University of British Columbia
Department of Computer Science
Technical Report TR-2009-16

A Bootstrap Approach to Analysing the Scaling of Empirical Run-time Data with Problem Size

Holger H. Hoos
University of British Columbia
Computer Science Department
hoos@cs.ubc.ca

15 June 2009

Abstract

In this report, we introduce a novel approach for analysing the scaling of empirical run-time data of an algorithm when applied to sets of inputs of growing size. Our method is based on the use of standard numerical techniques for fitting models, which are then challenged by extrapolation to larger problem sizes and statistically validated using bootstrap confidence intervals. It permits not only the automatic construction of predictive models of the given algorithm's run-time, but also the comparative evaluation of multiple hypothesis on the scaling in the form of different parametric functions. We illustrate our method using run-time data for Concorde, a state-of-the-art complete algorithm for the travelling salesperson problem (TSP), applied to a class of well-known Euclidean TSP instances. [1]

1 Motivation

The scaling of run-time with input size is of central interest in the design, analysis and application of many algorithms. Traditionally, scaling results have been established using theoretical methods, and such results along with the methods used for establishing them play an important role in theoretical computer science. Clearly, theoretical scaling results are often very useful, for example, in the context of assessing algorithms based on provable bounds on their run-time. However, to render mathematical treatment possible, such theoretical analyses typically make use of various simplifications:

1. The scaling behaviour is often characterised in terms of a family of functions, usually specified using notation such as $O(\cdot)$ or $\Theta(\cdot)$, that describes the asymptotic scaling of run-time with input size.
2. The analysis typically only considers extreme cases (in particular, the worst case) or average-case-behaviour on a given distribution of problem instances, parameterised by instances size.
3. Sometimes, additional simplifying assumptions are made regarding details of the algorithm or the input data, such as statistical independence of heuristic choices from problem instance features, or of instance features from each other.

[1]Here, this data (which has been generated in a joint project with Thomas Stützle) is used for illustrative purposed only; it is discussed and analysed in more detail in TR-2009-17 (Hoos and Stützle, 2009).

4. In some cases, an algorithm of interest cannot be analysed theoretically, and a modified (typically simpler) variant of the algorithm is studied instead.

5. Low-level aspects of the execution environment in which an algorithm is executed, in particular related to CPU cache and branch prediction, are typically considered in an idealised way or not at all.

In practice, these simplifications can, and often do, limit the extent to which theoretical results capture important aspects of the observed scaling of run-time, particularly in the case of high-performance algorithms that use sophisticated heuristics to solve practically relevant problems as efficiently as possible. For $\mathcal{NP}$-hard problems, empirical methods are usually the only way for characterising run-time such that practically relevant performance differences between various high-performance algorithms are captured (see, *e.g.*, Johnson and McGeoch, 2002; Hoos and Stützle, 2000). Similarly, for many practically relevant polynomial-time solvable problems, analytical worst- and average-case results are often complemented by extensive empirical studies to inform the choice or configuration of a procedure to be used in a given practical situation, as witnessed in the literature on sorting procedures (see, *e.g.*, Bentley and McIlroy, 1993; Li et al., 2004; Biggar et al., 2008).

Overall, for most computational problems of practical interest, there is a sizable gap between the performance guarantees that can be achieved using theoretical analysis and the performance observed in practice from carefully designed, practically useful algorithms. Consequently, the scaling behaviour of those latter algorithms needs to be studied and modelled based on empirical observations.

Mathematical models for the scaling of an algorithm's run-time in dependence of input size can serve various purposes. Firstly, they can be used to compactly summarise observed scaling behaviour while abstracting from details deemed unimportant. Secondly, they can be used to make predictions regarding the algorithm's run-time for input sizes for which run-time measurements have not been performed. Both types of uses, to the extent that they refer to algorithms and distributions of problem instances of interest in some application context, have important applications. Characterisation plays an important role in the comparison of algorithmic performance, and therefore also in the reporting of performance results which may be used for comparative purposes at some later stage (which applies to almost all scientific publications). Prediction is often used for assessing the suitability of an algorithm for solving problem instances in a given application context; this is particularly valuable, and sometimes indispensable, in cases where it is unclear whether the instances sizes occuring in practice can be solved within the available computational resources.

Scaling models, whether derived from the theoretical analysis of an algorithm or based on empirical run-time data, come in two distinct flavours: models that bound observed run-times and models of typical case run-time. Both types of models can be useful for assessing the suitability of an algorithm for a particular application as well as for the comparative evaluation of algorithms. Bounding models have the advantage of explicitly expressing the direction in which observed run-times can be expected to deviate from those obtained from the model. However, in order to obtain a reliable bound, it is often necessary to sacrifice some accuracy compared to models of typical case run-time (where 'typical' may refer to the average or median over an ensemble of instances and/or over multiple runs of a randomised algorithm). While in the following, we focus on models of typical case run-time, we note that the techniques we discuss apply to a large extent also to bounding models; in particular, this is the case for the bootstrap analysis presented in Section 6.

In the following, we will focus on the empirical scaling of the run-time of an algorithm for solving a given problem, such as the travelling salesperson problem (TSP), with the size of the instances of the problem to be solved, measured not necessarily in the total size of the input characterising a problem instance, but typically in terms of a salient feature, such as the number of locations to be visited in the case of the TSP. However, the methods we will discuss are more general, and can be easily applied to other performance measures (such as memory consumption) and instance characteristics (such as the variation coefficient of the distance matrix for a given TSP instance).

2 Collecting Run-time Data

Any analysis of the empirical scaling of the run-time of a given algorithm begins with the collection of run-time data. In this context, two fundamental issues arise: Which problem instances to use for the study and how to perform run-time measurements on these instances. We discuss these issues by considering the following six questions:

1. How do we select the (types of) benchmark instances used for our study?
2. Which instance sizes should be considered?
3. How many problem instances should be considered per instance size?
4. How do we measure the run-time required by the given algorithm for solving a given problem instance?
5. How do we deal with variations in run-time observed over several instances of the same size?
6. How do we deal with variations in the run-time observed over multiple runs of a randomised algorithm on the same problem instance?

Question 1 from this list has been discussed in general terms elsewhere (see, *e.g.*, Hoos and Stützle, 2004, Chapter 4), and the specific answer often depends on the problem (and algorithm) under consideration. Often, empirical scaling studies make use of random instance generators; we note that by deriving the instance distributions underlying such generators from 'real-world' instances, this approach can maintain a focus on instance characteristics relevant in practical applications (see, *e.g.*, Leyton-Brown et al., 2000; Andronescu et al., 2004; Aguirre-Hernández et al., 2007). Using an instance generator has the advantage that large numbers of instances per instance size can be obtained, which facilitates empirical scaling studies in various ways, as will become apparent in the following.

The answer to **Question 2** from our list, regarding the instance sizes to be used in a scaling analysis, depends strongly on the goals of the study. In many cases, these goals involve predicting the run-time of one or several given algorithms for instance sizes larger than those considered in the analysis (*extrapolation*). In other cases, the focus is on predictions for previously not considered, intermediate problem sizes (*interpolation*). In both cases, a parametric function called a *scaling model* is fitted to the run-times observed for a number of instance sizes and the values of that fitted function are then used as predictions. The data to which a model is fitted is called the *support* of the model, and we call the interval formed by the smallest and the largest instance size in the support the *range of the support*. The process of fitting a scaling model to observed run-time data and of challenging this model by comparing predictions instance sizes different from those in the support will be described in detail in Sections 4 and 5 of this report. However, it is intuitively clear that the accuracy of a scaling model typically benefits from a large range of support containing many different instance sizes.

