Analysing the Empirical Time Complexity

of High-performance Algorithms for SAT and TSP

by

Zongxu Mu

B.Sc. (Hons), City University of Hong Kong, 2013

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate and Postdoctoral Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

November 2015

© Zongxu Mu 2015

Abstract

The time complexity of problems and algorithms, *i.e.*, the scaling of the time required for solving a problem instance as a function of instance size, is of key interest in theoretical computer science and practical applications. In this context, the propositional satisfiability (SAT) and the travelling salesperson (TSP) are two of the most intensely studied problems, and it is generally believed that solving SAT or TSP requires exponential time in the worst case. In this work, we refine and extend a recent empirical scaling analysis approach and study the empirical scaling of the running times of several prominent, high-performance SAT and TSP algorithms. For SAT, we focus on random 3-SAT instances from the phase transition region, arguably the most prominent model for difficult SAT instances, and obtain interesting and surprising scaling results for both SLS- and DPLL-based solvers. Particularly, we find solid support for polynomial scaling for SLS-based solvers on phase-transition random 3-SAT instances. We also show that DPLL-based solvers scale exponentially and are faster by only a constant factor in solving satisfiable instances compared to unsatisfiable instances. We further report empirical scaling results for two classes of random 4-SAT instances to gain additional insights into the performance of state-of-the-art SAT solvers. For TSP, we concentrate on two-dimensional random uniform Euclidean (RUE) instances, and characterise the scaling of running time for complete and incomplete algorithms for finding optimal solutions. Our results indicate that the scaling of all these algorithms is consistent with or bounded from above by root-exponential models of the form $a \cdot b^{\sqrt{n}}$. We also explored the impact of automated algorithm configuration on the scaling of these algorithms. Since our approach is applicable beyond SAT and TSP, to enable its broad use, we designed Empirical Scaling Analyser (ESA), an automated tool that can be conveniently used to study empirical scaling of many types of algorithms. In particular, ESA presents scaling analysis results in the form of automatically generated, detailed technical reports. Many results reported in this thesis, including most tables and figures, were automatically generated by ESA and are only slightly modified to fit here.

Preface

This thesis covers several publications as well as manuscripts not yet published, which I co-authored with others.

A major part of the SAT-related work is published in [52], where I performed all experiments, including those for collection of running time data, modelling of phase transition, and analysis of scaling behaviours of the solvers. I also wrote the first draft of the paper, including preparation of the tables and the figures.

Work on the scaling of TSP solvers has been prepared for publication (with a complete draft finished in parallel with this thesis) [54]. I performed scaling analysis on the running time data collected by Dubois-Lacoste and Stützle, and wrote the first draft of the paper. For work on impact of automated configuration of TSP solvers, I ran the configuration experiments, collected running time data and performed the scaling analysis. Thomas Stützle helped prepare the parameter files of the solvers, and answered my questions regarding these parameters.

I was heavily involved in the conceptual design of ESA [53], and was responsible for the development of its code and website.

Hereafter, I adopt the plural subject "we" in recognition of the collaboration with Prof. Hoos and others.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to start by thanking Prof. Holger Hoos, my supervisor, for his capable supervision, enduring guidance and unceasing encouragement during the pursuit of my Master's degree in Computer Science. With both a vision of the big picture and a focus in details, he kept me on track by reminding me of the scientific questions that we are investigating, encouraged me to work harder by pointing out the impact that my work can have, and was always ready to dive into the data and analyse the problems. Prof. Hoos is extremely knowledgeable, and always encouraged me to formulate my own ideas and test them. He also shared with me his thoughts and comments on scientific writing, presentation and communication.

My collaborators at Université Libre de Bruxelles, Thomas Stützle and Jérémie Dubois-Lacoste, were extremely supportive. They have answered numerous questions from me on the TSP solvers and on the running time data and helped organise our data in a better way.

For work on empirical time complexity of phase-transition random 3-SAT, I thank the anonymous reviewers of our IJCAI paper for their thoughtful comments and suggestions. I also thank Dimitris Achlioptas, Paul Beame, Henry Kautz, Donald Knuth, Bart Selman and Moshe Vardi for valuable advice regarding previous work on the 3-SAT phase transition and empirical performance of SAT algorithms, as well as Alfredo Braunstein and Riccardo Zecchina for discussions related to the survey propagation solver.

I also owe my gratitude to members of the β-lab in the CS department: Prof. Kevin Leyton-Brown, Chris Cameron, Chris Fawcett, Alexandre Fréchette, Julieta Martinez, Steve Ramage, Shu Yang and others, for the insightful discussions and valuable support.

Lastly but not the least, I want to thank my fellow classmates in the department, especially Rui Ge, Yidan Liu, Yifan Peng, Michael M.A. Wu, and Ben Zhu, for making my graduate school an unforgettable experience, and my family and friends for their unconditioned support and encouragement.

Dedication

To my parents, Yutong Zhao and Shanwen Mu.

Chapter 1

Introduction

In theoretical computer science, time complexity is arguably the most important aspect for analysing and understanding problems and algorithms. Time complexity of an algorithm is the scaling of the time required for solving a problem instance as a function of instance size. Traditionally studied via rigorous mathematical methods, complexity results are usually studied and understood via a family of functions of instance size, which describe the asymptotic scaling of operation counts using notations such as $\mathbf{O}(n)$ or $\Theta(n)$. Essentially, such results are compact descriptors of the performance of algorithms and form the basis of the hierarchy of complexity classes in theoretical computer science. They also convey certain information regarding the feasibility of using a given algorithm to solve a problem instance of certain size.

In spite of the significant role that theoretical methods play in understanding complexity of problems and algorithms, many high-performance algorithms are beyond the reach of such methods due to the complexity of the algorithms. Yet, these algorithms often represent the state of the art for solving problems of wide interest in practice. Thus, empirical analysis arises as the alternative for studying the time complexity of these algorithms. In such studies, the propositional satisfiability problem (SAT) and the travelling salesperson problem (TSP) are two important problems that have attracted sustained academic interest and have seen many practical applications. In this thesis, we extended a recent empirical scaling analysis approach [33], which emphasises the idea of challenging by extrapolation and uses bootstrap re-sampling to statistically assess obtained models. We also applied the methodology to studying the empirical time complexity of several high-performance algorithms for SAT and TSP.

1.1 Empirical Analysis of Algorithms

Theoretical methods usually ignore or idealise low-level details of algorithm implementations and execution environments. Moreover, they are typically obtained for extreme or average cases and for simpler variants of algorithms. Empirical analysis of algorithms has seen increasing interest, because it permits predicting the

running times of high-performance algorithms in practice and may also provide useful insights into their behaviours. This is especially true for $\mathcal{NP}$-hard problems, where worst-case theoretical analysis typically does not offer much in understanding, evaluating and designing high-performance algorithms. Many of these problems have important applications in practice, for which high-performance algorithms have been designed. Examples of such problems include SAT and TSP, which we concentrate on in this work, and many others as well. In this context, empirical analysis is often the only way to understand the relevant performance differences that we observe in practice. Hoos and Stützle [34], for instance, conducted a systematic empirical analysis to evaluate a number of local search algorithms for SAT. Their results drew a comprehensive picture on the performance of these algorithms, and, for some of them, revealed some fundamental weaknesses in their design. Empirical analysis is also applicable to tractable or polynomial-time solvable problems, where empirical results often complement results obtained from worst- and average-case theoretical analysis. Such analysis can help choose or configure an algorithm for a given problem instance or situation, as demonstrated in several studies of sorting algorithms [9, 45, 10].

Empirical analysis recently witnessed progress in studying the scaling or time complexity of algorithms. For such analysis, the typical approach is to collect running time data for problem instances of various sizes, use a representative statistic such as the mean to characterise the running time for each size, and then derive a parametric function that fits or bounds the statistics. In other words, such analysis focus on obtaining a good model that describes the functional dependency of running time on instance size. The process usually involves comparing, based on goodness of the fit or a quantitative error measure, one parametric function with another to determine which describes the data better. Figure 1.1 (taken from [59]) illustrates a typical scenario of such a process, as done by Parkes and Walser [59] when studying the scaling behaviour of WalkSAT/SKC at phase transition.

1.2 The Propositional Satisfiability Problem

The propositional satisfiability problem (SAT) is a conceptually simple decision problem. Given a Boolean formula, typically in conjunctive normal form, the problem asks whether there exists an interpretation, *i.e.*, a mapping from propositional variables to truth values, that satisfies the formula. The following is an example of a Boolean formula:

$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4),$$

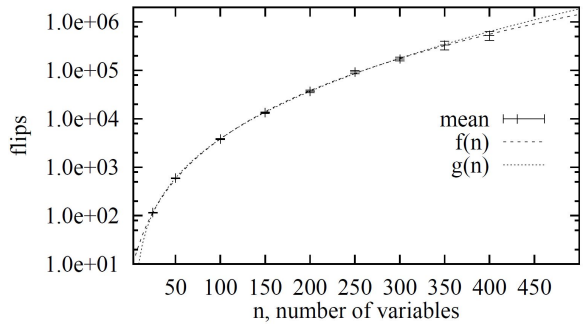


Figure 1.1: Typical scenario for the process of empirical scaling analysis (figure taken from [59]). Here, Parkes and Walser studied the empirical scaling of number of flips taken by WalkSAT/SKC to solve random 3-SAT instances at phase transition. They fitted two functions, $f(n) = f_1 \cdot n^{f_2 + f_3 \log(n)}$ and $g(n) = g_1 \cdot \exp\left(n^{g_2} \cdot (1 + g_3/n)\right)$, to the mean number of flips.

which can be satisfied, for instance, by setting all Boolean variables to true. SAT was the first problem proven to be $\mathcal{NP}$-complete [16], and no known algorithm solves the problem efficiently in the sense of better-than-exponential scaling of running time with instance size in the worst case. SAT also has a broad range of practical applications, and despite its discouraging worst-case time complexity, many SAT solvers have been constructed that perform well on theoretically and practically interesting sets of benchmark instances. The performance of SAT solvers is regularly assessed in SAT competitions [62] , which feature benchmark instances from various applications, instances that are hand-crafted to challenge solvers, and randomly generated instances.

Following seminal work by Cheeseman et al. [13] and Mitchell et al. [51], randomly generated 3-SAT instances at the so-called solubility phase transition, where 50% of the instances generated at a given size are satisfiable, have been widely studied; to this day, they are regarded as a model for the difficulty of the $\mathcal{NP}$-complete 3-SAT problem and represent one of the most prominent benchmarks for SAT solvers. It is conjectured, yet unproven, that the location of the solubility phase transition for uniform random 3-SAT, in terms of the ratio of the number of clauses and variables, m/n, converges towards a limiting value as n approaches infinity. Crawford and Auton [17] studied the location of the phase transition for random 3-SAT as a function of n and provided a widely used formula, based on empirical data for $n = 20 \ldots 300$.

In this thesis, we consider the question how the performance of high-performance SAT solvers on uniform random 3-SAT at the solubility phase transition scale with

n. Specifically, prompted in part by earlier, inconclusive results suggesting polynomial scaling of incomplete SAT solvers based on stochastic local search (SLS) [25, 59, 26], we address the problem of characterising the empirical scaling of the performance of prominent incomplete, SLS-based and complete, DPLL-based SAT solvers. SLS-based solvers start with a random interpretation and work by repeatedly flipping the value of a selected variable until a satisfiable interpretation is found. They usually implement heuristic strategies to select the variable to flip and often have mechanisms to restart with a new random assignment if no solution is found for too long. DPLL-based solvers explore the search space of variable assignment to find satisfiable interpretations following systematic backtracking procedures based on the Davis–Putnam–Logemann–Loveland algorithm [18, 19]. We are interested in determining whether there are qualitative differences in the scaling behaviour of SLS-based and DPLL-based solvers, in the scaling of DPLL-based solvers on satisfiable and unsatisfiable instances, and in the scaling of different solvers within the two major groups (SLS-based and DPLL-based). We also performed brief experiments on 4-SAT to further understand the scaling performance of these solvers. Our experiments concerned phase-transition instances, as well as a distribution of instances that is less constrained but believed to be hard by theorists.

To avoid 'falling off' the phase transition, which could bias our scaling results towards easier instances, we model the location of the solubility phase transition point with extensive experiments. For 3-SAT, we re-examine the model in [17] for the location of the phase transition point, and derive a new model that agrees better with observed data for $n > 300$ and with recent results from statistical physics (see Sec. 5.2.1); for 4-SAT, we fit a model using the same approach.

1.3 The Travelling Salesperson Problem

The travelling salesperson problem (TSP) is a well known and widely studied $\mathcal{NP}$-hard combinatorial optimisation problem. Given a set of cities and their pairwise distances, the objective of TSP is to find the shortest round trip to visit each city exactly once. TSP has motivated sustained development of new algorithmic ideas in the domain of combinatorial optimisation. TSP algorithms are usually categorised into two kinds: exact algorithms, which guarantee to find an optimal solution of any TSP instance and can prove the optimality of the solution, and inexact algorithms, which may find optimal solutions but cannot prove optimality. Until today, Concorde [4] represents the long-standing state-of-the-art complete algorithm for solving TSP. For incomplete algorithms, LKH [29, 30] had been the best available solver until the introduction of EAX [57], which is an evolutionary

algorithm that improved the edge assembly crossover operator Nagata [56] that recombines short tours effectively. Empirical results in [57] show that EAX tends to perform better than LKH on a broad range of TSP instances, though LKH, as shown by Kotthoff et al. [41], is not dominated in that it is more efficient in solving a substantial proportion of the instances.

In the following, we focus on the finding times, namely the times required by state-of-the-art complete and incomplete solvers to find optimal solutions of a TSP problem without proving optimality, and we want to investigate how the running times of different solvers scale with instance size. Natural as the focus on finding times is for incomplete solvers, it may not seem straightforward why finding time should of interest for a complete solver. On one hand, we want to compare the scaling models of finding and overall running times of complete solvers to better understand them. On the other hand, we are also interested in comparing the scaling models of complete and incomplete solvers.

2D Euclidean instances, *i.e.*, instances where the locations to be visited correspond to points in the Euclidean plane, often occur in practical applications. One special type of such instances are so-called RUE instances, which can be generated by placing cities uniformly at random. Even though RUE instances typically do not occur in practice, they have similar properties as general 2D Euclidean instances and are widely studied in the literature. Following previous works on empirical scaling of TSP solvers [35, 21], we choose 2D RUE instances because they represent a distribution of TSP instances that is widely studied and can be easily generated.

1.4 Outline of the Thesis

What follows immediately in Chapter 2 is discussion of previous works on the empirical time complexity of SAT and TSP and on empirical methods for scaling analysis, which motivated much of this work. Then we review the methodology proposed by Hoos [33] and describe our improvements in Chapter 3. These improvements make extended use of the bootstrap intervals for statistical assessment of fitted models and introduce a novel way for comparison of scaling models and thus scaling performances of solvers. This is followed by Chapter 4 on empirical scaling analyser (ESA), the automated tool that we designed to perform scaling analysis automatically. ESA takes a file of running time data as input and can output a technical report of empirical scaling analysis results.

In Chapter 5, we present our results on the empirical scaling of running times of six SAT solvers on random 3-SAT, followed by a similar analysis for 4-SAT instances. Our main findings on 3-SAT are as follows:

- The median running times of the three prominent SLS-based solvers we studied, WalkSAT/SKC [64], BalancedZ [44] and probSAT [7, 6], scale polynomially (with an exponent of about 3), and the best exponential models are rejected with 95% confidence. Furthermore, we found no evidence that higher percentiles of the distributions of running times over sets of instances for fixed n may scale exponentially.

