithm, thereby considering the sources of each scaling term independently. Such a procedure could be immensely useful in practice, by allowing algorithm developers to use small scale experiments to determine where algorithmic bottlenecks occur for large scale instances, since in practice different bottlenecks often occur for different instance sizes. An alternative direction is to include hypothesis testing to determine whether or not the running time distribution of the test set is similar to the distribution on the training set. A dissimilar running time distribution may indicate that a scaling model that appears to fit the data well may not be accurately summarizing all of the relevant information and hence may become unreliable for even larger instance sizes. Future work could also extend the method to predict running time as a function of multiple instance features (*e.g.*, the number of edges and vertices in a graph), or to apply the method to other performance metrics.

[6]https://www.cs.ubc.ca/~w-esa/ESA/

REFERENCES

[1] C. Ansótegui, Y. Malitsky, H. Samulowitz, M. Sellmann, and K. Tierney. 2015. Model-Based Genetic Algorithms for Algorithm Configuration. In *Proc. of IJCAI 2015*. 733–739.

[2] J Bentley and MD McIlroy. 1993. Engineering a sort function. *SPE* 23, 11 (1993), 1249–1265.

[3] A Biere. 2008. Hardware Model Checking Competition. http://fmv.jku.at/hwmcc/. (2008). Last visited on 2019-02-15.

[4] A. Biere. 2017. CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT Entering the SAT Competition 2017. In *Proc. of SAT Comp. 2017*. 14–15.

[5] A Biere, K Heljanko, and S Wieringa. 2011. *AIGER 1.9 And Beyond*. Technical Report. FMV Reports Series, Institute for Formal Models and Verification, Johannes Kepler University.

[6] L Breiman. 2001. Random forests. *Mach. Learn.* 45, 1 (2001), 5–32.

[7] R Brown. 2015. Building a Balanced k-d Tree in $O(kn \log n)$ Time. *JCGT* 4, 1 (2015), 50–68.

[8] E Coppa, C Demetrescu, and I Finocchi. 2012. Input-sensitive profiling. *ACM SIGPLAN Notices* 47, 6 (2012), 89–98.

[9] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning* 20, 3 (1995), 273–297.

[10] P Cunningham and SJ Delany. 2007. k-Nearest neighbour classifiers. *MCS* 34, 8 (2007), 1–17.

[11] K Eggensperger, M Lindauer, and F Hutter. 2018. Neural Networks for Predicting Algorithm Runtime Distributions.. In *Proc. of IJCAI*. 1442–1448.

[12] E Fix and J Hodges Jr. 1951. *Discriminatory analysis-nonparametric discrimination: consistency properties*. Technical Report. UC Berkeley.

[13] S Goldsmith. 2008. *Measuring empirical computational complexity*. Ph.D. Dissertation. UC Berkeley.

[14] S Goldsmith, A Aiken, and D Wilkerson. 2007. Measuring empirical computational complexity. In *Proc. of ESEC/FSE*. ACM, 395–404.

[15] M Heule and H van Marren. 2009. march_hi: Solver description. In *Proc. of SAT Comp. 2009*. 23–24.

[16] C Hoare. 1962. Quicksort. *Comput. J.* 5, 1 (1962), 10–16.

[17] P Holland and R Welsch. 1977. Robust regression using iteratively reweighted least-squares. *Commun. Stat. Theory Methods* 6, 9 (1977), 813–827.

[18] F Hutter, H Hoos, and K Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION (LNCS)*, Vol. 6683. 507–523.

[19] F Hutter, M López-Ibáñez, C Fawcett, M Lindauer, H Hoos, K Leyton-Brown, and T Stützle. 2014. AClib: A Benchmark Library for Algorithm Configuration. In *Proc. of LION*. 36–40.

[20] T Joachims. 1999. In *Advances in Kernel Methods*. MIT Press, Chapter Making Large-scale Support Vector Machine Learning Practical, 169–184.

[21] T Joachims. 2006. Training linear SVMs in linear time. In *Proc. of KDD*. ACM, 217–226.

[22] R Koenker and K Hallock. 2001. Quantile regression. *JEP* 15, 4 (2001), 143–156.

[23] Y LeCun, L Bottou, Y Bengio, and P Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. of the IEEE* 86, 11 (1998), 2278–2324.

[24] C Li, C Huang, and R Xu. 2014. Balance between intensification and diversification: a unity of opposites. In *Proc. of SAT Comp. 2014*. 10–11.

[25] C McGeoch, P Sanders, R Fleischer, P Cohen, and D Precup. 2002. In *Experimental Algorithmics*. Springer-Verlag New York, Inc., Chapter Using Finite Experiments to Study Asymptotic Performance, 93–126.

[26] Z Mu and H Hoos. 2015. On the empirical time complexity of random 3-SAT at the phase transition. In *Proc. of IJCAI*. 367–373.

[27] Z Mu, H Hoos, and T Stützle. 2016. The Impact of Automated Algorithm Configuration on the Scaling Behaviour of State-of-the-Art Inexact TSP Solvers. In *Proc. of LION (LNCS)*, Vol. 10079. 157–172.

[28] Y Nagata and S Kobayashi. 2013. A Powerful Genetic Algorithm Using Edge Assembly Crossover for the Traveling Salesman Problem. *INFORMS JOC* 25, 2 (2013), 346–363.

[29] N Newman, A Fréchette, and K Leyton-Brown. 2017. Deep optimization for spectrum repacking. *Commun. ACM* 61, 1 (2017), 97–104.

[30] Y Pushak and Z Mu and H Hoos. 2020. Empirical Scaling Analyzer: An Automated System for Empirical Analysis of Performance Scaling. *AI Communications, to appear* (2020).

[31] F Pedregosa, G Varoquaux, A Gramfort, V Michel, B Thirion, O Grisel, M Blondel, P Prettenhofer, R Weiss, V Dubourg, J Vanderplas, A Passos, D Cournapeau, M Brucher, M Perrot, and E Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *JMLR* 12 (2011), 2825–2830.

[32] T Peters. 2015. Timsort description. (2015).

[33] K Yu and MC Jones. 1998. Local linear quantile regression. *JASA* 93, 441 (1998), 228–237.

[34] D Zaparanuks and M Hauswirth. 2012. Algorithmic profiling. *ACM SIGPLAN Notices* 47, 6 (2012), 67–76.