For algorithms whose run-time is known or expected to scale super-polynomially or polynomially with large degree (*i.e.*, degree larger than 3), the instances sizes used in the support typically form an arithmetic series, while in the case of sub-polynomial scaling or polynomial scaling with small degree, arithmetic or geometric series are used.[2] Furthermore, in order to maximise prediction accuracy for an extrapolation challenge, it is useful to keep the distance between the largest instance size in the support and the smallest instance size for which a run-time prediction is desired relatively small. At the same time, if the main interest is to arrive at models that are as general as possible in terms of generating reasonably accurate predictions over a very large range of instance sizes, extrapolation challenges to instance sizes much larger than those contained in the support can provide more compelling empirical evidence in favour of a given model. (This issue will be further discussed in Sections 5 and 6.) Given limited overall computation time, there can furthermore be trade-offs between the number and range of instance sizes used for constructing a model and the number of instances per size.

[2]A common variation are pseudo-geometric progressions of instance sizes, such as $10, 20, 50, 100, 200, 500, 1000$.

The question regarding the number of instances to be used per problem size (**Question 3** from our list) is closely related to the way in which we deal with variations in run-time between different instances and multiple runs of a randomised algorithm on the same instance (Questions 5 and 6). We will therefore postpone discussing this question until the end of this section; since the ultimate answer also depends on other aspects of the analysis, we will furthermore revisit it in Sections 5 and 6 of this report.

Before addressing Questions 4–6, we make the following assumptions:

- The software implementation of the algorithm to be analysed, from here on referred to as *the solver*, takes as its only input a problem instance to be solved. For reasons of reproducibility, even in cases where instances are generated by means of a randomised instance generator, explicit representations of the instance data should be stored in files, ideally in a human-readable plain-text format. These files are read by the solver; any other arguments or parameters of the solver are kept constant.
- The solver terminates when a given problem instance is solved, reporting that fact along with data on the run-time used and possibly additional information on the solution found and the process of producing this solution (this may be useful for verifying solutions or performing more in-depth analyses of the behaviour of the solver).
- All runs of the solver are successful and error-free, *i.e.*, have resulted correct solutions to the respective problem instances (we will comment later on how to deal with cases in which runs had to be aborted due to excessive run-time requirements).

We now turn our attention to to **Question 4**, namely the manner in which the time required for a single run is measured. It has been argued elsewhere that for maximum reproducibility and comparability of results, run-time measurements should be performed in a manner that abstracts as much as possible from the execution environment (machine and operating system; see Hoos and Stützle, 2004, Ch. 4). This can be achieved by performing (machine-load-independent) CPU time measurements; as an additional precaution, it is advisable to keep the amount of other processes as well as memory, disk and network usage on the machine(s) used for run-time measurements to a minimum. Also, particularly in cases where the resolution of typical CPU time measurement methods become problematic (*i.e.*, in situations where run-times may be in the order of CPU seconds or less), it is advisable to measure run-time by using cost models based on suitably defined elementary operations (whose actual run-time needs to be constant for any given problem instance). When used appropriately, this enables CPU time measurement accuracies far greater than the resolution provided by the typical process timing mechanisms provided by the operating system. (For a more detaild discussion, see Hoos and Stützle, 2004, p. 169; operation counts have also been advocated by Ahuja and Orlin, 1996, Sanders and Fleischer, 2001, and Goldsmith et al., 2007.)

Question 5 from our list deals with the variation of the run-time of a given solver over problem instances of the same size; this variation can be, and often is, quite large. This is particularly the case for high-performance heuristic algorithms for computationally hard problems, where it is not uncommon for the run-times observed on instances of the same size to differ by multiple orders of magnitude (see, *e.g.*, Gomes et al., 1997; Hoos and Stützle, 1998; Gomes et al., 2000). Therefore, empirical scaling analyses are often focussed on location statistics of the distributions of run-time over instances of the same size (so-called *solution cost distributions*, short *SCDs*), such as the mean or median run-time. While the empirical mean run-time approximates the run-time a solver would be expected to require for solving an arbitrary instance of a given size, this expectation may not be very meaningful in the presence of SCDs with extreme variability; in fact, for so-called heavy-tailed SCDs, which have occasionally been observed for high-performance algorithms (see, *e.g.*, Gomes et al., 2000), the theoretical mean is infinite and the empirical mean behaves erratically. Therefore, it is often preferably to focus on median run-time instead; this has the additional advantage that in the presence of instances on which runs had to be aborted due to excessive run-time requirements, the median (unlike the mean) can still be determined (by considering the run-time of these censored runs to be larger than the cut-off time used).

While it can be of interest to report minimum and maximum run-times as empirical bounds on best-case and worst-case performance, they are typically unsuitable as a basis for scaling studies of empirical worst-case (or best-case) performance; the reason for this lies in the statistical instability of the sample maximum (or minimum), particularly if the underlying distibution has a fat or even heavy right tail.[3] Instead, higher (or lower) SCD quantiles, such as the 75-, 90- or 95-percentile, can provide a solid basis for studying the scaling of relatively high (or low) run-times observed for instances of a given size; it should be noted, however, that in order to obtain reasonably stable estimates of these quantiles, larger numbers of instances per problem size are required than for simply estimating the median.

Finally, **Question 6** from our list arises because high-performance solvers for many problems are based on randomised algorithms, that is, algorithms, whose run-time varies betweeen multiple independent runs on the same problem instance as a result of decisions within the algorithms that are made using a randomised mechanisms (these are usually implemented using data obtained from a pseudo-random number generator). In that case, the behaviour of the algorithm on any given instance is not characterised by any single run-time value, but by the probability distribution of run-times over multiple independent runs, the so-called *run-time distribution (RTD)*. As has been argued elsewhere, in this situation, it can be very important to distinguish the variation of run-time across multiple runs on the same problem instance (caused by randomised decisions within the algorithm and characterised by the RTD) from variation in run-time observed between different problem instances of the same size Hoos and Stützle (1998, 2004).

However, it has been shown that under certain circumstances, in particular when investigating solely the average run-time over both, instances and runs on the same instance in situations where at least as many instances are available as runs can be performed (*e.g.*, as a result of being able to produce arbitrary numbers of problem instances using a random instance generator), it is advantageous to perform only one run per problem instance (Birattari, 2004). To simplify our discussion without making overly restrictive assumptions, in the following, as far as randomised algorithms are concerned, we will focus on types of scaling analysis that only consider a location statistic of the RTD of each given problem instance, such as the mean or median; once such a statistic has been empirically determined (based on multiple runs of the algorithm on the given instance), it is treated in exactly the same way as the uniquely defined run-time of a deterministic algorithm. Therefore, when analysing the empirical scaling of run-time for a randomised solver, the run-time for each instance size could be the mean over instances and multiple runs or, in light of the considerations regarding high-variance SCDs, the median mean run-time (where the mean is taken over multiple runs on the same instance, and the median is taken over the means for multiple instances) or, if there is also high variability in the RTDs, the median median run-time.

Based on our discussions of Questions 5 and 6, we can now briefly make some general statements regarding the number of instances to be considered per instance size (**Question 3** from our list). In a nutshell, the number of instances needs to be large enough to obtain reasonably stable estimates for the run-time statistics collected over instance sets and, in the case of randomised solvers, over multiple runs on each individual problem instance. The higher the variability of run-time over instances of the same size, the more instances are needed to obtain stable estimates for mean run-time. The same holds, to a lesser extent, when measuring median run-times; furthermore, when considering higher (or lower) run-time quantiles, even larger numbers of instances per size are required to achieve reasonable statistical stability, depending on the probability mass found in the tails of the respective solution cost distribution. In addition, for randomised algorithms the accuracy of the run-time statistics obtained for individual instances is limited by the number of independent runs performed for each instance; as a consequence, typically larger sets of instances are needed to obtain similarly stable run-time statistics for each given instance size than in the case of deterministic algorithms.

From here on, we will assume that sufficiently many instances per instance size (and, in the case of randomised solvers, runs per instance) have been performed to obtain reasonably stable run-time statistics for each problem size. We will revisit the choice of these sample sizes in Sections 6 and 7. In the following, we will use $n_1, n_2, \dots n_k$ to denote the series of instance sizes that comprise the support of the models to be constructed, and $t(n_i)$ to refer to the aggregate run-time over the instances of n_i, where some statistic (such as mean or median) is used for aggregation. We will further denote the run-time data, aggregated by instance size, on a given instances set I as $B(I)$.