- The median running times of the three DPLL-based solvers we considered, kcnfs [20], march_hi [32] and march_br [31], exhibit exponential scaling (with a base of about 1.03), and the best polynomial models are rejected with 95% confidence.

- For all three DPLL-based solvers, the median running times when solving only satisfiable and only unsatisfiable instances, respectively, clearly exhibit exponential running time scaling, and the respective scaling models differ mainly by a constant factor.

- While the scaling models for the SLS-based solvers are very similar to each other, the two march-variants scale significantly better than kcnfs.

Afterwards, we investigate the time complexity of three TSP solvers in Chapter 6. For both complete and incomplete solvers, we focus on times required to find optimal solutions (without proving optimality), and compare the scaling models of these solvers. Our major conclusions are:

- The median finding times of Concorde, the state-of-the-art complete TSP solver, are consistent with a root-exponential model, *i.e.*, a model of the form $a \cdot b^{\sqrt{n}}$ (with a base of about 1.25), and the best exponential model and polynomial model can be rejected with 95% confidence.

- The time Concorde takes for finding an optimal solution scales slightly worse than the overall running time, but more detailed models with lower-order terms may be required to better capture this difference.

- For the two prominent incomplete solvers we studied, the median running times of EAX are consistent with a root-exponential model (with a base of about 1.12), and the best exponential and polynomial models can be rejected with 95% confidence; the median running times of LKH are bound by a polynomial (with an exponent of about 2.9) and a root-exponential model (with a base of about 1.19), and the best exponential model can be rejected with 95% confidence.

- Both incomplete solvers scale significantly better than Concorde, with b in the root-exponential model for Concorde (≈ 1.25) significantly larger than b in those for EAX (≈ 1.12) and LKH (≈ 1.19).

We also assess the impact of automated configuration on the scaling of incomplete TSP solvers in Chapter 6. For EAX, we showed that automated algorithm configuration and adapting population size with instance size can significantly improve the scaling behaviour; while for LKH, we observed overfitting during automated configuration in that it improved the performance of LKH for smaller instances but caused it to suffer for larger ones.

Finally, we conclude the thesis in Chapter 7 and discuss potential areas for future work.

Chapter 2

Related Work

In this chapter, we briefly survey related work on SAT and TSP as well as on empirical scaling analysis. For work on empirical time complexity of SAT and TSP, we discuss their limitations and point out gaps that we will later fill with additional analysis. For work on empirical scaling analysis, we discuss various methods and applications found in the literature, including the methodology that our work directly builds on.

2.1 Related Work on Random $k-$SAT

SAT is one of the most intensely studied problems in the computer science literature and beyond. The k-SAT problem is arguably the most widely studied class of SAT. Given a Boolean formula in conjunctive normal form for which every clause consists of exactly k literals, the problem asks whether there exists an interpretation that satisfies the formula. 3-SAT, the special case for $k = 3$, is arguably the most prominent $\mathscr{NP}$-complete decision problem [16]. Interest in phase transition phenomena in combinatorial problems and in uniform random 3-SAT specifically rose sharply when Cheeseman et al. [13] demonstrated that the hardest instances are found around a critical value of an order parameter, where a transition from predominantly soluble to mostly insoluble instances occurs. Uniform random k-SAT instances are generated by constructing uniformly and independently at random m clauses, each of which is obtained by sampling, again uniformly and independently at random, 3 of n variables, and negating each of them with probability 1/2 [51] (duplicate clauses are eliminated); the order parameter is the clauses/variable ratio, m/n. It is believed, but not yet proven, that for $k \geq 3$, the location of the phase transition point of uniform random k-SAT converges to a fixed threshold value as n approaches infinity. Assuming threshold values exist for small k, an accurate theoretical upper bound was proven by Franco and Paull [23] and was later improved to $2^k \log 2 - (1 + \log 2)/2 + o_k(1)$ by Kirousis et al. [40]. Achlioptas and Peres [1] achieved a major breakthrough in proving a lower bound, which was recently improved to $2^k \log 2 - (1 + \log 2)/2 - o_k(1)$ by Coja-Oghlan [15].

In a prominent study on the empirical difficulty of SAT instances, Mitchell

et al. [51] demonstrated that instances drawn from the phase transition region of uniform random 3-SAT instances tend to be the most difficult for a simple DPLL solver. Similar results were shown by Yokoo [68] for an SLS-based solver on the satisfiable phase of uniform random 3-SAT. Crawford and Auton [17] studied the phase transition region of uniform random 3-SAT empirically, developed a model for the location of the phase transition point and presented additional evidence for the difficulty of the SAT instances found there.

Gent and Walsh [25] studied the empirical behaviour of GSAT, one of the earliest and most prominent SLS-based SAT solvers [63], and its two variants, DSAT and HSAT. They noted that the scaling of the average number of variable flips required by these solvers for solving phase transition random 3-SAT instances was consistent with less-than-linear exponential scaling with n and did not rule out a polynomial scaling model with a degree of about 3. Later, Parkes and Walser [59] presented empirical evidence that the scaling of the average number of flips required by a more recent, prominent SLS-based SAT solver, WalkSAT/SKC [64], on the same class of instances might scale either as a power function with a slowly growing exponent, or as a sub-exponential function. Furthermore, Gent et al. [26] found that the 90th percentile of the number of flips of GSAT for random 3-SAT appears to grow no faster than n^4. However, in all cases, performance was measured in variable flips (which become more expensive as n increases) and based on limited curve fitting only, with a vaguely defined notion of a 'good fit'.

Coarfa et al. [14] used simple curve fitting to study the empirical scaling of median running time for three complete solvers on random 3-SAT and observed exponential scaling above certain solver-dependent density thresholds.

In contrast, in the following, we consider the actual scaling of running time and use a considerably more advanced and statistically sound approach for assessing scaling models. Unlike these earlier studies, we also challenge our scaling models by assessing their predictions on larger instances sizes than those used for fitting the models.

2.2 Related Work on TSP

The computation complexity of TSP has been intensely studied. It is $\mathscr{NP}$-hard to solve both the general TSP [24] and the special case of 2D Euclidean instances [58]. For Euclidean distances, in spite of known polynomial approximation schemes, it takes exponential time to find good solutions as the gap to optimality decreases [5].

Much less work has been done on investigating the empirical scaling of modern TSP solvers on interesting distributions of TSP instances. For complete solvers,

an important observation is made the book by Applegate et al. [3], who present a graphical analysis of observed mean running times suggesting that Concorde may scale exponentially. A more thorough investigation is presented by Hoos and Stützle [35], which is a direct precursor of our work on TSP. After observing a log-normal distribution of the running times for given instance size, they found that the median (and other quantiles of) running times of Concorde scale as a function of the form $a \cdot b^{\sqrt{n}}$, where b is about 1.24194. In addition to analysing running times required by Concorde to find an optimal solution and to prove its optimality, there is work investigating how much of the time is spent on finding the optimal solution. Hoos and Stützle [36] found that time spent on finding optimal solutions accounts for a larger percentage of the overall running time, and the percentage tends to increase with instance size.

For incomplete solvers, Dubois-Lacoste et al. [21] studied the scaling of running times of EAX and LKH (with restart mechanisms), also on solving RUE instances. They found that scaling models of EAX and LKH are of a similar form with Concorde, namely of the form $a \cdot b^{\sqrt{n}}$, but they seem to scale better than Concorde in that their models have smaller values of b. We note that some of this work was carried out in parallel with our work presented in the following, and ours complemented and extended their results for a better understanding of the scaling behaviour of these two incomplete algorithms.

2.3 Related Work on Empirical Scaling Analysis

Empirical analysis of the time complexity of algorithms has been applied to problem other than SAT and TSP. Some of these studies use graphical analysis only, while others involve the use of model fitting. Subramani and Desovski [67], for instance, investigated the empirical scaling of the vertex contraction (VC) algorithm, a very well known greedy procedure for the negative cost cycle detection (NCCD) problem. Their comprehensive results demonstrated that the VC algorithm outperform the standard Bellman-Ford algorithm by an order of magnitude. Another example is found in work by Kunkle [42], which studied four different algorithms for the longest common subsequence problem. Referencing theoretical results, Kunkle fitted one-parameter models to find values of the constants in the models. He also examined the impact of sequence structure and alphabet size on the empirical running times of the algorithms. Aguirre-Hernández et al. [2] also employed graphical analysis and model fitting to analyse two algorithms for the design of RNA secondary structure. In particular, they investigated the impact of RNA structures and primary structure constraints on the scaling of these algorithms and found support of polynomial scaling for both algorithms.

Moreover, there has been significant interest in the methodology for empirical scaling analysis and in its role in complexity research. Sanders and Fleischer [61], for instance, discussed the role of empirical investigations in guiding theoretical research, falsifying results from incomplete theoretical analysis or even producing high-quality surrogates for unproven conjectures on scaling of algorithms. They illustrated this via several examples, some involving graphical scaling analysis and even the standard t-test to evaluate hypotheses on scaling of algorithms. McGeoch et al. [48, 49] also examined the role of analysing asymptotic trends from experimental data as part of a scientific method and evaluated several curve-bounding techniques using designed experiments. In particular, they focused on polynomial models and evaluated their techniques on data, both artificial and real, that are known to be bounded by low-degree polynomials. Success was seen for a strategy that finds bounding polynomials essentially by performing linear regression on log-log transformed data. On the other hand, they observed unsatisfactory results from non-linear regression-based strategies. This observations, as Hoos [33] suspected, might come from the difficulty of bounding slowly-growing scaling behaviour and from bounding (rather than fitting) data with a RMSE-minimisation procedure.

Another development is to use empirical scaling analysis as a testing tool, allowing developers to test whether their software scales as theory predicts and/or as they expect. For this purpose, Goldsmith et al. [28] developed a tool named Trend Profiler that analyse the computational complexity of software as a function of a feature (such as size) of a given workload. Trend Profiler models the execution frequencies of blocks or clusters of blocks of software and fits them to linear or power law models using linear regression (on actual or transformed data). These models are qualitatively assessed by scatter plots that relate input feature values to observed and predicted execution frequencies, and to the residues, that is, differences between observed and predicted frequencies. R^2, the square of Pearson's correlation coefficient, is also used to quantify the quality of such models. Trend Profiler also calculates bootstrap confidence intervals for predictions made on workloads with extrapolated input feature values, though these confidence intervals are not utilised in statistical testing. Goldsmith et al. [28] have demonstrated the effectiveness of Trend Profiler on several pieces of real-world software, including bzip2 [12] and gcov [27]. All software they considered showed slow scaling of low-degree polynomials.

More recently, Hoos [33] proposed an empirical scaling analysis methodology that emphasised the idea of challenging by extrapolation and used bootstrap resampling to statistically assess obtained models. Different from earlier approaches, this method uses non-linear numerical techniques to fit models. Our work directly builds on this methodology and improves in two useful ways, described detailedly in Chapter 3.

Chapter 3

Empirical Scaling Methodology and Extensions

3.1 Empirical Scaling Methodology

A significant advance in the methodology for studying the empirical scaling of algorithm performance with input size was achieved by Hoos [33]. His method uses standard numerical optimisation approaches to automatically fit scaling models, which are then challenged by extrapolation to larger input sizes. Most importantly, it uses a re-sampling approach to assess the models and their predictions in a statistically meaningful way. Here, we briefly reviewed the methodology in steps, as presented in Figure 3.1:

1. **Collect running time data.**
   Running time data should be collected for one type of benchmark instances of interest. These instances need to be collected or generated in sets of different sizes, and the number of different sizes and the number of instances in each set should be carefully chosen, considering the goal of the analysis and the budget of computation resources. Then, algorithm runs should be performed on these instances in a homogeneous environment, and the running times should be reliably measured. For randomised algorithms, multiple runs are required for each instance.

2. **Fit parametric models.**
   Running times of an algorithm typically vary greatly for instances of the same size. To study scaling of running times with instance size, a statistic is typically used to summarise running time data of the same size. Typical examples include the median, the mean, and higher quantiles. Medians and other quantiles have the advantage that they can be more reliably estimated, even when there are (not too many) time-out or crashed runs. Then, a standard numerical optimisation procedures, such as the Levenberg-Marquardt Algorithm [43, 47], is used to fit scaling models over summarised running time data. To perform the next step, only running times of smaller instances

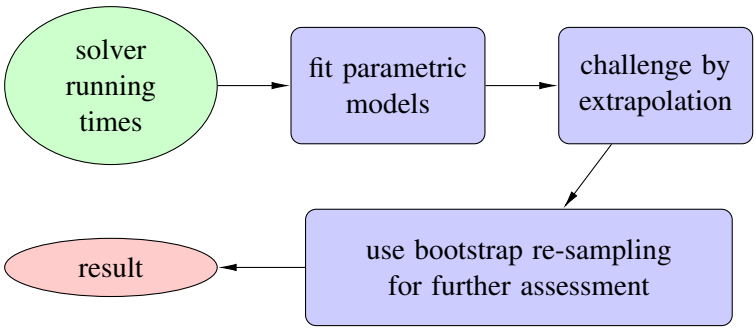


Figure 3.1: The methodology for empirical scaling analysis.

(support data) should be used for the fitting process. The numerical optimisation procedure tries to minimise an error measure such as root-mean-squared error (RMSE). Commonly considered models are polynomial and exponential models of the form $a \cdot n^b$ and $a \cdot b^n$, but other parametric models may be considered as necessary.

3. **Challenge models by extrapolation.**
Extrapolation is viewed as an important way to evaluate fitted models. The reason is that scaling models are of interest mainly because their ability to predict of running times for larger, unseen instances, and we usually accept or reject a scaling model based on whether good predictions can be made for solving larger problem instances. Thus, models obtained from the last step are challenged by running times required to solve larger instances, which are referred to as challenge data. In other words, RMSEs on the challenge data are more informative than those on the support data, and should be relied on when comparing scaling models.

4. **Assess models using bootstrap confidence intervals.**
Bootstrap analysis [22] is used to assess the statistical confidence we should have in the fitted models. In detail, the method re-samples, with replacement, the running times for each support size. For each sample, it computes the descriptive statistic of interest (*e.g.*, median running time) and fits a parametric model which gives predictions for the challenge sizes. Through this process, a collection of fitted models is obtained for each model family, which then generates a set of predictions for the challenge sizes. Bootstrap percentile confidence intervals (for certain confidence level denoted as α) are then computed for each challenge size and for each model family. By comparing the observed running time statistics with the confidence intervals of a model

family, we can determine whether a model is consistent with observations or should be rejected with confidence level of α.

3.2 Extensions to the Methodology

Our work presented in the following applies this empirical methodology to SAT and TSP solvers, and extends it in two useful ways.

Firstly, noting that observed running time statistics for challenge data are also based on measurements on sets of instances, we calculate bootstrap percentile confidence intervals for those, in addition to the point estimates used in the original approach. This way, we capture dependency of these statistics on the underlying sample of instances.