[3]Unfortunately, there is some disagreement regarding the definitions of the terms *heavy-tailed* and *fat-tailed* in the literature; we consider a distribution to be *fat-tailed* if, and only if, at has excess kurtosis larger than zero (*i.e.*, if at least one of its tails decays slower than that of a normal distribution) and *heavy-tailed* if, and only if, at least one tail decays according to a power-law, *i.e.*, following $x^{-(1+\alpha)}$ for some $\alpha > 0$. According to those definitions, heavy-tailed distributions are a special case of fat-tailed distributions.

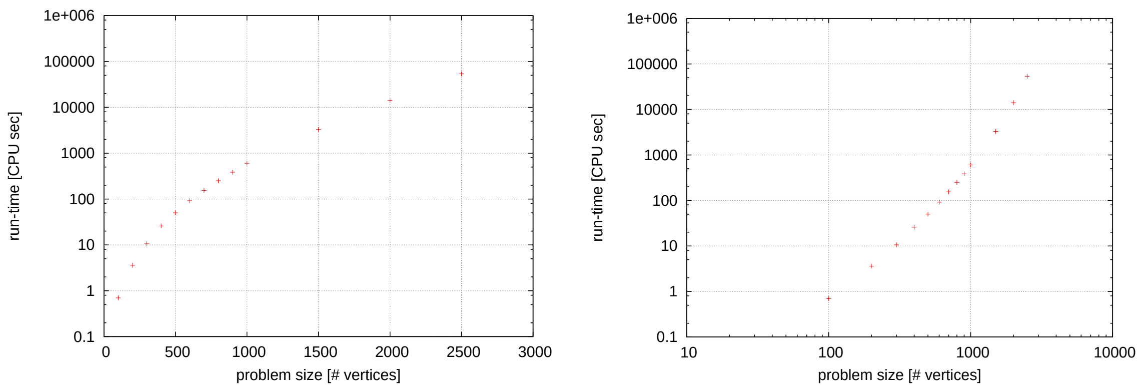

Figure 1: Scaling of mean run-time required by the Concorde solver for solving RUE TSP instances (*left pane:* semi-log plot; *right pane:* log-log plot). The data uses for this plot has been taken directly from the book by Applegate et al. (2006), p.496, Table 16.6; mean run-times for $n \leq 1000$ are based on 10 000 instances per instance size and those for $n > 1000$ are based on 1 000 instances per size.

3 Graphical Analysis

Perhaps the simplest way to empirically characterise and analyse the scaling behaviour of a given algorithm is to plot its observed run-time, $t(n)$, *vs* instance size, n. Depending on the type of plot used, various types scaling behaviour can be detected very easily, exploiting the ability of the human visual system to perceive straight lines. While linear trends can be detected directly in a standard Cartesian plot, logarithmic and exponential scaling is indicated by linear trends in semi-logarithmic plots (in which instance or run-time are shown on a logarithmic scale, respectively), and arbitrary polynomial scaling by linear trends in a log-log plot (*i.e.*, using logarithmic scales for run-time and instance size).

The example shown in Figure 1 illustrates this type of graphical scaling analysis. The semi-log plot in the left pane is a reproduction of Figure 16.1 from the book of Applegate et al. (2006), who interpret it as follows (p. 496):

> The plot of mean values in Figure 16.1 indicates that the running times are increasing as an exponential function of n, [...]

This interpretation is supported by the fact that for increasing instance size, the observed mean run-times appear to asymptotically approach a straight line, while the deviations from that line may indicate atypical behaviour for small instance sizes. Interestingly, a log-log plot of the same data (shown in the right pane of Figure 1) admits the same interpretation with respect to polynomial scaling, which indicates the limitations of this type of scaling analysis.

An actual scaling model can be determined by fitting a line to the plotted data; this can be done either manually or by using appropriate fitting techniques on the linearised data, in particular, least-squares linear model fitting. The goodness of the resulting fits can be evaluated using standard metrics, such as the *root mean square deviation (RMSD)* or *root mean square error (RMSE)*, defined as

$$\sqrt{\sum_{i=1}^{k} r_i^2},$$

where each residual r_i is the difference between an observed run-time value, $b(n_i)$, and the corresponding value of the model, $p(n_i)$, *i.e.*, $r_i := b(n_i) - p(n_i)$.

One potential problem with this approach is that any transformation of the run-time data also leads to a transformation of the error metric minimised by least-squares fitting on the linearised data. In such cases, what is minimised by the fitting procedure no longer corresponds to RMSE on the original data, but to RMSE on the transformed run-time data. While in some circumstances, this can be desirable (for example, when in the context of exponential scaling behaviour one cares more about relative than about absolute deviations between the run-times derived from the model and those observed), in others it may introduce undesired bias into a scaling analysis.

There are factors beyond RMSE that are important when assessing the quality of a fit, in particular the presence of systematic deviations between observed run-times and a given scaling model. Such deviations can also be detected graphically, by plotting the residuals against instance size. The presence of any regularity or trend in the residuals not only indicates a systematic weakness of the scaling model, but may also suggest corrections or improvements.

The basic idea of linearising scaling data by applying appropriate transformations to instance sizes and/or run-time measurements can be generalised in a straight-forward manner by considering larger families of transformations. However, unless very clearly defined hypotheses regarding the scaling behaviour of the algorithm(s) under consideration are investigated, this approach can be labour-intensive and open-ended.

4 Model Fitting

Often, hypotheses regarding the scaling behaviour of a given algorithm take the form of a parametric function; for example, simple exponential scaling can be expressed in the form Exp$[\alpha_1, \alpha_2](n) := \alpha_1 \cdot \exp(\alpha_2 \cdot n)$, where specific members of this parametric family are obtained by instantiating the parameters α_1 and α_2. Hypotheses of this form can, for example, be based on theoretical results on the worst-case complexity of a given problem, or the (theoretically or empirically determined) scaling behaviour of another, previously studied algorithm.

Given this type of hypothesis and the same type of empirical data considered in the previous section (*i.e.*, pairs of instance sizes, n_i, and run-times, $t(n_i)$), the question arises of how to determine parameter values $\alpha_1, \ldots, \alpha_p$ such that the corresponding scaling function $F[\alpha_1, \ldots, \alpha_p](n)$ fits the given run-time data best. Clearly, this is a generalisation of the line fitting approach discussed in the previous section, and in principle, similar methods can be applied. In particular, least squares fitting techniques based on numerical optimisation procedures can be used in this context, such as the Levenberg-Marquardt Algorithm (implemented, *e.g.*, in the *fit* function of the widely used *Gnuplot* data and function plotting software).

Figure 2 illustrates the results of fitting the two 2-parameter polynomial and exponential models Exp$[a, b](n) = a \cdot b^n$ and Poly$[a, b](n) = a \cdot n^b$ to the data from our earlier example (see Figure 1). As can be seen from the semi-log and log-log plots, both scaling models provide good fits to the run-times observed for large instance sizes, but show systematic deviations for smaller instances. The reason for this lies, of course, in the fact that in the logarithmic representation of run-time used in both plots, the large relative differences between the models and the observed data for small instance sizes correspond to small differences in absolute run-time, as considered by the model fitting procedure.

Several issues have to be addressed when using this model fitting approach. The use of standard metrics that are minimised to obtain good fits, in particular RMSE, in principle provides a basis for quantitative assessments of the fits thus obtained; however, as previously mentioned, RMSE does not adequately reflect systematic deviations between the scaling model and the observed behaviour. Therefore, at least during the final evaluation of a scaling model obtained by model fitting based on RMSE-minimisation, the residuals should be analysed as described in the previous section. Particularly in cases where there is substantial interest of using a scaling model for predicting the run-time of the given algorithm for instance sizes larger than those used in the fit, residuals that systematically grow with instance size are a concern, and fits that do not suffer from this problem may be preferable to ones that do, even if their RMSE is larger (*e.g.*, due to poorer fits for small n, which may be caused by overhead of the algorithm on small instances that becomes irrelevant for larger n).

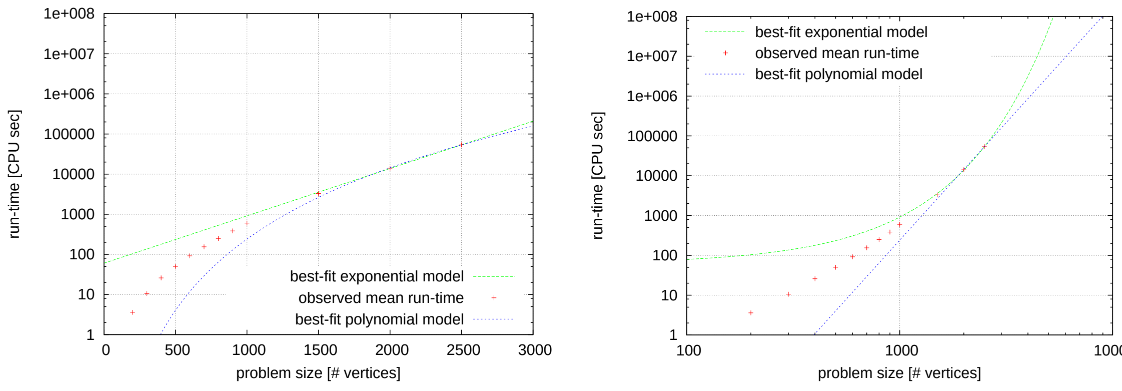

Figure 2: Scaling of mean run-time required by the Concorde solver for solving RUE TSP instances (*left pane:* semi-log plot; *right pane:* log-log plot); the exponential and polynomial scaling models shown, $60.31 \cdot 1.00272^n$ and $4.71331 \cdot 10^{-16} \cdot n^{5.90256}$, respectively, were fitted using the Levenberg-Marquardt Algorithm (as implemented in the Gnuplot 'fit' function) on the untransformed instance size and run-time data for $n = 100..2500$, resulting in RMSE values of 214.80 for the exponential and 246.83 for the polynomial model. The data uses for this plot has been taken directly from the book by Applegate et al. (2006), p.496, Table 16.6; mean run-times for $n \leq 1000$ are based on 10 000 instances per instance size and those for $n > 1000$ are based on 1 000 instances per size. (This difference in the number of instances per instance size, although not ideal in terms of experimental design, results from a compromise between the goal of obtaining accurate observations and the need to keep overall computing times within reasonable limits.)

Furthermore, standard numerical optimisation procedures, such as the Levenberg-Marquardt Algorithm, can be quite sensitive to the initial values of the parameters $\alpha_1, \ldots, \alpha_p$. Particularly for highly parameterised fitting functions, even small differences in those initial values can result in final fits of substantially different quality, due to numerical instability of the fitting algorithm or local optima in the fitting function. One strategy that tends to minimise the occurence of these problems in practice is to limit the number of parameters to be optimised, and to choose initial values such that the corresponding initial fit is close to prominent observed data points, such as those for the smallest, the largest and an intermediate problem size (depending on the number of model parameters). In the case of complex models with many parameters, preliminary fits with a reduced number of parameters (*e.g.*, by clamping a lower-order term to zero) can lead to reasonable initial parameter values for the final fit of the full model. In any case, it is advisable to carefully check the quality of the fits obtained on the full support of the fitted model, graphically or at least in terms of RMSE, and to allow for the possibility that a poor fit may reflect a problem with the fitting procedure (caused by unsuitable initial parameter values) rather than indicate that no member of the given functional family fits the empirical run-time data well.

Finally, the complexity of a model, *i.e.*, the number of parameters to be optimised, has important consequences for the quality of the fits obtained based on that model. As is well-known from statistics, machine learning and numerical optimisation, models with many parameters can typically fit a wider range of observed data; nevertheless, they do not necessarily lead to better predictions. One reason for this stems from the fact that complex models are often much more difficult to fit (as discussed previously). Furthermore, there can be a risk of overfitting the observed data: Because of the flexibility of a complex model, it can fit spurious aspects of the observed data, such as measurement inaccuracies (*e.g.*, due to randomness in the given algorithm), discretisation effects for small problem sizes or small CPU times, or spurious aspects of scaling (*e.g.*, small-size effects due to cost of initialising data structures) that are not of interest for the analysis. A related problem can in principle arise from the fact that two very different, but sufficiently highly parameterised models could potentially give equally good fits for a given set of observations; as an extreme example, consider a case where only two data points are given, and 2-parameter exponential and polynomial models are fitted to those observations. In reality, this problem does not arise in empirical scaling analyses to the same extent as it occurs in other contexts (*e.g.*, in many machine learning tasks), because the cost of computational experiments is usually small enough to allow for the collection of sufficiently large data sets.

When comparing the fits obtained using two different models, it is important to ensure that both models have similar flexibility, *i.e.*, the same or almost the same number of parameters. Therefore, it would be problematic to compare fits obtained from a 2-parameter exponential model and a 6-parameter polynomial model. At the same time, Occam's razor does apply in this context: Between two models of the same quality (*i.e.*, the same quality of fit to the given observations and the same predictive power), the simpler (*i.e.*, that with fewer parameters) is to be preferred. The application of this principle is complicated by the fact that two models rarely have exactly the same quality, and that (as previously discussed), quality may not be easily and entirely quantifiable. Therefore, the application of Occam's razor may involve the use of judgement as to when two models are considered to be of the same quality.

5 Challenging Models by Extrapolation

In all of science, models serve two main purposes: To provide a conceptual framework for understanding phenomena and to generate predictions. Mathematical models for the scaling of the run-time of a given algorithm with problem instance size are often considered useful because they allow to predict run-time on new instances. This is particularly relevant in cases where running the algorithm in question may require a large amount of limited or costly resources (such as run-time on a fast machine), or where computation is severely limited by the nature of an application (e.g., in a real-time system).

Indeed, the capacity of a model to produce falsifiable predictions is key to the scientific method of generating knowledge. Such predictions challenge the model from which they were obtained in the sense that they may turn out to be in disagreement with observations from a subsequent experiment. Models for the scaling of algorithm run-time with instance size can be subjected to *interpolation* and *extrapolation challenges*. In case of the former, the predictions concern instances sizes in-between those used for generating the observations underlying the model; in case of the latter, predictions are made for instance sizes smaller or larger than any of those considered when constructing the model. Considering the practical motivation behind scaling studies as well as the tendency of model inaccuracies to increase with distance from the model's support, extrapolation – especially to larger problem sizes – is typically more interesting and also more challenging than interpolation.

The basic procedure used for challenging a scaling model by extrapolation is as follows:

1. collect run-time data $B(I)$ for a set of problem instances I (see Section 2);
2. fit a parametric scaling model M to the data $B(I)$ (see Section 4);
3. obtain predictions $P(J)$ from M for a set a set J of instances sizes that fall outside of the range covered by D (*i.e.*, outside of the support of M);
4. collect run-time data $B(J)$ for the instance sizes in J;
5. compare the observed run-times in $B(J)$ against the predictions $P(J)$.

In Step 3, the selection of one or more instance sizes for which first predictions and subsequently observations will be made will typically be strongly based on the scaling model M and the time budget available for experiments in Step 4. Particularly in cases where substantial differences in run-time are expected on different instances of the same size, there is a tradeoff between the number of instances and the problem sizes considered: Using a larger set of instances for a given problem size will result in more accurate run-time statistics but also be costlier, and may therefore limit the extrapolation study to smaller instance sizes; for large instance sizes, on the other hand, each run may be so costly that only a small number of runs can be performed.