More formally, we have running time data for instance sizes $n_1, \cdots, n_{s+t}$, divided into a support set of s different sizes and a challenge set of t sizes. For each size n_i, we have a set of instances I_i. We have also collected a set of running time data $D(I_i)$ for each size. When computing bootstrap confidence intervals, we resample, with replacement, r samples of the instance set $I_{i,1}, \cdots, I_{i,r}$. For each sample $I_.$, we compute statistics $S(I_.)$ of the support sizes for model fitting, which then gives us confidence intervals $CI_1, \cdots, CI_t$ of model predictions (see step 4 above). In [33], these confidence intervals are directly compared to $S(I_{s+1}), \cdots, S(I_{s+t})$ to assess whether a model is a good fit. However, these statistics are also based on sets of running time data. Thus, we calculate bootstrap confidence intervals for observed challenge data as well. That is, we compute CIO_i for $i = 1, \cdots, t$ as confidence intervals of $S(I_{s+i,1}), \cdots, S(I_{s+i,r})$ and compare it with CI_i to assess whether a model is a good fit or not. The comparison results can be: the two confidence intervals are disjoint (which we treat as inconsistent), they overlap with each other but are not fully contained (which we say to be weakly consistent), and one is contained in another (which we say to be fully consistent). (In addition, if the point estimate of the observed running time falls within the confidence interval of the prediction, we say it to be strongly consistent.)

Secondly, we propose a way to compare scaling models between algorithms. This is done by cross-checking model predictions and observed data. We determine to which extent a solver $\mathscr{A}_1$ shows scaling behaviour different from another solver $\mathscr{A}_2$, by comparing the observed running time statistics of $\mathscr{A}_1$ to the bootstrap confidence intervals obtained from the scaling model of $\mathscr{A}_2$. In other words, we compare CIO_i's for $i = 1, \cdots, n$ of $\mathscr{A}_1$ to the CI_i's of $\mathscr{A}_2$. If the latter do not overlap with the former, we can reject the hypothesis that the performance of $\mathscr{A}_1$ is consistent with the scaling model of $\mathscr{A}_2$. We note that this method also applies to comparing scaling models of one algorithm solving two different distributions of

instances.

A similar methodology can be used to determine if two scaling models differ by a constant factor. For example, if we have shown that the running times of $\mathscr{A}_1$ and $\mathscr{A}_2$ are consistent with an exponential model of the form $a \cdot b^n$, and the model for $\mathscr{A}_1$ is $a_1 \cdot b_1^n$, then we can determine if the scaling of the two algorithms differ by a constant factor by fitting a one-parameter model of the form $a \cdot b_1^n$, where a is a free parameter, on the running times of $\mathscr{A}_2$. If the resulting model is not a good fit (determined based on bootstrap confidence intervals, as explained in step 4 of Sec. 3.1), we reject the hypothesis that the two models differ by only a constant factor with confidence level of α.

Chapter 4

Empirical Scaling Analyser: An Automated System for Scaling Analysis[1]

In this chapter, we present an automated tool – the Empirical Scaling Analyser (ESA) – that can perform core elements, particularly steps 2 through 4, of the analysis described in Chapter 3. (The workflow implemented by ESA is illustrated in Figure 3.1) To use ESA, a user needs to prepare an input file of running time data of an algorithm (referred to as target algorithm hereafter), as well as other optional files, including a configuration file, a file specifying the parametric models to be fitted, a LaTeX template and a gnuplot template. Details of these input files will be given in Section 4.1. Note that ESA is not limited to fitting and assessing a single scaling model, but can deal with multiple models simultaneously. In other words, once data collection is finished, a user can put all running time data into a file, feed it into ESA and obtain the results from the scaling analysis using several parametric models. Results are presented in a technical report, which contains easy-to-read tables and figures for the scaling of the target algorithm. The details of the output report are described in Section 4.2.

We believe the tool is useful for other researchers who want to study the empirical time complexity of other algorithms, and can thus promote the use of such analysis for other problems and algorithms. The tool is available as an easy-to-use online service at `www.cs.ubc.ca/labs/beta/Projects/ESA/esa-online.html` and as a command-line tool with additional functionality (see Section 4.5 on how to run ESA). Here, we describe all features available in the command-line version.

4.1 Input

To perform scaling analysis, ESA requires input data containing the sizes of the instances studied and the running times the target algorithm requires for solving

[1] A shorter version was published as a late breaking abstract in GECCO'15 [53].

these instances. The input running time data need to follow the following formatting rules:

- the input file contains lines of details of the instances, one instance per line;
- in each line, the following three pieces of information are provided in order and are separated by ",":
  - instance name (e.g., file name) and other optional information (this field is for the user's reference only; ESA does not use this field in the scaling analysis);
  - instance size;
  - running time required to solve the instance, which itself may be a statistic for multiple runs of the target algorithm solving the instance and may be "inf" for time-out or crashed runs.

Besides, the user may specify the number of instances for some sizes. If there are not enough entries for one size, ESA will treat the missing entries as instances with unknown running time. An example for such data is described in Section 6.3, in the context of analysing the scaling behaviour of EAX and LKH, where the running times of some instances are unknown because no optimal solution has been found in previous runs of Concorde. An excerpt of an input file for ESA is in Figure 4.1.

ESA also takes as input a configuration file, containing details on the target algorithm (algName), the instance distribution (instName), the number of bootstrap samples (numTrainingData), etc. The file contains lines of configurations, one configuration per line. Each configuration follows the "`name : value`" format. An example of a configuration file is shown in Figure 4.2.

There are a number of other files that a user may supply (if not supplied, ESA will use the default file(s) distributed with the code), including:

- a file specifying the models to be fit
- a LaTeX template specifying the content and format of the output report
- gnuplot templates specifying the format of the plots

The first of these is needed, because ESA supports customised models, as long as the models are supported by python (including the math and the numpy packages) and gnuplot. This file contains lines of models, one model per line. Each contains the following items, separated by ",":

- Model name (e.g., Exponential)

```
# instance name, size, datum (running time)
portgen-500-1000.tsp,500,2.3
portgen-500-100.tsp,500,2.58
portgen-500-101.tsp,500,2.36
portgen-500-102.tsp,500,2.51
portgen-500-103.tsp,500,2.63
portgen-500-104.tsp,500,2.84
portgen-500-105.tsp,500,2.62
portgen-500-106.tsp,500,3
...
portgen-600-1000.tsp,600,3.42
...
portgen-4500-10.tsp,4500,727.68
portgen-4500-11.tsp,4500,inf
...
#instances,4000,100
#instances,4500,100
```

Figure 4.1: Excerpt of an input file for ESA, where deleted lines are represented by "...".

- Number of parameters (e.g., 2)
- LaTeXexpression of the model
- Python expression of the model
- Gnuplot expression of the model
- Default values of the parameters, separated by ","

For all expressions of the models, x represents the size, and the parameters should be $a, b, \ldots$, and should be surrounded by "@@". For example, the specification in Figure 4.3, which is also the default model specification, tells ESA to fit an exponential and a polynomial model of the form $a \cdot b^x$ and $a \cdot x^b$ respectively:

For the LaTeX template, ESA will use the default template, if no customised template is found. In the template, dynamic values should be surrounded by "@@". For instance, the name of the algorithm (which is defined in the configuration file) is a dynamic value. Wherever mentioned in the template file, the user should use "@@algName@@", and ESA will instantiate it to be the real name of the algorithm when generating the report. Users can also specify the formats of the plots

```
fileName : runtimes.csv
algName : WalkSAT/SKC
instName : random 3-SAT instances at phase transition
modelFileName : models.txt
numTrainingData : 7
alpha : 95
numBootstrapSamples : 1000
statistic : median
latexTemplate : template-AutoScaling.tex
modelPlotTemplate : template_plotModels.plt
residuePlotTemplate : template_plotResidues.plt
```

Figure 4.2: Example of a configuration file for ESA.

```
Exp,2,@@a@@\times @@b@@^{x},@@a@@*@@b@@**x,@@a@@*
   @@b@@**x,1e-4,1.01
Poly,2,@@a@@\times x^{@@b@@},@@a@@*x**@@b@@,@@a@@*x**
   @@b@@,1e-8,1
```

Figure 4.3: An example of model specification for ESA.

n	500	600	700	800
# instances	1000	1000	1000	1000
# running times	1000	1000	1000	1000
mean	19.33	31.55	56.8	89.35
coefficient of variation	1.164	1.418	1.303	1.455
Q(0.1)	0.43	2.21	6.73	10.62
Q(0.25)	3.75	8.75	15.3	22.31
median	12.22	18.64	33.15	48.95
Q(0.75)	25.01	37.64	68.29	102.5
Q(0.9)	47.33	70.35	130.4	195.5

n	900	1000	1100	1200
# instances	1000	1000	1000	1000
# running times	1000	1000	1000	1000
mean	139.7	201.2	314.6	385.4
coefficient of variation	1.734	1.759	1.851	1.713
Q(0.1)	17.23	23.28	28.8	38.76
Q(0.25)	32.72	43.23	60.84	78.66
median	70.64	98.56	145.7	177.2
Q(0.75)	142.7	216.1	341.3	409.2
Q(0.9)	302.7	429.4	693.2	846.3

n	1300	1400	1500
# instances	1000	1000	1000
# running times	1000	1000	1000
mean	548.7	749	1072
coefficient of variation	1.859	2.227	2.109
Q(0.1)	53.27	73.78	93.04
Q(0.25)	112.2	153.3	210.1
median	271.7	344.3	483.5
Q(0.75)	583.6	783.6	1136
Q(0.9)	1190	1517	2277

Table 4.1: Example output of ESA – statistics of running times for support data.

via the template gnuplot script. For instance, users may choose whether to use a log-log plot or a semi-log plot via the template gnuplot script. Default templates are presented in Appendices A and B and are available for download together with the source code (see Section 4.5 for details).

4.2 Output

ESA automatically generates a technical report containing detailed empirical scaling analysis results and interpretation. This report contains tables and figures that users can easily read (see Chapters 5 and 6 for examples), including:

- two tables of statistics of running times, one for support data and the other for challenge, as illustrated in Tables 4.1 & 4.2, respectively;

n	2000	2500	3000
# instances	1000	100	100
# running times	1000	100	100
mean	5402	∞	∞
coefficient of variation	2.624	N/A	N/A
Q(0.1)	307	671.5	3538
Q(0.25)	765.5	1694	8188
median	1969	6149	1.84×10^4
Q(0.75)	5207	1.766×10^4	4.118×10^4
Q(0.9)	1.137×10^4	4.611×10^4	9.048×10^4

n	3500	4000	4500
# instances	100	100	100
# running times	100	100	100
mean	∞	∞	∞
coefficient of variation	N/A	N/A	N/A
Q(0.1)	6060	2.096×10^4	3.041×10^4
Q(0.25)	1.226×10^4	4.125×10^4	1.22×10^5
median	3.246×10^4	1.312×10^5	2.633×10^5
Q(0.75)	9.717×10^4	3.938×10^5	∞
Q(0.9)	2.76×10^5	∞	∞

Table 4.2: Example output of ESA – statistics of running times for challenge data.

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

Table 4.3: Example output of ESA – fitted models and corresponding RMSE values.

- a table of fitted models and corresponding RMSE values, as illustrated in Table 4.3;
- a figure of running times, fitted models and corresponding bootstrap confidence intervals of each model, as illustrated in Figure 4.4;
- a figure of residues of the fitted models, as illustrated in Figure 4.5;
- a table of bootstrap confidence intervals for all model parameters, as illustrated in Table 4.4;
- two tables of bootstrap confidence intervals for observed and predicted running times, one for support data and the other for challenge, as illustrated in Tables 4.5 & 4.6.

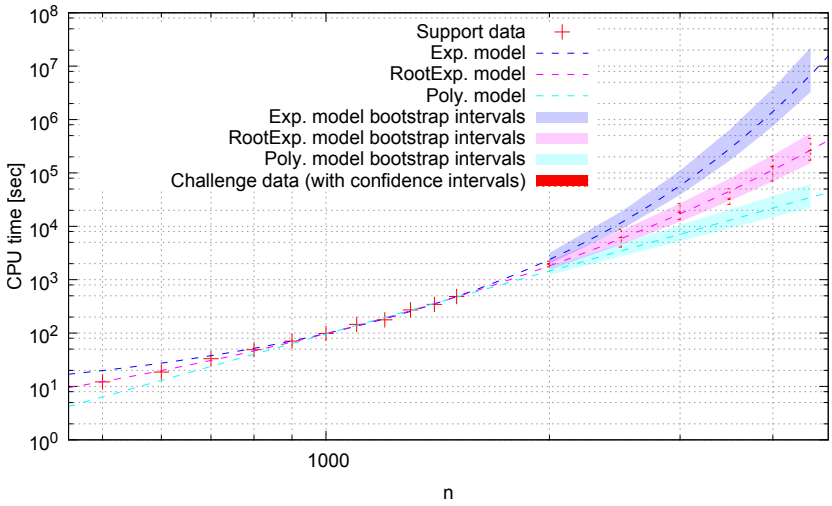


Figure 4.4: Example output of ESA – a figure of running times, fitted models and corresponding bootstrap confidence intervals of each model.

Solver	Model	Confidence interval of a	Confidence interval of b
Concorde	Exp.	$[2.6108, 5.2975]$	$[1.003, 1.0036]$
	RootExp.	$[0.037056, 0.15111]$	$[1.2287, 1.2793]$
	Poly.	$[6.1872 \times 10^{-12}, 1.7351 \times 10^{-9}]$	$[3.5859, 4.3713]$

Table 4.4: Example output of ESA – bootstrap confidence intervals for all model parameters.

Solver	n	Predicted confidence intervals Exp. model	Observed median run-time Point estimates	Observed median run-time Confidence intervals
Concorde	500	[15.43, 23.47]	12.22	[11.01, 13.33]
	600	[22, 31.52]	18.64	[16.99, 20.16]
	700	[**31.38**, **42.57**]#	33.15	[30.38, 35.9]
	800	[**44.68**, **57.59**]#	48.95	[44.05, 53.15]
	900	[**63.49**, **77.63**]*	70.64	[64.34, 77.28]
	1000	[**90.18**, **104.8**]#	98.56	[90.16, 107.8]
	1100	[**127.6**, **141.9**]	145.7	[130.9, 158.6]
	1200	[**178.6**, **193**]	177.2	[165.8, 196.1]
	1300	[**244.7**, **266.8**]	271.7	[244.9, 298.3]
	1400	[**332.7**, **375.7**]#	344.3	[318.9, 383.3]
	1500	[**448.8**, **534.7**]#	483.5	[432.8, 532.9]

Solver	n	Predicted confidence intervals RootExp. model	Observed median run-time Point estimates	Observed median run-time Confidence intervals
Concorde	500	[**9.176**, **15.31**]*	12.22	[11.01, 13.33]
	600	[**15.56**, **23.81**]*	18.64	[16.99, 20.16]
	700	[**25.21**, **35.83**]#	33.15	[30.38, 35.9]
	800	[**39.43**, **52.46**]#	48.95	[44.05, 53.15]
	900	[**60.23**, **74.99**]#	70.64	[64.34, 77.28]
	1000	[**89.56**, **105.2**]#	98.56	[90.16, 107.8]
	1100	[**130.4**, **145.2**]	145.7	[130.9, 158.6]
	1200	[**184.4**, **199.1**]	177.2	[165.8, 196.1]
	1300	[**252**, **274**]#	271.7	[244.9, 298.3]
	1400	[**336.6**, **380.4**]#	344.3	[318.9, 383.3]
	1500	[**442**, **522.8**]#	483.5	[432.8, 532.9]

Solver	n	Predicted confidence intervals Poly. model	Observed median run-time Point estimates	Observed median run-time Confidence intervals
Concorde	500	[4.206, 8.58]	12.22	[11.01, 13.33]
	600	[9.398, 16.47]	18.64	[16.99, 20.16]
	700	[18.41, 28.62]	33.15	[30.38, 35.9]
	800	[**32.91**, **46.48**]	48.95	[44.05, 53.15]
	900	[**55.04**, **71.24**]#	70.64	[64.34, 77.28]
	1000	[**86.98**, **104.3**]#	98.56	[90.16, 107.8]
	1100	[**131.4**, **147.6**]#	145.7	[130.9, 158.6]
	1200	[**188.5**, **204.4**]	177.2	[165.8, 196.1]
	1300	[**257.9**, **280.1**]#	271.7	[244.9, 298.3]
	1400	[**339.3**, **384.2**]#	344.3	[318.9, 383.3]
	1500	[**435.4**, **516.3**]#	483.5	[432.8, 532.9]

Table 4.5: Example output of ESA – bootstrap confidence intervals for observed and predicted running times for support data.