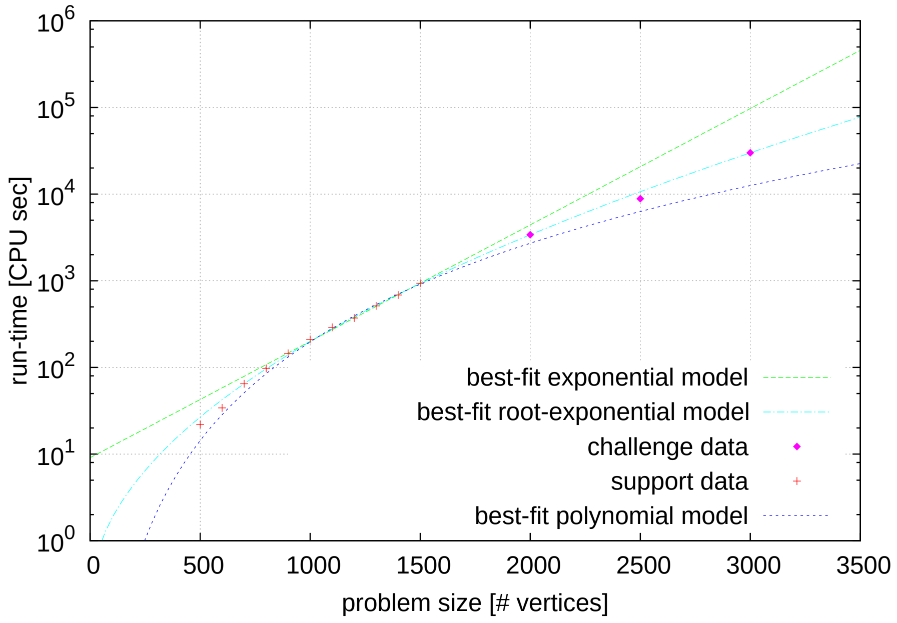

Figure 3: Scaling of median run-time required by the Concorde solver for solving RUE TSP instances; the exponential, root-exponential and polynomial scaling models shown, $9.07594 \cdot 1.0031^n$, $0.210057 \cdot 1.24194^{\sqrt{n}}$, $8.83431 \cdot 10^{-10} \cdot n^{3.78269}$, respectively, were fitted using the Levenberg-Marquardt Algorithm (as implemented in the Gnuplot 'fit' function) on the untransformed instance size and run-time data for $n = 500..1500$; the data for larger instance sizes have not been used for fitting the models. The data uses for this plot has been taken from a new scaling study by Hoos and Stützle (2009); median run-times for $n \leq 2000$ are based on 1 000 instances per instance size and those for $n > 2000$ are based on 1 00 instances per size.

Further factors that may play into the selection of problem sizes for extrapolation are the accuracy of the model on the original instance set I, the presence of systematic error in M on I and the cost of producing or obtaining large instances for inclusion in J. (The latter would arise, for example, in the context of a study of incomplete algorithms for solving problem instances that need to be pre-processed using a complete algorithm; in the context of SAT, complete solvers would be used to filter out unsatisfiable instances, and in the context of TSP, a complete solver would be used to determine provably optimal solution qualities.)

The predictions made in an extrapolation challenge will rarely be precisely confirmed by empirical observations. This give rise to the question how the quality of the predictions given corresponding observations (from Step 4) should be assessed. In principle, it is possible to use RMSE in this context, either based on the instance set J or based on the combined instance set $I \cup J$; the latter reflects the overall accuracy of the given model M, while the former focusses solely on the extrapolations obtained from M and provides therefore a more direct indication of M's ability to make meaningful predictions outside of its support.

The absolute value of the RMSE $R(J)$ obtained on instance set J is often not very meaningful, but can be useful as a basis for comparing different models. Furthermore, $R(J)$ can be meaningfully compared to $R(I)$, and in particular, to the residuals on the largest problem instances in I. Finally, even imprecise extrapolations (as, for example, obtained from biased models) can be useful, *e.g.*, in that they can provide empirical upper or lower bounds on scaling behaviour.

An example for challenging scaling models by extrapolation is shown in Figure 3 and Table 1. We note that, unlike in the examples illustrated in Figures 1 and 2, where mean run-time was considered, here, median run-time was studied. The empirical median has the advantage of being more robust with respect to outliers than the empirical mean and, more importantly, unlike the mean, the median is unaffected by

instance size	predicted median run-time: polynomial model	predicted median run-time: exponential model	predicted median run-time: root-exponential model	observed median run-time
2 000	2 709.80	4 446.53	3 393.90	3 400.82 (1000/1000)
2 500	6 302.61	21 020.46	10 651.96	8 855.28 (100/100)
3 000	12 561.47	99 371.86	29 957.63	30 024.49 (99/100)

Table 1: Predicted median run-times from polynomial and exponential fits for Concorde on RUE instances *vs* observed median run-times. Best polynomial fit ($n = 500..1500$, untransformed data): $8.83 \cdot 10^{-10} \cdot n^{3.78}$, RMSE= 15.38; best exponential fit ($n = 500..1500$, untransformed data): $8.90 \cdot 1.00311^{n}$, RMSE= 12.81; best root-exponential fit ($n = 500..1500$, untransformed data): $0.21 \cdot 1.24194^{\sqrt{n}}$, RMSE= 8.83; RMSE values are on the support of the respective model. The data for $n = 3000$ is based on run-time data for 99 out of 100 instances; the run on the remaining instance was terminated after several weeks of CPU time, but the median run-time has been calculated correctly over all 100 instances.

'censored' runs, which were aborted after reaching a high CPU time cutoff (in this case, more than one CPU day). Evidently, in this example, the root-exponential model produces substantially more accurate run-time predictions for larger instance sizes than the polynomial and exponential models. The close agreement between the prediction of the root-exponential model for $n = 3000$ is particularly impressive, considering that the support of the model only ranges from $n = 500$ to 1500.

An interesting generalisation of the method described earlier in this section can be used to assess whether the run-times measured on *individual problem instances* agrees with the predictions obtained by extrapolating from scaling models: In this case, scaling models are independently fitted on two quantiles of the SCDs (*i.e.*, the distributions of run-time over instances of the same size), Q_l and Q_h. Extrapolation of those models then yields predictions for the respective quantiles at larger (or smaller) instances sizes, If the models were accurate, we would expect a fraction $h - l$ of instances at each size to show observed run-times falling within the intervals formed by these predictions, and therefore, observed fractions close to the target of $h - l$ can be seen as evidence supporting the model. (Some deviations between the observed and target fraction of instances are likely to arise, especially when very small numbers of instances are tested in this manner.) When using this approach, care needs to be taken that the number of instances corresponding to the quantiles computed on the support for each instance size is sufficiently large to give reasonably stable estimates (the required number depends on the nature of the underlying distribution of run-times).

Alternatively, if a parametric model can be fitted to the SCDs at a given instance size, model fitting and extrapolation can be performed on the parameters of that SCD model. This allows the extrapolation of SCD models to instance sizes outside of the support, from which quantiles can then be determined analytically, and run-times observed on individual instances can be assessed against these. Formally, the Kolmogorov-Smirnov test can be used to investigate the hypothesis that a given set of observed run-times stems from the extrapolated SCD for a given instance size. As a special case, for a single instance of size n with run-time t, the value r for which the quantile Q_r of the extrapolated SCD for size n is equal to t can be used to determine a p-value for the statistical hypothesis test that t does indeed stem from the predicted SCD. A challenge instance (or set of instances) that pass this test provide support for the combination of the model predicting SCDs as a function of instance size.

6 Validation using Bootstrap Confidence Intervals

Typically, the run-time of a given algorithm varies between instances of the same size, and even for distributions of syntactically very similar problem instances (such as Uniform-Random-3-SAT instances with a fixed number of clauses and variables) extreme variations have been observed (see, *e.g.*, Hoos and Stützle, 1998). As previously mentioned, in such cases, models for the scaling of run-time with instance size typically capture dscriptive statistics such as mean or median run-time. Clearly, any such model M depends on the on actual set of instances I for which run-time measurements were performed and subsequently used for fitting the model, and for different sets I', different models M' are be obtained. This raises the question to which extent predictions obtained from a model M reflect particularities in the underlying instance set I rather than the true scaling behaviour of the given algorithm.

To investigate this question, a statistical method known as *bootstrap analysis* can be used (see, *e.g.*, Efron and Tibshirani, 1993). The key idea underlying this method is to repeatedly resample a given set S of empirical data in order to obtain a collection of samples (where each sample has the same size as S and corresponds to a subset of S, with some elements repeated), which can then be used as a basis for analysing properties of a statistical estimator (such as its variance) or for performing statistical hypothesis testing. In the context considered here, bootstrap analysis can be used to obtain confidence intervals for the predictions generated by a given scaling model M; based on such confidence intervals, it is then possible to determine to which degree observed run-times are statistically supported by the model, and hence to assess the validity of the model in terms of its compatibility with those data.

To perform this type of analysis, we propose the following basic procedure:

1. collect run-time data $B(I)$ for a set of problem instances I (see Section 2) such that for each problem size in I run-time data for m instances are available;
2. for each problem size n_i in I, independently draw r samples $I_{i,1}, \ldots, I_{i,r}$, where each $I_{i,j}$ consists of m instances and is determined by independent uniform random sampling *with replacement* from the full set of size-i instances in I;
3. for each series of instance sets $I_{1,j}, \ldots, I_{k,j}$, where $n_1, \ldots, n_k$ are the instances sizes represented in I, fit a parametric scaling model M_j to the corresponding observed run-times $B(I_{1,j}), \ldots, B(I_{k,j})$ (see Section 4);
4. for a given instance size n, determine a series of predictions $P := (P_1(n), \ldots, P_r(n))$, where $P_j(n)$ is the prediction obtained from model M_j;
5. from the series of predictions P, determine the *bootstrap percentile confidence interval* for a given confidence level α as ***CI*** $:= [Q_{(0.5-\alpha/2)}, Q_{(0.5+\alpha/2)}]$, where Q_x denotes the x-quantile of the empirical distribution of the values in P (typically, $\alpha = 0.95$ is used in this context);
6. determine the actual run-time $B(n)$ by running the given algorithm on a set of problem instances of size n;
7. if $B(n) \in$ ***CI***, we say that the observed data $B(n)$ is in agreement with the given parametric scaling model, otherwise we say that $B(n)$ is in disagreement with the model (at confidence level α).

In Step 2, the run-time data used for building the model is resampled, motivated by the idea that the empirical distribution of the run-time data for a given instance size can be used as a proxy for the underlying true solution cost distribution. Obviously, sampling has to be performed *with replacement*, since otherwise the samples $I_{i,j}$ would be identical to the original set of data for instance size n_i.

In most bootstrap analyses, obtaining and processing a sample from the original data is computationally cheap; therefore, typically thousands of samples are taken. In the context of our procedure, we need to fit a model for each of the r samples, which can be costly and difficult. This not only limits the number of samples, r, that can be processed within reasonable time, but also requires a fully automated and reasonably reliable method for model fitting. As explained in Section 4, depending on the complexity of the scaling model, the numerical methods used for model fitting are not always guaranteed to produce a good fit; therefore, care should be taken to ensure that all models produced in Step 3 are fitted correctly (*e.g.*, by examining the models $M_1, \ldots, M_r$ for suspiciously poor fits to the respective underlying data, based on RMSE values or graphical inspection).

The differences between the models obtained in Step 4, and therefore also between the predictions obtained from these models in Step 5, intuitively reflect the impact of variation in the run-time data that can be considered an artefact of working with particular samples of empirical run-time data (as well as with model fitting procedures that may not achieve truly optimal fits). The size of the confidence intervals obtained in Step 5 depends on several factors, including

- the size of the original instance set I and the number m of instances per size (small instance sets tend to lead to larger confidence intervals);
- the distance of n from the nearest problem sizes in I (if that distance is large, *e.g.*, in the case of an extrapolation challenge, the confidence interval tends to be large as well);
- the parametric model used in Step 3 (for example, exponential scaling models tend to produce larger confidence intervals than low-order polynomial models, as do models with more parameters compared to models with fewer parameters);
- the confidence level α used in Step 5 (obviously, smaller values of α lead to larger confidence intervals).

The confidence level is usually set to a standard value (*e.g.*, $\alpha = 0.95$) and cannot be varied without incurring substantial changes in the interpretation of the results, because it controls the acceptable probability of type 1 errors made in Step 7 of our procedure. As is always the case in statistical hypothesis testing, α should be chosen before running an experiment and must never be adapted *post-hoc* in order to achieve a desired result in Step 7 of our procedure. The parametric model used in a given scaling study is typically determined (or at least considerably constrained) by the context of the study – although it is worth reïterating that models with many parameters are often more difficult to fit and may be prone to producing poor predictions.

The first two factors, however, can be varied more freely by the experimenter. The first of those is primarily constrained by the computational resources available for the study, but otherwise should be chosen as generously as possible, by using a relatively large range of instances sizes and a large number of instances per problem size. As stated previously, the latter is especially important for the rather commonly encountered case of high variability in solver run-time across instances of the same size; typically, sets of least 50–100 instances per size should be used. The range and number of instances sizes considered is related to the range of instances sizes for which predictions are desired and the number of adjustable parameters in the model. As a rule of thumb, in order to prevent overfitting, the number of problem sizes considered should be at least twice as high as the number of problem parameters.

In the case of an extrapolation challenge to larger instance sizes, the extrapolation distance (*i.e.*, the second of the four factors listed above) aimed for in a study often depends on the nature of the scaling model. For fast-growing models, *i.e.*, super-polynomial functions or polynomials with high degree (in particular, higher than cubic), ratios ξ between the extrapolation distance and the range of instance sizes used for fitting the model between 0.5 and 1 would pose considerable challenges to a model; extrapolation to even larger ratios can be considered extreme challenges, and smaller ratios would only be modestly challenging. For slow-growing scaling models, *i.e.*, sub-polynomial functions or polynomials with small degree, the ratios ξ characterising moderate, considerable and extreme extrapolation challenges would be somewhat larger, to reflect the fact that small errors would not be expected to amplify as they do in the case of fast-growing models.

Clearly, obtaining agreement between an observation and a scaling model in Step 7 provides stronger support for the model if the confidence interval, for reasonably chosen α, is relatively small (since in that case there is a smaller risk of incorrectly failing to obtain disagreement, because of the model inaccuracy as reflected in the size of the confidence interval). Therefore, in order to be meaningful, modest extrapolation challenges should use large numbers of instances per instance size in order to achieve narrow confidence intervals.

The validation procedure outlined above is particularly useful in the context of two or more scaling models. In that case, it is desirable to choose I and m (and, in the case of an extrapolation challenge, the extrapolation distance) such that the confidence intervals obtained for the given models are disjoint. If this cannot be achieved, the overlap in the confidence intervals should be minimised. To avoid problematic bias in the

	predicted median run-time			
instance size	polynomial model	exponential model	root-exponential model	observed median run-time
2 000	[2 298.22 , 3 160.39]	[3 793.00 , 5 266.68]	**[2 854.21 , 3 977.55]**	3 400.82 (1000/1000)
2 500	[4 987.78 , 7 870.81]	[16 378.03 , 28 010.39]	**[8 266.46 , 13 601.02]**	8 855.28 (100/100)
3 000	[9 430.35 , 16 615.93]	[70 584.38 , 147 716.74]	**[21 549.28 , 41 271.35]**	30 024.49 (99/100)

Table 2: Bootstrap confidence intervals for $\alpha = 0.95$ for median run-time predictions for Concorde on RUE instances. The support data for all three models is based on 1000 instances for each problem size, $n = 500..1500$; for 1000 bootstrap replicates with 1000 data points per problem size each, polynomial, root-exponential and exponential fits were performed using the Levenberg-Marquardt Algorithm (as implemented in the Gnuplot 'fit' function) on the untransformed instance size and run-time data, resulting in 1000 predictions per instance size for each model. The $\alpha = 0.95$ bootstrap confidence intervals for the model parameters are $[8.69 \cdot 10^{-11}, 8.44 \cdot 10^{-9}]$, $[3.46, 4.11]$ for the polynomial model, $[6.71, 11.21]$, $[1.00292, 1.00334]$ for the exponential model and $[0.115, 0.373]$, $[1.2212, 1.2630]$ for the root-exponential model. Observed run-time data for $n \geq 2000$ are identical to those in Table 1.

experimental protocol, confidence intervals need to be determined before Steps 6 and 7 of our procedure are executed, *i.e.*, before the observations used for assessing the given models are made.