Solver	n	Predicted confidence intervals	Observed median run-time	
		Exp. model	Point estimates	Confidence intervals
Concorde	2000	$[\mathbf{1988}, \mathbf{3179}]$	1969	$[1739, 2222]$
	2500	$[\mathbf{8718}, \mathbf{1.884 \times 10^4}]$	6149	$[4084, 8812]$
	3000	$[3.853 \times 10^4, 1.103 \times 10^5]$	1.84×10^4	$[1.332 \times 10^4, 2.669 \times 10^4]$
	3500	$[1.698 \times 10^5, 6.479 \times 10^5]$	3.246×10^4	$[2.581 \times 10^4, 5.038 \times 10^4]$
	4000	$[7.5 \times 10^5, 3.809 \times 10^6]$	1.312×10^5	$[7.073 \times 10^4, 2.024 \times 10^5]$
	4500	$[3.301 \times 10^6, 2.245 \times 10^7]$	2.633×10^5	$[1.73 \times 10^5, 4.419 \times 10^5]$

Solver	n	Predicted confidence intervals	Observed median run-time	
		RootExp. model	Point estimates	Confidence intervals
Concorde	2000	$[\mathbf{1528}, \mathbf{2269}]$*	1969	$[1739, 2222]$
	2500	$[\mathbf{4536}, \mathbf{8335}]$#	6149	$[4084, 8812]$
	3000	$[\mathbf{1.212 \times 10^4}, \mathbf{2.694 \times 10^4}]$*	1.84×10^4	$[1.332 \times 10^4, 2.669 \times 10^4]$
	3500	$[\mathbf{3.001 \times 10^4}, \mathbf{7.925 \times 10^4}]$#	3.246×10^4	$[2.581 \times 10^4, 5.038 \times 10^4]$
	4000	$[\mathbf{6.95 \times 10^4}, \mathbf{2.163 \times 10^5}]$*	1.312×10^5	$[7.073 \times 10^4, 2.024 \times 10^5]$
	4500	$[\mathbf{1.528 \times 10^5}, \mathbf{5.563 \times 10^5}]$*	2.633×10^5	$[1.73 \times 10^5, 4.419 \times 10^5]$

Solver	n	Predicted confidence intervals	Observed median run-time	
		Poly. model	Point estimates	Confidence intervals
Concorde	2000	$[\mathbf{1228}, \mathbf{1795}]$	1969	$[1739, 2222]$
	2500	$[\mathbf{2737}, \mathbf{4771}]$	6149	$[4084, 8812]$
	3000	$[5252, 1.057 \times 10^4]$	1.84×10^4	$[1.332 \times 10^4, 2.669 \times 10^4]$
	3500	$[9149, 2.069 \times 10^4]$	3.246×10^4	$[2.581 \times 10^4, 5.038 \times 10^4]$
	4000	$[1.477 \times 10^4, 3.708 \times 10^4]$	1.312×10^5	$[7.073 \times 10^4, 2.024 \times 10^5]$
	4500	$[2.248 \times 10^4, 6.205 \times 10^4]$	2.633×10^5	$[1.73 \times 10^5, 4.419 \times 10^5]$

Table 4.6: Example output of ESA – bootstrap confidence intervals for observed and predicted running times for support data.

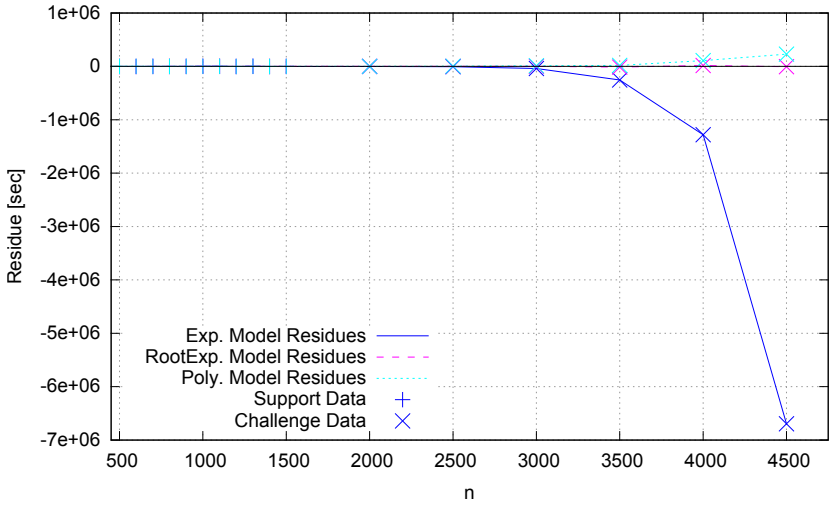


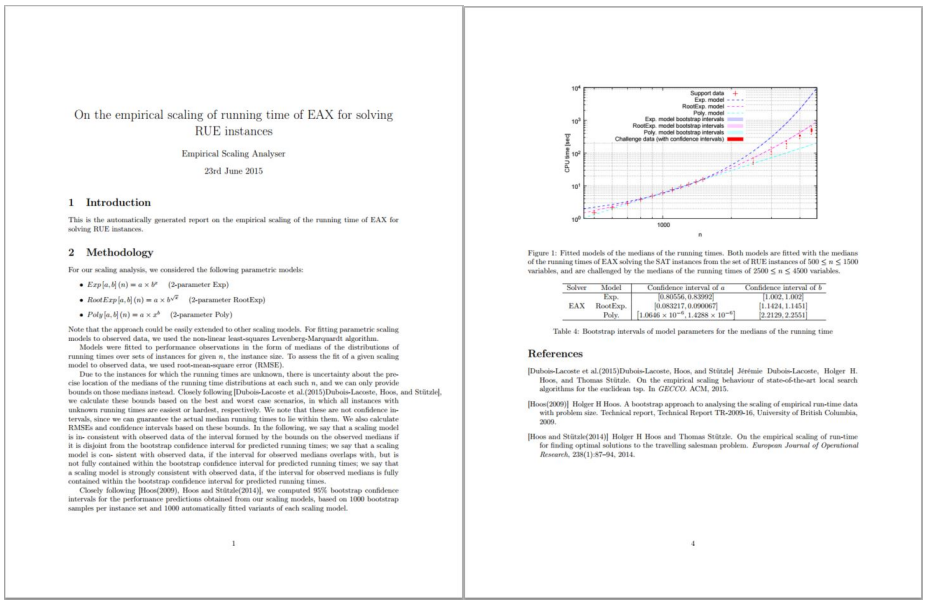
Figure 4.5: Example output of ESA – a figure of residues of the fitted models.

A snapshot of the reported generated by ESA using the default LaTeX template is shown in Figure 4.6.

4.3 Automated Interpretation of Scaling Results

In addition, ESA generates automated interpretations for scaling analysis results. It evaluates how well a model fits the given data based on the percentage of challenge sizes for which the model predicts the corresponding running times reasonably accurately. If a model predicts well for most challenge sizes, then the model should be accepted as a good fit. Technically, the evaluation is based on the percentage of challenge points that lie within the predicted bootstrap confidence intervals of the model. It especially emphasises the challenge points for larger input sizes, as those provide more information regarding whether the model predicts well. The detailed criteria, which we design based on extensive experiments with SAT and TSP algorithms, are as follows:

- fair fit: the model predicts well for a fair percentage of the challenge sizes, more precisely, $> 70\%$ of the challenge points or $> 70\%$ of the larger half of the challenge points are within the predicted bootstrap intervals;
- very good fit: the model predicts well for almost all challenge sizes, more precisely,$> 95\%$ of the challenge points are within the predicted bootstrap intervals;



On the empirical scaling of running time of EAX for solving RUE instances

Empirical Scaling Analyser

23rd June 2015

1 Introduction

This is the automatically generated report on the empirical scaling of the running time of EAX for solving RUE instances.

2 Methodology

For our scaling analysis, we considered the following parametric models:

- $Exp[a,b](n) = a \times b^n$ (2-parameter Exp)
- $RootExp[a,b](n) = a \times b^{\sqrt{n}}$ (2-parameter RootExp)
- $Poly[a,b](n) = a \times n^b$ (2-parameter Poly)

Note that the approach could be easily extended to other scaling models. For fitting parametric scaling models to observed data, we used the non-linear least-squares Levenberg-Marquardt algorithm.

Models were fitted to performance observations in the form of medians of the distributions of running times over sets of instances for given n, the instance size. To assess the fit of a given scaling model to observed data, we used root-mean-square error (RMSE).

Due to the instances for which the running times are unknown, there is uncertainty about the precise location of the medians of the running time distributions at each such n, and we can only provide bounds on those medians instead. Closely following [Dubois-Lacoste et al.(2015)Dubois-Lacoste, Hoos, and Stützle], we calculate these bounds based on the best and worst case scenarios, in which all instances with unknown running times are easiest or hardest, respectively. We note that these are not confidence intervals, since we can guarantee the actual median running times to lie within them. We also calculate RMSEs and confidence intervals based on these bounds. In the following, we say that a scaling model is in- consistent with observed data of the interval formed by the bounds on the observed medians if it is disjoint from the bootstrap confidence interval for predicted running times; we say that a scaling model is con- sistent with observed data, if the interval for observed medians overlaps with, but is not fully contained within the bootstrap confidence interval for predicted running times; we say that a scaling model is strongly consistent with observed data, if the interval for observed medians is fully contained within the bootstrap confidence interval for predicted running times.

Closely following [Hoos(2009), Hoos and Stützle(2014)], we computed 95% bootstrap confidence intervals for the performance predictions obtained from our scaling models, based on 1000 bootstrap samples per instance set and 1000 automatically fitted variants of each scaling model.

1

Figure 1: Fitted models of the medians of the running times. Both models are fitted with the medians of the running times of EAX solving the SAT instances from the set of RUE instances of $500 \leq n \leq 1500$ variables, and are challenged by the medians of the running times of $2500 \leq n \leq 4500$ variables.

Solver	Model	Confidence interval of a	Confidence interval of b
EAX	Exp.	[0.80556, 0.83992]	[1.002, 1.002]
	RootExp.	[0.083217, 0.090067]	[1.1424, 1.1451]
	Poly.	$[1.0646 \times 10^{-6}, 1.4288 \times 10^{-6}]$	[2.2129, 2.2551]

Table 4: Bootstrap intervals of model parameters for the medians of the running time

References

[Dubois-Lacoste et al.(2015)Dubois-Lacoste, Hoos, and Stützle] Jérémie Dubois-Lacoste, Holger H. Hoos, and Thomas Stützle. On the empirical scaling behaviour of state-of-the-art local search algorithms for the euclidean tsp. In *GECCO*. ACM, 2015.

[Hoos(2009)] Holger H Hoos. A bootstrap approach to analysing the scaling of empirical run-time data with problem size. Technical report, Technical Report TR-2009-16, University of British Columbia, 2009.

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

4

Figure 4.6: Snapshot of the technical report generated by ESA.

- over-/under-estimate: the model over-/under-estimates the running times of a significant percentage of the challenge sizes, more precisely, $> 75\%$ of the challenge points or $> 75\%$ of the larger half of the challenge points are below/above the predicted bootstrap intervals.

These criteria are combined into the fully automated interpretation procedure illustrated in Figure 4.7. Note that when medians (or other quantiles) are not definitely known (due to instances with unknown running times), we compare the intervals of the medians against the predicted bootstrap intervals. To be more precise, for instance, we say a challenge point is below the predicted bootstrap interval, if the upper bound of the median is smaller than the lower bound of the bootstrap interval.

4.4 Implementation

ESA is implemented in python 2.7 and calls gnuplot to generate plots. All provided gnuplot scripts are prepared for gnuplot version 4.6, but only minimal modifications, if any, will be needed to use with another gnuplot version. The online service has a clear user interface implemented with HTML/CSS, and the back-end service was realised using python CGI.

One technical difficulty for implementing ESA is to interact with the LaTeX

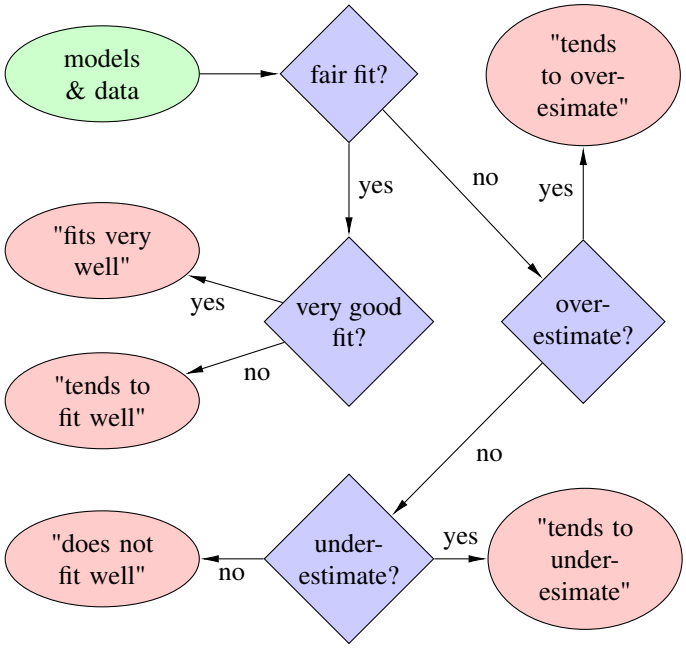


Figure 4.7: Flow diagram on how ESA automatically interprets the fitting results. Detailed definitions of the conditions are given in the main text.

template for producing the output technical report. Generating LaTeX commands, especially those concerning tables, requires careful implementation. Moreover, it was important to modularise the code for maximum re-use. For instance, there is one function that takes a string as input and output a string that instructs LaTeX to make the content boldface. This is very frequently used in ESA, since boldface is widely used for highlighting models and bootstrap confidence intervals. The function will also need to distinguish between plain text and mathematics environments, as different commands are needed to make them boldface. Other examples include the conversion of numbers into LaTeX expressions, and the generation of table rows. Modularisation is also of great value beyond working with LaTeX, as several operations need to be performed in more than one stage of an ESA run. An example of such operations is to call gnuplot for model fitting/plotting, which needs to be done for each model in steps 3 and 4.

In addition, ESA was designed to work with customisable LaTeX and gnuplot templates (see Sec. 4.1 for details). There is a special LaTeX syntax that can be used as "variables" in the LaTeX template, which will be replaced by actual content when ESA runs. The LaTeX template can also be compiled "as-is"to make it easier for users to adapt it to their specific needs. We chose gnuplot for plot generation so that users can easily supply a template for customised figure formatting. ESA also supports user-defined models for scaling analysis. To achieve this, ESA defines functions on-the-fly from user-supplied strings.

4.5 Downloading and Running ESA

4.5.1 Downloading and Running ESA from the Command Line

ESA can be downloaded from the project page online at `www.cs.ubc.ca/labs/beta/Projects/ESA/`. After unzipping the compressed file, there will be a directory named ESA, which contains runESA.sh, the source code and other support files. A user needs to have python and gnuplot installed in order to run ESA. We used python 2.7 and gnuplot 4.6 in our environment, but expect ESA to work for other versions with minimal modifications, if any.

To run ESA from command line, a user should follow the following steps:

1. Create a directory for input and output files.
2. Put the primary input file for running time data into the directory.
3. Create files for models and LaTeX and gnuplot templates (optional; if not provided, ESA will use the default files distributed with the source code).

4. Create a configuration file named configurations.txt within the directory, telling ESA the details it requires, including names of the algorithm and the instance set/distribution, file names of running time data and, if used, the file names of model specifications and LaTeX and gnuplot templates.

5. Run the script in the ESA directory by `./runESA.sh <directory name>`, and ESA will run according to the specifications in <directory name>/configurations.txt.

4.5.2 Running ESA as a Web Service

ESA is also available online as a web service, which supports the essential but not all features of the command-line version. In particular:

- it only supports three pre-defined models, include:
  - exponential: $a \cdot b^n$,
  - root-exponential: $a \cdot b^{\sqrt{n}}$, and
  - polynomial: $a \cdot n^b$;
- it uses the default LaTeX and gnuplot templates for report generation;
- it only supports a limited number of statistics, include median, mean, as well as 75th, 90th and 95th percentiles.

To run ESA as a web service, visit `www.cs.ubc.ca/labs/beta/Projects/ESA/esa-online.html`, upload the file of running time data and fill in the other details of the form. After submission, ESA will run in the back and redirect the user to the output technical report. The user interface is shown in Figure 4.8.

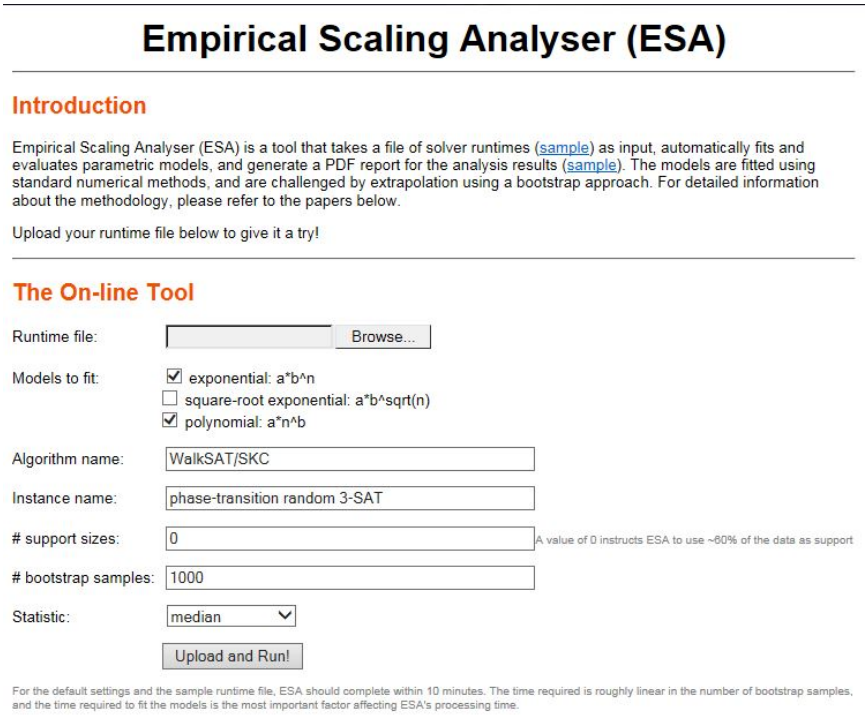


Figure 4.8: The user interface of ESA as a web service.

Chapter 5

Empirical Time Complexity of Random k-SAT[2]

In this chapter, we study the empirical time complexity of random k-SAT, with a focus on random 3-SAT at the phase transition. Our analyses were performed on prominent complete and incomplete SAT solvers. We also study the empirical scaling of these solvers on two interesting distributions of random 4-SAT instances to further understand the time complexity of random k-SAT.

5.1 Experimental Setup

For our study, we selected three SLS-based solvers, WalkSAT/SKC [64], BalancedZ [44] and probSAT [7], and three DPLL-based solvers, kcnfs [20], march_hi [32] and march_br [31]. We chose these, because WalkSAT/SKC and kcnfs are two classical solvers that are widely known in the community, and the others showed excellent performance in recent SAT competitions.

All sets of uniform random 3-SAT instances used in our study were obtained using the same generator used for producing SAT competition instances [8]. To separate satisfiable from unsatisfiable instances with $n \leq 500$, we used the CSHCrandMC hybrid solver[46], the winner of Random SAT+UNSAT Track in the 2013 SAT Competition. For larger instances, for which the use of complete solvers is impractical, we performed long runs (43 200 CPU sec) of BalancedZ and treated those instances not solved in those runs as unsatisfiable. Since BalancedZ never failed to solve any of our instances known to be satisfiable, and because the cut-off time we used for these long runs was at least 20 times of the maximum running time observed on any satisfiable instance of the same size, we have high confidence that we did not miss any satisfiable instances. We note that, even if we had misclassified some satisfiable instances, the quantile performance measures would not be much affected (if at all).

[2]This chapter covers major results from [52] on the empirical time complexity of random 3-SAT at the phase transition.

After observing floor effects for WalkSAT/SKC due to small CPU times required for solving small instances, we calculated running times based on the number of local search steps performed, and on estimates of CPU time per local search step for each problem size obtained from long runs on unsatisfiable instances. The CPU time estimates thus obtained closely agreed with measured CPU times for short and long runs, but are considerably more accurate. For all other solvers, which were less affected by such floor effects, we used runsolver [60] to record CPU times (and to enforce CPU time limits).

For our scaling analysis, we restricted ourselves to 2-parameter models. In principle, our methodology works for models with fewer or more parameters; however, fitting models with more parameters requires more data points (and hence computational resource) and has more potential for overfitting. In particular, we mainly considered two parametric models:

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

For investigating 4-SAT, we also added the following model, which was motivated by Hoos and Stützle [35], to better characterise the scaling behaviour of the solvers:

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

We note that these models are used to capture the first-order terms of the actual scaling function, which we expect to also contain lower-order terms in most cases. Particularly, we did not include constant offsets in these models, which could captured running time for setting up, reporting results, etc., because we believe that, for the problems and instance sizes studied here, those offsets are small enough to be negligible.

Models were fitted to performance observations in the form of quantiles, chiefly the median of the distributions of running times over sets of instances for given n. Compared to the mean, the median has two advantages: it is statistically more stable and is immune to the presence of a certain amount of timed-out runs. Considering the stochastic nature of the SLS-based solvers, we performed 5 independent runs per instance and used the median over those 5 running times as the running time for the respective instance. We note that more stable estimates of the medians can be obtained by performing more runs per instance, but we restricted ourselves to 5 runs due to limitations of our computational resources. Our approach could be easily extended to other scaling models, but, as we will show in the following, these models jointly characterise the scaling observed in all our experiments, and thus we saw no need to consider different or more complex models. For fitting

parametric scaling models to observed data, we used the non-linear least-squares Levenberg-Marquardt algorithm.

Closely following works by Hoos [33] and Hoos and Stützle [35], we computed 95% bootstrap confidence intervals for the performance predictions obtained from our scaling models, based on 1000 bootstrap samples per instance set and 1000 automatically fitted variants of each scaling model. We extended their approach in two ways, which are described detailedly in Chapter 3.

For collecting running time data for our SAT solvers, we used Compute Canada / Westgrid cluster orcinus (DDR), each equipped with two Intel Xeon E5450 quad-core CPUs at 3.0 GHz with 16GB of RAM running 64-bit Red Hat Enterprise Linux Server 5.3.

5.2 Location of Phase Transition

We first revisited studies on the location of phase transition for 3- and 4-SAT, and refined existing models for them. This refinement is important because we want to avoid drifting off the phase transition region, which may bias our results.

5.2.1 Location of Phase Transition for 3-SAT

Crawford and Auton modelled the location of the phase transition point based on extensive experiments on uniform random 3-SAT instances with up to 300 variables as:

$$m_c = 4.258 \cdot n + 58.26 \cdot n^{-2/3}, \tag{5.1}$$

where n is the number of variables and m_c the critical number of clauses, at which about 50% satisfiable instances are obtained [17]. The parametric form of this model was derived using finite-size scaling by Kirkpatrick and Selman [39]. While this model does provide a good fit for $n \leq 300$, in preliminary experiments, we found that for $n > 300$, it increasingly underestimates m_c. Furthermore, the model in Eq. 5.1 predicts that, as n grows, m_c/n approaches 4.258, which is in contradiction with a more recent result by Mertens et al., who used the heuristic one-step replica symmetry breaking cavity method from statistical physics to estimate the value of m_c/n as 4.26675 ± 0.00015 [50].

Because in the empirical scaling study that follows, we wanted to be sure to not drift off the phase transition point (which could bias the scaling models, especially, as the phase transition, in terms of m_c/n, is known to become increasingly steep as n grows), we decided to revisit and improve the Crawford & Auton's model.

We first chose several m/n ratios for each $n \in \{300, 400, \ldots, 1400\}$, around the predictions from Eq. 5.1, loosely adjusted based on results from preliminary

experiments. Next, we generated sets of 600 uniform random 3-SAT instances for each m/n ratio. We separated out the satisfiable instances using a hybrid complete solver, CSHCrandMC [46], with a cutoff of 3600 CPU seconds per run. Up to $n = 500$, we solved all instances within that cutoff. Beyond, it would have been impractical to run any complete solver sufficiently long to prove unsatisfiability, and we therefore heuristically assumed that the instances not solved by CSHCrandMC are unsatisfiable. As mentioned earlier, we later used much longer runs of BalancedZ to challenge these putative unsatisfiable instances further, and among the thousands of instances for which this was done, only one was found to be satisfiable. Nevertheless, throughout what follows we are well aware of the fact that we may underestimate the fraction of satisfiable instances in our sets.

We note that, even at large numbers of instances sampled at the correct (unknown) value of m_c/n, we should expect to get sets with a fraction of satisfiable instances varying according to a binomial distribution. Based on this observation, we determined 95% confidence intervals for the fraction of satisfiable instances observed for each n. We then rejected the m/n values for which we empirically observed fractions of satisfiable instances outside of the respective 95% confidence interval as valid estimates of m_c/n. In this way, we obtained bounds on the location of the phase transition point, m_c/n. In many cases, the confidence intervals for our initial sets of 600 instances were too wide to yield bounds, and we subsequently increased the number of instances in those sets until for every n, we obtained an upper and lower bound on m_c/n. Those bounds and the respective set sizes are shown in Table 5.1.

These results provide further evidence for our earlier observation that for larger n, Crawford & Auton's model becomes inaccurate, as the respective predictions are below the lower bounds from our analysis. We note that our lower bounds are valid, as they could only increase if some of our putatively unsatisfiable instances were in fact satisfiable. For the same reason, the upper bounds may be inaccurate.

Interestingly, and notably different from Crawford & Auton's model, our results suggest that m_c/n as a function of n is not monotonic, but first drops, before slowly increasing again, possibly towards a threshold value. This observation, in combination with the limiting value found by Mertens et al. [50] led us to choose a model of the form

$$m_c = 4.26675 \cdot n + a \cdot n^b + c \cdot n^d,$$

where $a \cdot c < 0$, $b < 0$ and $d < 0$.[3]

Finally, taking the data points from work by Crawford and Auton [17] up to

[3] This type of model is consistent with current theoretical thinking that, for small n, second-order terms may cause non-monotonic behaviour of the function $m_c(n)$. (Dimitris Achlioptas, personal communication.)

n	Lower bound	Upper bound	Prediction	
			Eq. 5.1	Eq. 5.2
300	4.2600 (3600)	4.2667 (3600)	4.2623	4.2638
400	4.2575 (4800)	4.2650 (600)	4.2607	4.2626
500	4.2580 (2400)	4.2660 (600)	4.2598	4.2622
600	4.2567 (600)	4.2667 (4800)	4.2594	4.2622
700	4.2600 (1200)	4.2657 (4800)	4.2591*	4.2623
800	4.2575 (2400)	4.2638 (2400)	4.2588	4.2624
900	4.2600 (2400)	4.2667 (3600)	4.2587*	4.2626
1000	4.2600 (600)	4.2670 (2400)	4.2586*	4.2627
1100	4.2609 (1200)	4.2655 (600)	4.2585*	4.2629
1200	4.2600 (2400)	4.2650 (3600)	4.2584*	4.2630
1300	4.2608 (600)	4.2646 (600)	4.2584*	4.2631
1400	4.2600 (1200)	4.2664 (2400)	4.2583*	4.2633

Table 5.1: Lower and upper bounds on the location of the phase transition (m_c/n) for 3-SAT determined with 95% confidence, and predictions from the two models discussed in the text. In parentheses: number of instances used for determining the bound. Model predictions inconsistent with our lower bounds are marked with asterisks (*).

$n = 300$ (which we believe to be of high quality) and the mid-points between our bounds from Table 5.1, we fitted this 4-parameter model, resulting in:

$$m_c = 4.26675n + 447.884n^{-0.0350967} - 430.232n^{-0.0276188} \tag{5.2}$$

Figure 5.1 shows the predictions obtained from this model, along with the bounds and mid-points between them from Table 5.1. This model was subsequently used to generate the instance sets used in the scaling analysis described in Section 5.3. While it is possible that our model is biased by the fact that we may have missed satisfiable instances for larger n, it provides a better basis than previously available for generating instances at or very near the solubility phase transition of uniform random 3-SAT.

5.2.2 Location of Phase Transition for 4-SAT

Following the same approach used for modelling the locations of the phase transition points for 3-SAT (see Section 5.2.1), we run experiments to determine the intervals of the phase transition points for 4-SAT as well. The lower and upper bounds can be found in Table 5.2.