Table 2 and Figure 4 show the bootstrap confidence intervals obtained by applying our procedure to the same TSP data and model families considered in Section 5. As could be expected, the width of the confidence intervals increases with the distance of the instance size considered from the upper boundary of the support. Nevertheless, the confidence intervals for the three models quickly become disjoint, and only the ones for the root-exponential model contain the observed median run-times for the three challenge instance sizes – an observation that strongly supports this particular model of scaling for the median run-time of the Concorde solver on the RUE TSP instances studied here.

We note that the bootstrap analysis used here for obtaining confidence intervals on model predictions can be used analogously to determine confidence intervals for model parameters and thus to assess how accurately the parameter values of a given model are determined by the observations based on which the model has been built.

7 Discussion and Related Work

Investigations of the scaling of empirical run-time (and other performance measures) with problem instance size are becoming increasingly popular in various areas of computing science. Such studies are pursued not only by researchers interested primarily in solving particular application problems as efficiently as possible, but also by theoreticians (see, *e.g.*, Sanders and Fleischer, 2001).

Several earlier empirical scaling studies can be found in the literature on artificial intelligence. Gent and Walsh (1993) investigated the scaling of several high-performance stochastic local search procedure for SAT on satisfiable instances from the phase transition region of Uniform Random-3-SAT. Similar studies have been reported by Parkes and Walser (1996), Gent et al. (1997) and Hoos and Stützle (2000), who also fitted various parametric scaling models to the observed run-time data, using the approach discussed in Section 4. Gent et al. (1997) challenged their models by interpolation and extrapolation to instance sizes modestly smaller and larger than those used for fitting the models; while they provide anecdotal evidence for the accuracy of their fits and predictions, they did not use any formalised methods for assessing their models. Interestingly, the scaling results from all of these studies are somewhat inconclusive and may well be worth revisiting, using more recent algorithms for the problems studied (SAT and asymmetric TSP) and the methodology described in this work (particularly in Section 6).

More recent examples of graphical scaling analyses can be found in work by Subramani and Desovski (2005) on algorithms for the negative cost cycle detection problem, and work by Kunkle (2002) on algo-

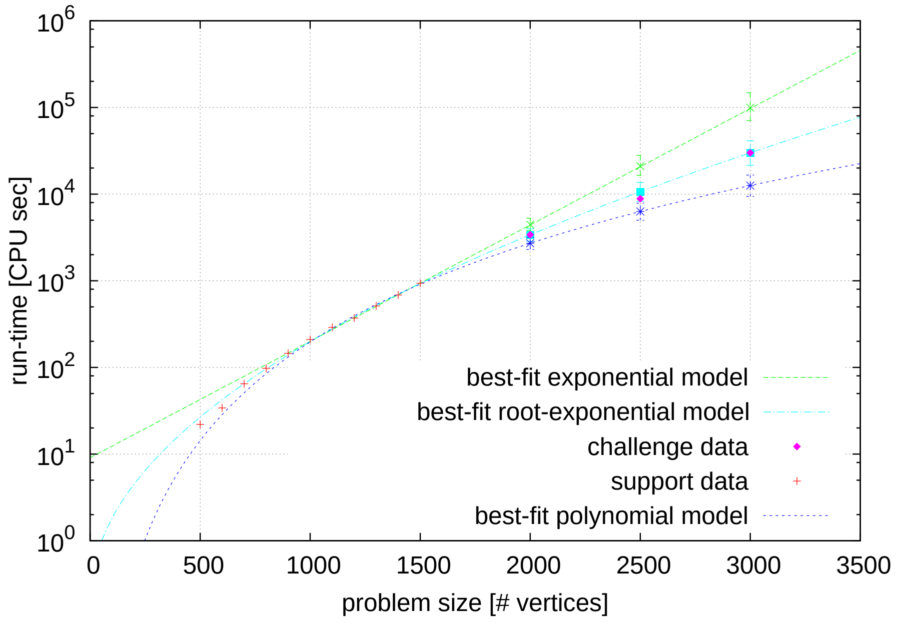

Figure 4: Scaling of median run-time required by the Concorde solver for solving RUE TSP instances; the data and best-fit exponential, root-exponential and polynomial scaling models shown are the same as in Figure 3. The 95-percentile bootstrap confidence intervals indicated for $n \geq 2000$ are the same as in Table 2.

rithms for longest common subsequence algorithms (this latter study also uses fitting of a single-parameter model to the observed run-time data).

Aguirre-Hernández et al. (2007) used graphical analysis and model fitting techniques for analysis the run-time scaling of several algorithms for RNA secondary structure design. To our knowledge, theirs is the first study to contrast the scaling of the run-times observed for real-world problem instances with run-time percentiles for a distribution of problem instances that were generated by means of a random generator designed to produce realistic instances.

Goldsmith et al. (2007) developed a tool called *Trend Profiler* for determining how many times a set of basic block in a given program will be executed as a function of given features of the input; besides instance size, arbitrary other features can be considered. The tool builds simple linear and power-law models using linear regression (in the latter case on log-transformed data); it also uses clustering of these models to better deal with a large number of basic blocks (and hence execution counts). The quality of these models is assessed graphically by means of scatter plots relating input feature values to observed and predicted execution counts, and to residuals (*i.e.*, differences between observed and predicted counts). Additionally, R^2 measures are used to quantify model accuracy (here, R^2, also known as the coefficient of determination, is the square of Pearson's correlation coefficient). Interestingly, *Trend Profiler* also determines bootstrap percentile confidence intervals for predicting the performance on extrapolated input feature values (using essentially the same method as we do in Section 6), but does not provide direct support for validating models based on those extrapolation challenges.

Goldsmith et al. (2007) have used *Trend Profiler* to analyse several pieces of real-world software, including the widely used *bzip* file compressor; all examples they considered show slow-growth scaling of run-time (polynomial in instance size with degree smaller than two). Their use of linear regression allows them to circumvent the potential difficulties arising in the context of more general function-fitting techniques at the price of being restricted to optimising the fits by means of RMSE minimisation on transformed data (those fits can, and often do, differ significantly from those obtained by minimising errors on the untransformed data and may lead to worse predictions, particularly for fast-growing scaling functions currently not supported by *Trend Profiler*).

Sanders and Fleischer (2001) discuss various roles empirical run-time data can play in the context of obtaining, assessing and sharpening hypotheses about the scaling of a given algorithm's performance with instance size and illustrate these with several examples. Two of these examples involve the scaling of run-time and use graphical analysis (and in one case, a standard t-test) to assess given scaling hypotheses.

McGeoch et al. (1997) and McGeoch et al. (2002) study the use of heuristic techniques for finding functions that provide empirical bounds for the scaling of an algorithm's run-time with instance size. The various bounding procedures were designed to work with polynomial scaling functions; they were experimentally evaluated on artificial and real scaling data designed or known to be bounded by low-order polynomial functions (of degree at most 2). One of the most successful strategies investigated by McGeoch et al. is based on linear regression (a simple model fitting technique) on log-log transformed data; this is conceptually closely related to the use of log-log plots for the graphical detection of polynomial scaling (see Section 3). At the same time, they reported that attempts to fit scaling models using non-linear regression (resulting in minimum RMSE fits) produced poor results (see McGeoch et al., 2002, Section 5.4.6). We believe that the reason for this obvservation might lie in their focus on slow-growing scaling functions with lower-order terms that can render accurate model fitting difficult, as well as in the inadequacy of RMSE-minimisation as an objective in the context of curve bounding (rather than curve fitting).[4]

It would be interesting to rigorously evaluate the methodology presented in this report on data from slow-growing scaling functions, such as the ones considered by McGeoch et al. (1997), and furthermore to extend our methodology to deal with the problem of bounding asymptotic growth rather than with that of modelling and estimating run-time on which we focussed in this report.