The bounds, in combination with the limiting value found by [50], led us to choose a model of the form

$$r = 9.931 + a \cdot n^b,$$

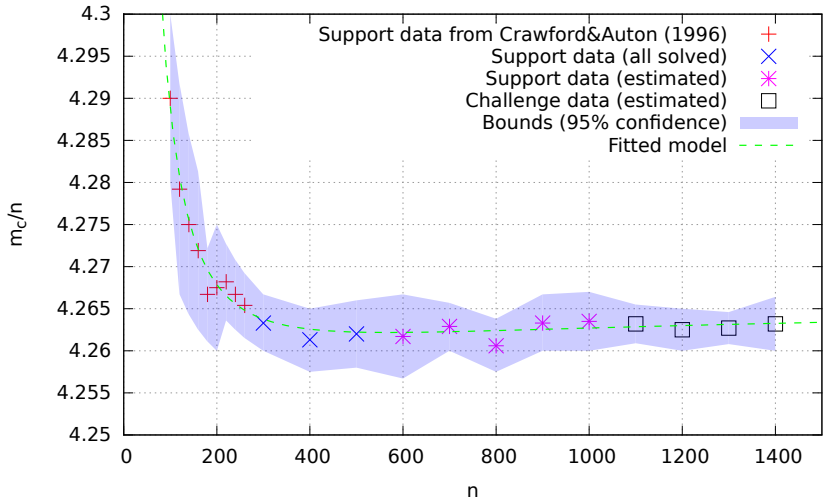


Figure 5.1: Empirical bounds on the location of the phase transition points for 3-SAT, m_c/n for different n, data used for fitting our model (eq. 5.2) and model predictions. Data for $n > 500$ is based on heuristic estimates of the fraction of satisfiable instances and may underestimate the true m_c/n.

resulting, after fitting, in

$$r = 9.931 + 155.729 \times n^{-2.01596}. \tag{5.3}$$

Figure 5.2 shows the predictions obtained from this model, along with the bounds and mid-points between them from Table 5.2. This model was subsequently used to generate the instance sets used in the scaling analysis described in Section 5.4.

5.3 Empirical Scaling of Solver Performance on Random 3-SAT at Phase Transition

We first studied 3-SAT – arguably the conceptually simplest $\mathcal{NP}$-complete problem, and focused on random 3-SAT instances at phase transition, which is a widely known distribution of hard 3-SAT instances.

5.3.1 Empirical Scaling of the Performance of SLS-based Solvers

We first fitted our two parametric scaling models to the median running times of the three SLS-based solvers we considered, as described in Section 5.1. For each SLS-based solver, both models were fitted using median running times for $n =$

# Variables	Lower bound	Upper bound	Prediction by eq. 5.3
20	10.20 (600)	10.40 (600)	10.302
40	9.975 (600)	10.10 (600)	10.023
60	9.95 (2400)	10.0 (1200)	9.972
80	9.90 (600)	9.975 (2400)	9.954
100	9.91 (600)	9.95 (1200)	9.945
120	9.925 (600)	9.958 (2400)	9.941
140	9.921 (600)	9.95 (1200)	9.938
160	9.919 (600)	9.931 (2400)	9.937
180	9.922 (600)	9.944 (600)	9.935
200	9.92 (1200)	9.94 (600)	9.935
250	9.916 (1200)	9.94 (600)	9.933
300	9.917 (600)	9.94 (600)	9.933
350	9.92 (600)	9.9428 (600)	9.932
400	9.92 (600)	9.945 (600)	9.932

Table 5.2: Exclusive lower and upper bounds of phase transition points for 4-SAT determined with 95% confidence, and predictions from the model discussed in the text. The number after the lower/upper bounds (in parentheses) are the least number of instances we have used to conclude with 95% confidence that the ratio is away from the phase transition point.

$200, 250, \ldots 500$ (support) and later challenged with median running times for $n = 600, 700, \ldots 1000$. Details of the instances can be found in Table 5.3. This resulted in the models, shown along with RMSEs on support and challenge data, in Table 5.4. In particular, we illustrate the fitted models of WalkSAT/SKC in Figure 5.3; similar results are obtained for BalancedZ and probSAT (figures not shown).

We note that the RMSEs on support data, *i.e.*, the data used for fitting the models, often, but not always provide a good indication of their predictive power. Based on the latter, in the form of RMSEs on challenge data, we see clear indications that the median running times of all three SLS-based solvers are overall more consistent with our polynomial scaling model than the best exponential model.

But how much confidence should we have in these models? Are the RMSEs small enough that we should accept them? To answer this question, we assessed the fitted models using the bootstrap approach described in Chapter 3 and implemented in ESA. The results of this analysis, shown in Table 5.5 , clearly show that observed median running times for the SLS-based solvers are consistent with our polynomial scaling model and inconsistent with the exponential model (as illustrated in Figure 5.3 for WalkSAT/SKC). Also, this analysis gives us bootstrap confidence intervals for the model parameters, as shown in Table 5.6. Note that the scaling of all SLS-based solvers are consistent with a low-degree polynomial model ($b < 4$). This is especially striking for the larger challenge instances sizes. Limited experiments for even larger instances sizes (up to $n = 2000$) produced

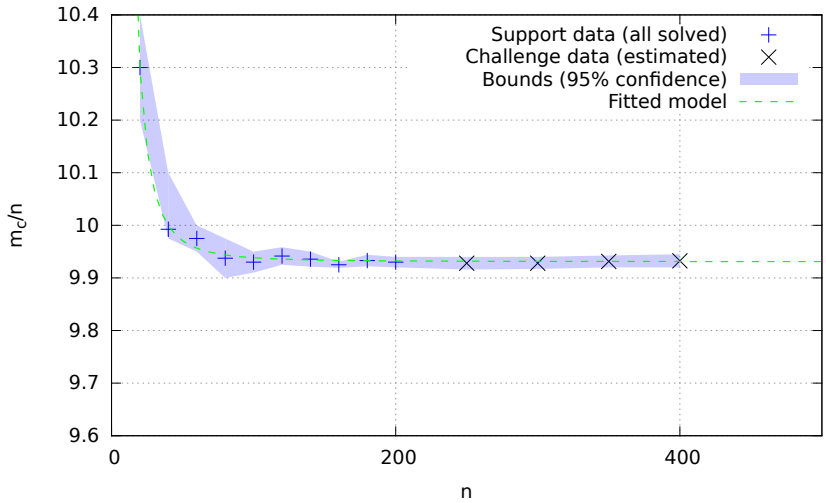


Figure 5.2: Empirical bounds on the location of the phase transition points for 4-SAT, m_c/n for different n, data used for fitting our model (eq. 5.3) and model predictions. Data for $n \geq 200$ is based on heuristic estimates of the fraction of satisfiable instances and may underestimate the true m_c/n.

# Variables	200	250	300	350	400	450	500
# Clauses	854	1066	1279	1492	1704	1918	2130
C-to-V ratio	4.27	4.264	4.263	4.2629	4.260	4.260	4.260
equation 5.2	4.2687	4.2652	4.2638	4.2630	4.2626	4.2623	4.2622

# Variables	600	700	800	900	1000
# Clauses	2557	2984	3410	3836	4263
C-to-V ratio	4.2617	4.2629	4.2625	4.2622	4.2630
equation 5.2	4.2622	4.2623	4.2624	4.2625	4.2627

Table 5.3: We used UniformKSAT to generate 12 sets of random 3-SAT instances of different sizes, at clause-to-variable ratios computed as equation 5.2. This table summarises the numbers of variables and of clauses, and the clause-to-variable ratios.

Solver	Model		RMSE (support)	RMSE (challenge)
WalkSAT/SKC	Exp. Model	$6.89157 \times 10^{-4} \times 1.00798^{n}$	0.0008564	0.7600
	Poly. Model	$\mathbf{8.83962 \times 10^{-11} \times n^{3.18915}}$	**0.0007433**	**0.03142**
BalancedZ	Exp. Model	$1.32730 \times 10^{-3} \times 1.00759^{n}$	0.001759	1.081
	Poly. Model	$\mathbf{5.14258 \times 10^{-10} \times n^{2.97890}}$	**0.002870**	**0.05039**
probSAT	Exp. Model	$8.35877 \times 10^{-4} \times 1.00763^{n}$	0.0013867	0.6487
	Poly. Model	$\mathbf{2.92275 \times 10^{-10} \times n^{2.99877}}$	**0.002285**	**0.03301**

Table 5.4: Fitted models for median running times of SLS-based solvers solving phase-transition random 3-SAT instances and the corresponding RMSE values (in CPU sec). Model parameters are shown with 6 significant digits, and RMSEs with 4 significant digits; the models yielding more accurate predictions (as per RMSEs on challenge data) are shown in boldface.

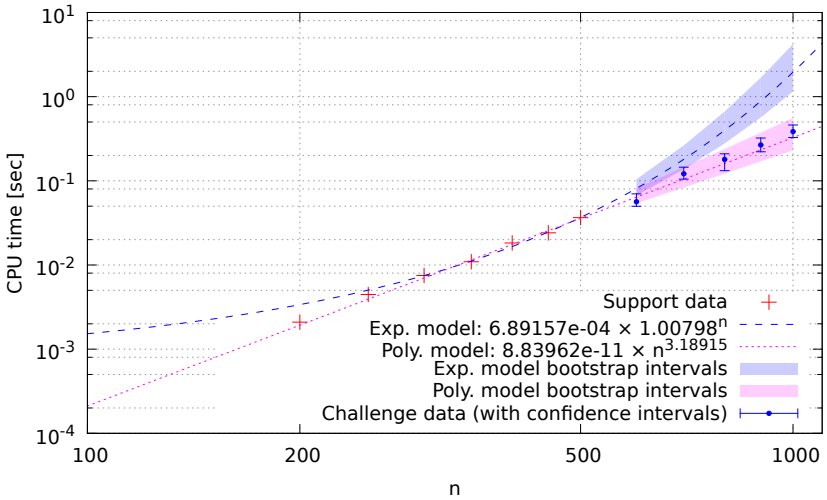


Figure 5.3: Fitted models for the median running times (in CPU-sec). Both models are fitted with the median running times of WalkSAT/SKC solving the satisfiable instances from the set of 1200 random 3-SAT instances of 200, 250, ..., 500 variables, and are challenged by median running times of 600, 700, ..., 1000 variables.

Solver	n	Observed median running time (sec)		Predicted confidence intervals (sec)	
		Point estimates	Confidence intervals	Poly. model	Exp. model
WalkSAT/SKC	600	0.056	[0.050, 0.070]	[**0.054**, **0.081**]	[0.067, 0.104]
	700	0.108	[0.093, 0.120]	[**0.083**, **0.146**]*	[0.137, 0.264]
	800	0.180	[0.132, 0.209]	[**0.122**, **0.238**]*	[0.277, 0.664]
	900	0.267	[0.222, 0.323]	[**0.170**, **0.373**]*	[0.565, 1.676]
	1000	0.385	[0.327, 0.461]	[**0.229**, **0.557**]*	[1.151, 4.200]
BalancedZ	600	0.095	[0.085, 0.102]	[**0.082**, **0.116**]*	[0.103, 0.149]
	700	0.142	[0.131, 0.154]	[**0.124**, **0.195**]*	[0.204, 0.348]
	800	0.194	[0.177, 0.212]	[**0.177**, **0.308**]*	[0.400, 0.816]
	900	0.270	[0.231, 0.324]	[**0.240**, **0.462**]	[0.782, 1.915]
	1000	0.353	[0.307, 0.398]	[**0.316**, **0.663**]	[1.531, 4.493]
probSAT	600	0.050	[0.043, 0.059]	[**0.050**, **0.085**]	[0.063, 0.110]
	700	0.089	[0.077, 0.105]	[**0.073**, **0.149**]*	[0.119, 0.271]
	800	0.151	[0.133, 0.197]	[**0.101**, **0.245**]*	[0.222, 0.664]
	900	0.237	[0.209, 0.295]	[**0.135**, **0.379**]*	[0.413, 1.640]
	1000	0.357	[0.304, 0.438]	[**0.174**, **0.559**]*	[0.771, 4.050]

Table 5.5: 95% bootstrap confidence intervals for observed and predicted running times of SLS-based solvers solving random 3-SAT instances. The instance sizes shown here are larger than those used for fitting the models. Bootstrap intervals on predictions that agree with the observed point estimates are shown in boldface, and those that fully contain the confidence intervals on observations are marked by asterisks (*).

Solver	Model	Confidence interval of a	Confidence interval of b
WalkSAT/SKC	Poly.	$[2.58600 \times 10^{-12}, 8.63869 \times 10^{-10}]$	[2.80816, 3.76751]
	Exp.	$[4.05064 \times 10^{-4}, 1.00662 \times 10^{-3}]$	[1.00709, 1.00924]
BalancedZ	Poly.	$[3.65984 \times 10^{-11}, 4.53094 \times 10^{-9}]$	[2.60985, 3.41689]
	Exp.	$[8.91856 \times 10^{-4}, 1.83333 \times 10^{-3}]$	[1.00675, 1.00855]
probSAT	Poly.	$[5.00843 \times 10^{-12}, 1.02411 \times 10^{-8}]$	[2.40567, 3.66266]
	Exp.	$[4.90478 \times 10^{-4}, 1.42905 \times 10^{-3}]$	[1.00629, 1.00907]

Table 5.6: Bootstrap intervals of model parameters for median running times of SLS-based solvers solving phase-transition random 3-SAT instances.

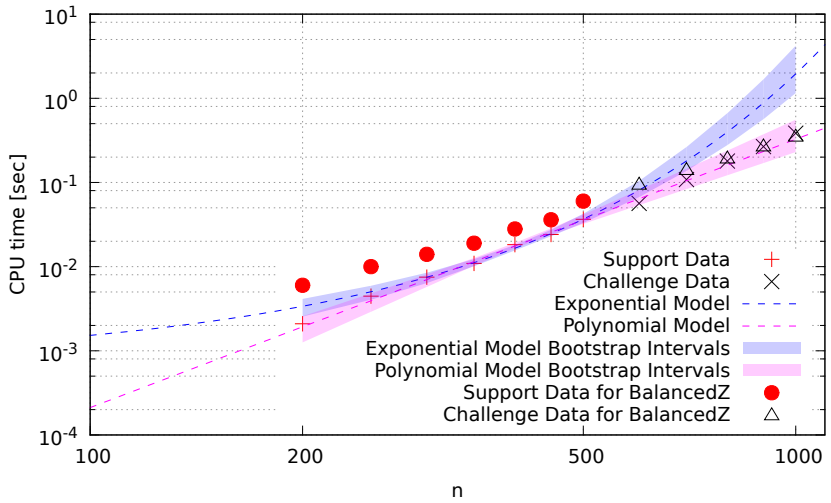


Figure 5.4: Comparing the scaling model for BalancedZ with that for WalkSAT/SKC on solving 3-SAT instances. Both models are fitted with the median running times of WalkSAT/SKC or BalancedZ solving 3-SAT instances of 200, 250, ..., 500 variables, and are challenged by median running times of 600, 700, ..., 1000 variables. Similar results can also been obtained by doing the comparison in the other way around.

further evidence consistent with the polynomial scaling models for all three SLS-based solvers (see Section 5.3.3). Although the running time of the SLS-based solvers are still quite moderate even at these instances sizes, the much longer runs required to filter out satisfiable instances with high confidence make it difficult to further increase n.

Next, we used the same bootstrap confidence intervals to investigate the significance of differences observed between the scaling models for different solvers. Using the approach described in Section 3.2, we cannot reject the hypothesis that the differences reflected in the constants for the polynomial models of our three SLS-based solvers seen in Table 5.4 are insignificant. The difference between WalkSAT/SKC and BalancedZ, for instance, is illustrated in Figure 5.4. Examining these results in detail, we believe that by substantially increasing the number of instances for each value of n, the bootstrap confidence intervals can likely be narrowed to also demonstrate significant scaling differences between all pairs of SLS-based solvers.