The bootstrap analysis method presented in Section 6 is based on the standard non-parametric bootstrap by Efron (see, *e.g.*, Efron and Tibshirani, 1993) with percentile confidence intervals. There are several alternative resampling methods that draw smaller samples (with or without replacement), as well as more sophisticated methods for determining confidence intervals (see, *e.g.*, Davison et al., 2003). While in principle, such more advanced method should be applicable to the problem of validating scaling models (by adapting Steps 2 and 5 in the procedure from Section 6 accordingly), the question which methods yield the best results under which circumstances appears to be still under investigation (using a combination of theoretical and empirical approaches).

In our own ongoing work on the scaling of high-performance TSP algorithms (which is partially based on the data used in the examples used for illustrative purposes in this report) we have recently found evidence that by using the standard bootstrap confidence intervals described in Section 6 we can distinguish models that fare well when subjected to extreme extrapolation challenges from those who do not (Hoos and Stützle, 2009). We suspect that similar results can be achieved for algorithms whose empirical run-times are characterised by slow-growing functions of instance size.

Finally, we note that the methodology outlined in this report for analysing the empirical scaling of run-time with instance size can be applied analogously to performance measures different from run-time, such as memory consumption or amount of communication, and to instance characteristics different from instance size, such as the density of a given graph or the empirical entropy of a collection of items to be sorted.

In future work, we are planning to develop tools to support empirical scaling analyses as outlined in this report. While similar in functionality to the recent *Trend Profiler* tool (Goldsmith et al., 2007), we intend to provide support for a large family of non-linear scaling models, including models based on super-polynomial functions, as well as for automated extrapolation challenges (using parameterised random instance generators). We are also planning to develop tools to support the construction of parameterised random problem instance generators based on features of collections of real-world instance, and the use of such generators for obtaining empirical scaling results that meaningfully relate to the run-times observed for real-world instances. We anticipate to integrate these tools into the *High-performance Algorithm Laboratory (HAL)* system outlined by Hoos (2008).

[4]This latter hypothesis was also supported by Catherine McGeoch in personal communication.

Acknowledgements.

We thank Jonathan L. Shapiro (University of Manchester) for useful discussion of ideas underlying the bootstrapping method used for assessing the scaling models during the 2008 Dagstuhl Seminar on Theory of Evolutionary Algorithms and Thomas Stützle for additional comments. We are also grateful for clarifying comments received from Catherine McGeoch regarding details of her work on curve bounding.

The run-time data for the Concord TSP Solver that has been used to exemplify our approach has been generated in a joint project with Thomas Stützle, whose results are discussed and analysed in more detail in TR-2009-17 (Hoos and Stützle, 2009).

References

Aguirre-Hernández, R., Hoos, H. H., and Condon, A. (2007). Computational RNA secondary structure design: Empirical complexity and improved methods. *BMC Bioinformatics*, 8:34.

Ahuja, R. K. and Orlin, J. B. (1996). Use of Representative Operation Counts in Computational Testing of Algorithms. *INFORMS Journal on Computing*, 8(3):318–330.

Andronescu, M., Fejes, A. P., Hutter, F., Hoos, H. H., and Condon, A. (2004). A new algorithm for RNA secondary structure design. *Journal of Molecular Biology*, 336(3):607–624.

Applegate, D., Bixby, R., Chvatal, V., and Cook, W. (2006). *The Traveling Salesman Problem: A Computational Study*. Princeton University Press.

Bentley, J. L. and McIlroy, M. D. (1993). Engineering a sort function. *Software–Practice and Experience*, 23:1249–1265.

Biggar, P., Nash, N., Williams, K., and Gregg, D. (2008). An experimental study of sorting and branch prediction. *J. Exp. Algorithmics*, 12:1–39.

Birattari, M. (2004). On the estimation of the expected performance of a metaheuristic on a class of instances. Technical Report TR/IRIDIA/2004-01, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.

Davison, A. C., Hinkley, D. V., and Young, G. A. (2003). Recent developments in bootstrap methodology. *Statistical Science*, 18:141–157.

Efron, B. and Tibshirani, R. J. (1993). *An Introduction to the Bootstrap*. Chapman & Hall, New York.

Gent, I., Macintyre, E., Prosser, P., and Walsh, T. (1997). The scaling of search cost. In *Proceedings of the American Association of Artificial Intelligence, AAAI-97*, pages 315–320. MIT Press.

Gent, I. P. and Walsh, T. (1993). Towards an understanding of hill–climbing procedures for SAT. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 28–33. AAAI Press / The MIT Press, Menlo Park, CA, USA.

Goldsmith, S. F., Aiken, A. S., and Wilkerson, D. S. (2007). Measuring empirical computational complexity. In *ESEC-FSE '07: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 395–404, New York, NY, USA. ACM.

Gomes, C. P., Selman, B., and Crato, N. (1997). Heavy-tailed distributions in combinatorial search. In *Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming (CP97)*, volume 1330 of *Lecture Notes in Computer Science*, pages 121–135. Springer.

Gomes, C. P., Selman, B., Crato, N., and Kautz, H. (2000). Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. Autom. Reason.*, 24(1-2):67–100.

Hoos, H. and Stützle, T. (2000). Local search algorithms for SAT: An empirical evaluation. *J. Automated Reasoning*, 24:421–481.

Hoos, H. H. (2008). Computer-aided design of high-performance algorithms. Technical Report TR-2008-16, University of British Columbia, Department of Computer Science.

Hoos, H. H. and Stützle, T. (1998). Evaluating Las Vegas algorithms — pitfalls and remedies. In Cooper, G. F. and Moral, S., editors, *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 238–245. Morgan Kaufmann Publishers, San Francisco, CA, USA.

Hoos, H. H. and Stützle, T. (2004). *Stochastic Local Search—Foundations and Applications*. Morgan Kaufmann Publishers, USA.

Hoos, H. H. and Stützle, T. (2009). On the empirical scaling of run-time for finding optimal solutions to the traveling salesman problem - part 1: Concorde on rue and tsplib instances. Technical Report TR-2009-17, University of British Columbia.

Johnson, D. S. and McGeoch, L. A. (2002). Experimental analysis of heuristics for the STSP. In Gutin, G. and Punnen, A., editors, *The Traveling Salesman Problem and its Variations*, pages 369–443. Kluwer Academic Publishers.

Kunkle, D. (2002). Empirical complexities of longest common subsequence algorithms. Technical report, Rochester Institute of Technology, Computer Science Department, Rochester, NY, USA.

Leyton-Brown, K., Pearson, M., and Shoham, Y. (2000). Towards a universal test suite for combinatorial auction algorithms. In *ACM Conference on Electronic Commerce*, pages 66–76.

Li, X., Garzarán, M. J., and Padua, D. (2004). A dynamically tuned sorting library. In *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization*, CGO '04, pages 111–122, Washington, DC, USA. IEEE Computer Society.

McGeoch, C., Sanders, P., Fleischer, R., Cohen, P. R., and Precup, D. (2002). Using finite experiments to study asymptotic performance. pages 93–126.

McGeoch, C. C., Precup, D., and Cohen, P. R. (1997). How to find big-oh in your data set (and how not to). In *IDA '97: Proceedings of the Second International Symposium on Advances in Intelligent Data Analysis, Reasoning about Data*, pages 41–52, London, UK. Springer-Verlag.

Parkes, A. J. and Walser, J. P. (1996). Tuning local search for satisfiability testing. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, volume 1, pages 356–362. AAAI Press / The MIT Press, Menlo Park, CA, USA.

Sanders, P. and Fleischer, R. (2001). Asymptotic complexity from experiments? a case study for randomized algorithms. In *WAE '00: Proceedings of the 4th International Workshop on Algorithm Engineering*, pages 135–146, London, UK. Springer-Verlag.

Subramani, K. and Desovski, D. (2005). On the empirical efficiency of the vertex contraction algorithm for detecting negative cost cyles in networks. In *Computational Science ICCS 2005*, volume 3514 of *LNCS*, pages 180–187. Springer Berlin / Heidelberg.