Finally, we investigated the question whether our observations regarding the qualitative differences in the scaling of median running time of SLS-based SAT solvers also hold when considering higher quantiles of the distribution of running

Solver	Model		RMSE (support)	RMSE (challenge)
WalkSAT/SKC	Exp. Model	$1.3165 \times 10^{-3} \times 1.0092^{n}$	0.002370	5.354
	Poly. Model	$\mathbf{6.5151 \times 10^{-12} \times n^{3.8148}}$	**0.002143**	**0.06216**
BalancedZ	Exp. Model	$3.0082 \times 10^{-3} \times 1.0075^{n}$	0.005848	2.024
	Poly. Model	$\mathbf{1.0385 \times 10^{-9} \times n^{2.9892}}$	**0.004009**	**0.05739**
probSAT	Exp. Model	$1.5357 \times 10^{-3} \times 1.0088^{n}$	0.004594	3.912
	Poly. Model	$\mathbf{2.1829 \times 10^{-11} \times n^{3.6122}}$	**0.004743**	**0.2951**

Table 5.7: Fitted models for 0.75-quantile of the running times of SLS-based solvers solving phase-transition random 3-SAT instances and the corresponding RMSE values (in CPU sec). Model parameters are shown with 5 significant digits and RMSEs with 4 significant digits; the models yielding more accurate predictions (as per RMSEs on challenge data) are shown in boldface.

times across instance sets. For these solvers, we noted weak evidence that the ratio between the higher quantiles and the median increases with n, but the observed scaling of the 0.75-quantile is still found to be consistent with a polynomial model and inconsistent with an exponential model. Details of the models, along with RMSEs on support and challenge data, are shown in Table 5.7. Note that for BalancedZ, the parameter b in the polynomial model for 0.75-quantiles does not increase much from that of the model for medians, which is in contrast to our findings for the other two solvers. The fitted models of WalkSAT/SKC are illustrated in Figure 5.5.

5.3.2 Empirical Scaling of the Performance of DPLL-based Solvers

Applying the same methodology, we studied the empirical scaling of the performance of DPLL-based solvers. For each DPLL-based solver, we used support data for $n = 200, 250, \ldots 400$ and challenge data for $n = 450, 500, 550$, as for even larger n, we could no longer complete sufficiently many runs to estimate even median running times. The fitted models and the corresponding RMSEs are shown in Table 5.8. We note that running times of the DPLL-based solvers are more consistent with our exponential scaling model – even when only considering performance on satisfiable instances. As an example, we illustrate the fitted models of kcnfs in Figure 5.6; similar results are obtained for the two march variants (figures not shown).

We also performed similar bootstrap analysis for DPLL-based solvers. The result of this analysis, presented in Table 5.9, shows opposite results for SLS-based solvers: the median running times of DPLL-based solvers, even solving satisfiable instances only, are consistent with exponential models but not polynomial ones. Also, this analysis gives us bootstrap confidence intervals for the model parameters,

Solver	Instances	Model		RMSE (support)	RMSE (challenge)
kcnfs	All	**Exp. Model**	$\mathbf{4.30400 \times 10^{-5} \times 1.03411^{n}}$	**0.05408**	**143.3**
		Poly. Model	$9.40745 \times 10^{-31} \times n^{12.1005}$	0.06822	1516
	Sat.	**Exp. Model**	$\mathbf{2.41708 \times 10^{-5} \times 1.03205^{n}}$	**0.02496**	**83.86**
		Poly. Model	$2.41048 \times 10^{-30} \times n^{11.7142}$	0.05600	225.8
	Unsat.	**Exp. Model**	$\mathbf{6.38367 \times 10^{-5} \times 1.03386^{n}}$	**0.001466**	**52.19**
		Poly. Model	$9.70804 \times 10^{-31} \times n^{12.1448}$	0.1813	2291
march_hi	All	**Exp. Model**	$\mathbf{4.93309 \times 10^{-5} \times 1.03295^{n}}$	**0.05449**	**460.0**
		Poly. Model	$1.05593 \times 10^{-30} \times n^{12.0296}$	0.05971	1266
	Sat.	**Exp. Model**	$\mathbf{8.33113 \times 10^{-6} \times 1.03119^{n}}$	**0.03035**	**3.087**
		Poly. Model	$2.44435 \times 10^{-30} \times n^{11.4789}$	0.03879	61.77
	Unsat.	**Exp. Model**	$\mathbf{7.86081 \times 10^{-5} \times 1.03281^{n}}$	**0.03336**	**399.7**
		Poly. Model	$2.10794 \times 10^{-30} \times n^{11.9828}$	0.16703	1912
march_br	All	**Exp. Model**	$\mathbf{6.17600 \times 10^{-5} \times 1.03220^{n}}$	**0.05401**	**402.4**
		Poly. Model	$5.56146 \times 10^{-30} \times n^{11.7408}$	0.05199	1253
	Sat.	**Exp. Model**	$\mathbf{1.02788 \times 10^{-5} \times 1.03048^{n}}$	**0.02497**	**13.72**
		Poly. Model	$1.25502 \times 10^{-29} \times n^{11.1944}$	0.03341	67.85
	Unsat.	**Exp. Model**	$\mathbf{6.10959 \times 10^{-5} \times 1.03344^{n}}$	**0.03230**	**262.8**
		Poly. Model	$5.18600 \times 10^{-31} \times n^{12.2154}$	0.1586	1920

Table 5.8: Fitted models for median running times of the DPLL-based solvers solving phase-transition random 3-SAT instances and the corresponding RMSE values (in CPU sec). Model parameters are shown with 6 significant digits, and RMSEs with 4 significant digits; the models yielding more accurate predictions (as per RMSEs on challenge data) are shown in boldface.

Solver	n	Observed median running time (sec)		Predicted confidence intervals (sec)	
		Point estimates	Confidence intervals	Poly. model	Exp. model
kcnfs	450	156.480	[143.340, 166.770]	[98.326, 122.115]	[**120.078**, **161.444**]
	500	750.510	[708.290, 806.130]	[327.997, 439.089]	[**561.976**, **889.428**]*
	550	3896.450	[3633.630, 4130.915]	[971.862, 1402.255]	[**2622.488**, **4901.661**]*
march_hi	450	112.553	[101.957, 121.167]	[62.021, 91.787]	[**74.982**, **116.729**]
	500	564.821	[508.433, 614.105]	[190.395, 333.963]	[**317.398**, **628.498**]*
	550	2971.450	[2660.430, 3152.570]	[523.034, 1074.244]	[**1342.438**, **3375.460**]*
march_br	450	112.812	[101.108, 121, 469]	[69.649, 91.290]	[**84.743**, **117.937**]
	500	594.095	[542.564, 620.963]	[226.874, 332.773]	[**385.943**, **640.179**]*
	550	2975.580	[2544.450, 3179.950]	[659.553, 1070.478]	[**1754.277**, **3492.830**]*

Table 5.9: 95% bootstrap confidence intervals for observed running times of DPLL-based solvers solving random 3-SAT instances. The instance sizes shown here are larger than those used for fitting the models. Bootstrap intervals on predictions that agree with the observed point estimates are shown in boldface, and those that fully contain the confidence intervals on observations are marked by asterisks (*).

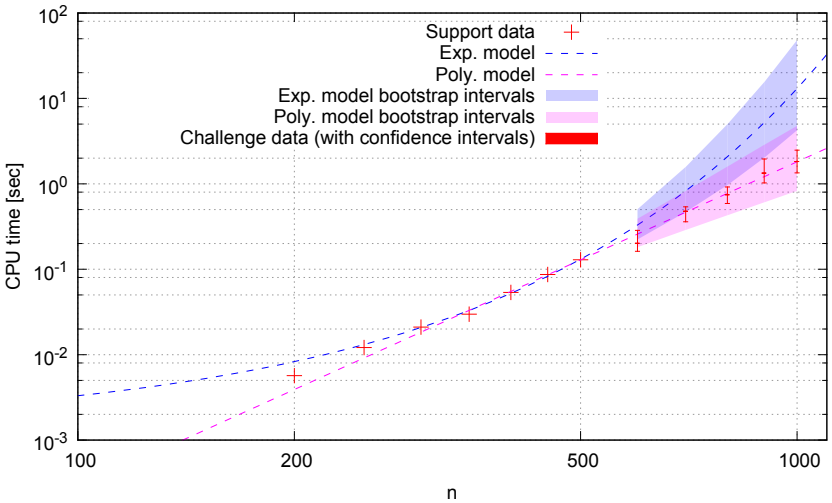


Figure 5.5: Fitted models for the 0.75-quantiles of the running times (in CPU-sec). Both models are fitted with the 0.75 quantiles of the running times of WalkSAT/SKC solving the satisfiable instances from the set of 1200 random 3-SAT instances of 200, 250, ..., 500 variables, and are challenged by median running times of 600, 700, ..., 1000 variables.

as shown in Table 5.10. Note that we found no evidence for a substantial change in the ratios between the median and higher quantiles as n increases; thus, there is no reason to assume any difference in the scaling models of a higher quantile and the median other than a constant factor.

Additionally, we investigated whether the differences between the scaling models of DPLL-based solvers are significant using the approach described in Section 3.2. We found that the bootstrap confidence intervals for march_hi and march_br largely overlap with each other, and we cannot detect sufficient evidence for statistically significant differences in scaling behaviour. On the other hand, the observed running times for kcnfs are inconsistent with the scaling models for march_hi and march_br, confirming that the performance of the march solvers indeed scales significantly better than that of kcnfs. (See Figure 5.7 for an illustration of the comparison of the models for kcnfs and march_hi.)

Using the same approach, we investigated the significance of the differences in scaling behaviour for each DPLL-based solver when applied to satisfiable and unsatisfiable instances only, respectively. Intuitively, we would expect unsatisfiable instances to be harder, and the differences between the respective scaling models are significant, in that the observed running times for solving unsatisfiable instances are generally inconsistent with the scaling model for the same solver on

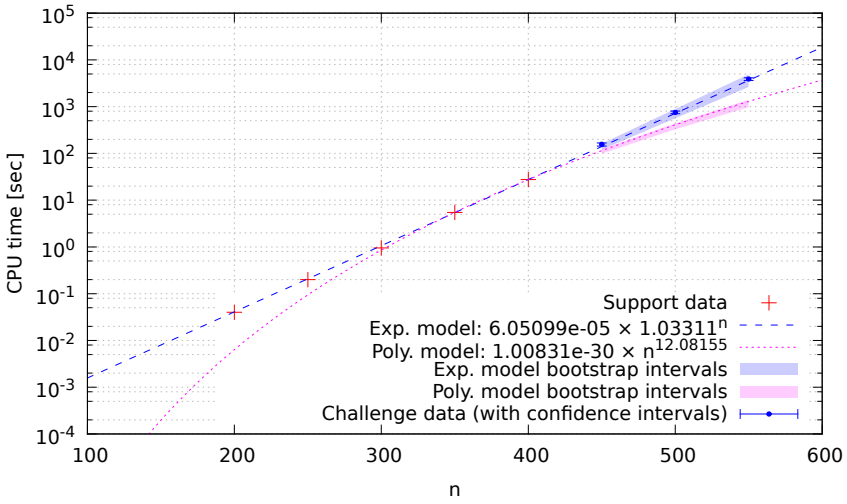


Figure 5.6: Fitted models for the median running times of kcnfs [CPU-sec]. Both models are fitted with the median running times of kcnfs solving 1200 random 3-SAT instances of 200, 250, ..., 400 variables, and are challenged by median running times of 450, 500 and 550 variables.

only satisfiable instances (as illustrated in Figure 5.8 for march_hi). To further investigate whether the performance difference can be attributed to the constant factor a in the exponential models, we fitted a one-parameter model of the form $a \cdot b_{sat}^n$ of the median running times for solving unsatisfiable instances, where a is a free parameter and b_{sat} is the value of parameter b from the scaling model for satisfiable instances. For all three DPLL-based solvers, the observed running times for solving unsatisfiable instances is consistent with these one-parameter models, suggesting that, indeed, the scaling on satisfiable and unsatisfiable instances differs only by a constant factor of around 20.

We also experimented with sat11, the running times of which are highly consistent with exponential models but not with polynomial ones. The exponent is similar to those of other DPLL-based solvers, but the constant factor is much larger (potentially due to implementation details of the solver itself).

5.3.3 Challenging the Scaling Models of SAT Solvers with Larger Instances

We further challenged our models with runs of solvers of larger instances, the sizes of which match those used in the most recent SAT Competitions. Due to the cost

Solver	Instances	Model	Confidence interval of a	Confidence interval of b
kcnfs	All	Poly.	$[3.33969 \times 10^{-31}, 4.30846 \times 10^{-29}]$	$[11.4234, 12.2674]$
		Exp.	$[3.33378 \times 10^{-5}, 1.07425 \times 10^{-4}]$	$[1.03136, 1.03476]$
	Sat.	Poly.	$[1.81751 \times 10^{-36}, 2.47924 \times 10^{-21}]$	$[8.19161, 14.0613]$
		Exp.	$[2.02817 \times 10^{-6}, 5.85540 \times 10^{-4}]$	$[1.02283, 1.03835]$
	Unsat.	Poly.	$[5.65987 \times 10^{-32}, 5.28138 \times 10^{-30}]$	$[11.8558, 12.6251]$
		Exp.	$[4.22382 \times 10^{-5}, 1.03613 \times 10^{-4}]$	$[1.03252, 1.03508]$
march_hi	All	Poly.	$[1.72213 \times 10^{-31}, 3.51322 \times 10^{-27}]$	$[10.6379, 12.3284]$
		Exp.	$[2.90480 \times 10^{-5}, 1.72479 \times 10^{-4}]$	$[1.02928, 1.03433]$
	Sat.	Poly.	$[5.68054 \times 10^{-32}, 1.93360 \times 10^{-24}]$	$[9.16765, 12.1075]$
		Exp.	$[7.97341 \times 10^{-7}, 7.07414 \times 10^{-5}]$	$[1.02521, 1.03765]$
	Unsat.	Poly.	$[5.01545 \times 10^{-31}, 1.43762 \times 10^{-29}]$	$[11.6557, 12.2263]$
		Exp.	$[5.51043 \times 10^{-5}, 1.06014 \times 10^{-4}]$	$[1.03195, 1.03386]$
march_br	All	Poly.	$[1.26357 \times 10^{-31}, 1.79577 \times 10^{-28}]$	$[11.1507, 12.3819]$
		Exp.	$[2.61030 \times 10^{-5}, 1.08165 \times 10^{-4}]$	$[1.03064, 1.03466]$
	Sat.	Poly.	$[4.33022 \times 10^{-32}, 1.53387 \times 10^{-24}]$	$[9.10489, 12.1426]$
		Exp.	$[1.81911 \times 10^{-6}, 6.40234 \times 10^{-5}]$	$[1.02515, 1.03519]$
	Unsat.	Poly.	$[4.36438 \times 10^{-31}, 6.51380 \times 10^{-30}]$	$[11.7848, 12.2416]$
		Exp.	$[4.27173 \times 10^{-5}, 9.18950 \times 10^{-5}]$	$[1.03230, 1.03443]$

Table 5.10: Bootstrap intervals of model parameters for median running times of DPLL-based solvers solving phase-transition random 3-SAT instances.

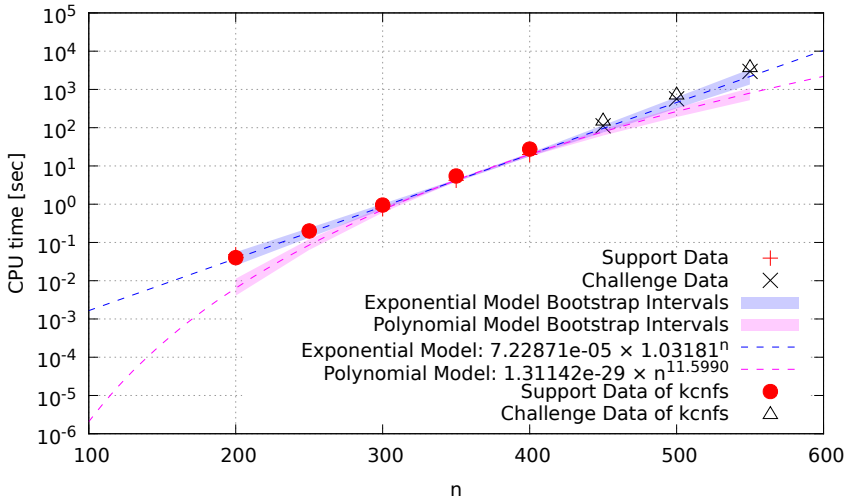


Figure 5.7: Comparing scaling models of kcnfs and march_hi on solving 3-SAT instances. Both models are fitted with the median running times of kcnfs or march_hi solving 1200 3-SAT instances of 200, 250, ..., 400 variables, and are challenged by median running times of 450, 500 and 550 variables.

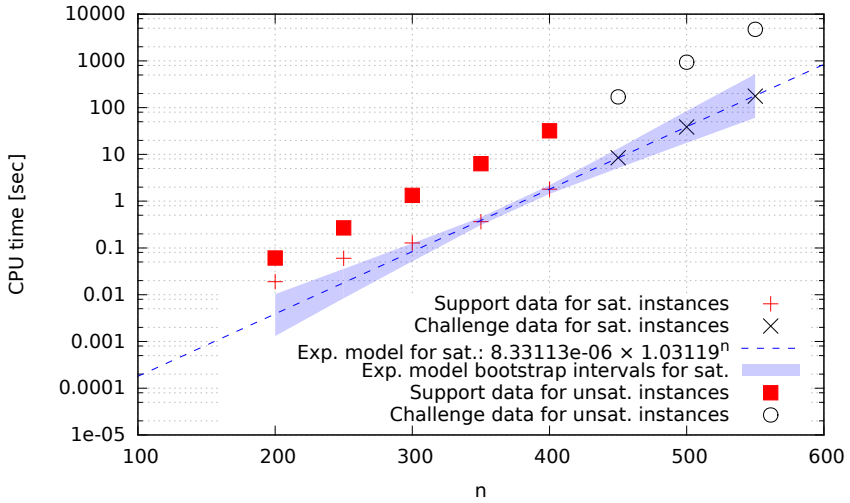


Figure 5.8: Scaling of the median running time of march_hi on satisfiable *vs* unsatisfiable 3-SAT instances. We show the scaling model and bootstrap confidence intervals for performance on satisfiable 3-SAT instances, along with observations on support and challenge data for satisfiable and unsatisfiable instances.

of solving these instances, we only used a small number of these. For SLS-based solvers, we generated 120 instances of sizes 1500 and 2000 as well as 11 instances of sizes 5000 and 10000. For these instances, 66, 58, 5 and 7 instances were solved respectively, with a cut-off of 2 CPU days. Note that, WalkSAT/SKC failed to solve one $n = 5000$ instance and five $n = 10000$ instances that BalancedZ managed to solve, and probSAT missed one instance of each size. For DPLL-based solvers, we generated 120 instances with $n = 650$, 67 our of which are satisfiable and 53 are unsatisfiable. Note that about half of the march_hi and march_br runs produced SIGSEGV errors when solving these larger instances, and we did not further analyse them with larger instances.

The detailed results are shown in Table 5.11. From these results, we see evidence that the exponential model remains a good descriptor of the scaling of kcnfs for the large $n = 650$ instances. For the SLS-based solvers, the predictions made by the polynomial models are also well consistent with the observed running times, except for WalkSAT/SKC solving $n = 10000$ instances. We have not found concrete explanations for this exception, but believe that it may result from implementation details of the solver. Nevertheless, our results are striking, considering that we fitted our polynomial models on instances with $n \leq 500$ and these models remain valid, even when challenged with 10 times larger instances that are challenging for modern SAT solvers.

Solver	n	Predicted Confidence intervals	Observed median running time (sec) Point estimate	Confidence intervals
WalkSAT/SKC	1500	[**0.721**, **2.569**]#	1.775	[1.135, 5.286]
	2000	[**1.641**, **7.571**]	12.002	[4.969, 32.103]
	5000	[**20.48**, **217.6**]#	149.6	[94.91, 331.9]
BalancedZ	1500	[**0.906**, **2.653**] #	1.081	[0.541, 2.586]
	2000	[**1.915**, **7.068**] #	3.465	[1.485, 6.383]
	5000	[**23.72**, **185.0**]	18.07	[4.515, 4077]
	10000	[**149.9**, **2072**]#	1064	[224.1, 20475]
probSAT	1500	[**0.488**, **2.755**]	2.820	[1.335, 5.745]
	2000	[**0.989**, **8.129**]	19.38	[3.740, 37.944]
	5000	[**9.387**, **257.527**]#	12.75	[10.329, ∞]
	10000	[**51.502**, **3546.870**]	11377	[1326, ∞]
kcnfs	650, All	[**57720**, **149011**]*	108071	[62308, 126999]
	650, Sat.	[**1285**, **96993**]*	26370.6	[11877, 44980]
	650, Unsat.	[**111920**, **228028**]*	159747	[135445, 191781]

Table 5.11: Challenging the models with running times for large 3-SAT instances. For the bootstrap intervals of the predicted median running times, those overlapping with the observed confidence intervals are in boldface, those that agree with the observed point estimates are marked by #, and those agreeing with the observed confidence intervals are followed by *.

5.4 Empirical Scaling of Solver Performance on Random 4-SAT

We next studied two distributions of random 4-SAT instances: phase-transition instances (as discussed in Section 5.2.2) and a class of less constrained instances proposed by Dimitris Achlioptas.

5.4.1 Empirical Scaling of Solver Performance on Random 4-SAT at Phase Transition

Similar to the analysis performed for random 3-SAT, we analysed the scaling performance of the same set of SAT solvers (WalkSAT/SKC, BalancedZ, probSAT, kcnfs, march_hi and march_br) on phase-transition random 4-SAT as well. For SLS-based solvers, we used $n = 70, 80, \cdots 150$ as the support data and $n = 160, 170, \cdots 200, 225, \cdots, 300$ as the challenge data; while for DPLL-based solvers, we used $n = 80, 90, \cdots, 130$ as the support data and $n = 140, 150, \cdots, 190$ as the challenge data. The sizes were chosen considering the running times of the solvers and our budget of computational resources, and we roughly used half of the sizes for model fitting and the other half for challenging the obtained models. To reduce floor effects, we made sure that median running times for SLS-based

	Model		RMSE (support)	RMSE (challenge)
WalkSAT/SKC	Exp. Model	$6.9784 \times 10^{-5} \times 1.0456^{n}$	0.00084331	11.175
	RootExp. Model	$\mathbf{2.0584 \times 10^{-7} \times 2.777^{\sqrt{n}}}$	**0.0011903**	**1.1389**
	Poly. Model	$9.6674 \times 10^{-15} \times n^{5.8626}$	0.0015716	3.5491
BalancedZ	**Exp. Model**	$\mathbf{1.1841 \times 10^{-3} \times 1.0208^{n}}$	**0.00077**	**0.03889**
	RootExp. Model	$1.0763 \times 10^{-4} \times 1.5631^{\sqrt{n}}$	0.00089	0.08843
	Poly. Model	$1.473 \times 10^{-7} \times n^{2.4042}$	0.00107	0.13516
probSAT	Exp. Model	$8.7936 \times 10^{-4} \times 1.0208^{n}$	0.00071	0.06340
	RootExp. Model	$\mathbf{8.082 \times 10^{-5} \times 1.5616^{\sqrt{n}}}$	**0.00087**	**0.03013**
	Poly. Model	$1.1498 \times 10^{-7} \times n^{2.3939}$	0.00104	0.06473

Table 5.12: Fitted models for median running times of the SLS-based solvers solving phase-transition random 4-SAT instances and the corresponding RMSE values (in CPU sec). The models yielding the most accurate predictions (as per RMSEs on challenge data) are shown in boldface.

Solver	Model	Confidence interval of a	Confidence interval of b
WalkSAT/SKC	Poly.	$[8.3451 \times 10^{-17}, 7.7952 \times 10^{-13}]$	$[4.7239, 6.7928]$
	RootExp.	$[3.375 \times 10^{-8}, 1.1002 \times 10^{-6}]$	$[2.308, 3.2189]$
	Exp.	$[2.9942 \times 10^{-5}, 0.00014855]$	$[1.0377, 1.0518]$
BalancedZ	Poly.	$[1.9088 \times 10^{-8}, 4.2697 \times 10^{-7}]$	$[2.1762, 2.8316]$
	RootExp.	$[4.7598 \times 10^{-5}, 1.6375 \times 10^{-4}]$	$[1.5002, 1.6841]$
	Exp.	$[7.9832 \times 10^{-4}, 1.4443 \times 10^{-3}]$	$[1.0189, 1.0242]$
probSAT	Poly.	$[2.4434 \times 10^{-8}, 3.9394 \times 10^{-7}]$	$[2.1333, 2.7203]$
	RootExp.	$[4.3564 \times 10^{-5}, 1.3324 \times 10^{-4}]$	$[1.4899, 1.6541]$
	Exp.	$[6.5415 \times 10^{-4}, 1.1207 \times 10^{-3}]$	$[1.0186, 1.0234]$

Table 5.13: Bootstrap intervals of model parameters for median running times of SLS-based solvers solving phase-transition random 4-SAT instances.

solvers are above 0.005 CPU sec, and those for DPLL-based solvers above 0.1 CPU sec.

We first fitted parametric models to the median running times. Preliminary experiments showed that polynomial and exponential models do not accurately capture the scaling behaviour of the SLS-based solvers. Thus, as mentioned in Section 5.1, we included root-exponential models to better characterise their scaling behaviour. The fitted models and corresponding RMSE values for SLS-based solvers are shown in Table 5.12. We also performed bootstrap analysis as described in Section 3.1 and computed bootstrap confidence intervals for the model parameters, observed and predicted running times, which are presented in Tables 5.13, 5.14 and 5.15, respectively. Our results show that exponential (for BalancedZ) or root-exponential (for the other solvers) models usually fit the running times of SLS-based solvers better, which stands in contrast to random 3-SAT. Figure 5.9, for instance, shows the fitted models and corresponding bootstrap confidence intervals

Solver	n	Observed median running time (sec)	
		Point estimates	Confidence intervals
WalkSAT/SKC	160	0.07675	[0.06683, 0.09427]
	170	0.1112	[0.09516, 0.1296]
	180	0.1661	[0.1439, 0.2145]
	190	0.2111	[0.1773, 0.252]
	200	0.3488	[0.2739, 0.3967]
	225	0.7695	[0.5895, 1.055]
	250	2.306	[1.936, 3.239]
	275	5.159	[4.166, 7.331]
	300	13.29	[8.837, 17.68]
BalancedZ	160	0.03399	[0.02849, 0.03878]
	170	0.03899	[0.03447, 0.04252]
	180	0.04849	[0.04247, 0.05478]
	190	0.06099	[0.04947, 0.07156]
	200	0.08699	[0.06646, 0.1016]
	225	0.106	[0.08699, 0.1492]
	250	0.209	[0.186, 0.2967]
	275	0.283	[0.232, 0.3613]
	300	0.4724	[0.3586, 0.6418]
probSAT	160	0.02499	[0.022, 0.02852]
	170	0.02899	[0.027, 0.03452]
	180	0.03099	[0.02899, 0.03499]
	190	0.03899	[0.03347, 0.04652]
	200	0.04699	[0.03999, 0.05751]
	225	0.06599	[0.05641, 0.07851]
	250	0.118	[0.09241, 0.136]
	275	0.166	[0.128, 0.209]
	300	0.2615	[0.17, 0.3467]

Table 5.14: 95% bootstrap confidence intervals for observed running times of SLS-based solvers solving phase-transition random 4-SAT instances. The instance sizes shown here are larger than those used for fitting the models.

Solver	n	Predicted confidence intervals of median running time (sec)		
		Poly. model	RootExp. model	Exp. model
WalkSAT/SKC	160	[**0.06396**, **0.09442**]*	[**0.06672**, **0.09797**]*	[**0.06985**, **0.1021**]#
	170	[**0.08559**, **0.1412**]*	[**0.09274**, **0.153**]*	[**0.1013**, **0.1676**]#
	180	[**0.1121**, **0.2077**]#	[**0.1272**, **0.2377**]*	[**0.1466**, **0.2776**]#
	190	[**0.1447**, **0.2971**]*	[**0.173**, **0.3618**]*	[**0.2123**, **0.4554**]
	200	[**0.1844**, **0.42**]*	[**0.2334**, **0.5483**]*	[**0.3074**, **0.7519**]#
	225	[**0.3217**, **0.9324**]#	[**0.4784**, **1.49**]*	[**0.7752**, **2.645**]
	250	[0.5291, 1.907]	[**0.943**, **3.848**]*	[**1.955**, **9.34**]#
	275	[0.83, 3.644]	[**1.798**, **9.486**]*	[**4.932**, **32.98**]#
	300	[1.252, 6.581]	[**3.332**, **22.46**]*	[**12.44**, **116.5**]#
BalancedZ	160	[**0.02644**, **0.03333**]	[**0.0275**, **0.0349**]#	[**0.02862**, **0.03662**]#
	170	[**0.03021**, **0.03962**]#	[**0.03225**, **0.04282**]*	[**0.03455**, **0.0466**]#
	180	[**0.03425**, **0.04663**]	[**0.03764**, **0.05223**]#	[**0.04173**, **0.05931**]*
	190	[**0.03857**, **0.05441**]	[**0.04374**, **0.06337**]#	[**0.05039**, **0.07523**]#
	200	[0.04317, 0.06298]	[**0.05064**, **0.07626**]	[**0.06086**, **0.09537**]#
	225	[**0.05592**, **0.08773**]	[**0.07192**, **0.119**]#	[**0.09753**, **0.1728**]#
	250	[0.07048, 0.118]	[0.1001, 0.1812]	[**0.1561**, **0.313**]*
	275	[0.08678, 0.1543]	[**0.1371**, **0.2704**]	[**0.2495**, **0.5671**]#
	300	[0.1049, 0.1971]	[**0.1849**, **0.3968**]	[**0.3985**, **1.03**]#
probSAT	160	[**0.01968**, **0.02449**]	[**0.0205**, **0.02566**]#	[**0.02136**, **0.02694**]#
	170	[**0.02242**, **0.02885**]	[**0.02397**, **0.03119**]#	[**0.02574**, **0.03394**]#
	180	[**0.02535**, **0.03368**]#	[**0.02791**, **0.03769**]*	[**0.03105**, **0.04275**]
	190	[**0.02848**, **0.03899**]	[**0.03239**, **0.04532**]#	[**0.03731**, **0.05385**]#
	200	[**0.03183**, **0.0448**]	[**0.03733**, **0.05423**]#	[**0.04482**, **0.06784**]#
	225	[**0.0409**, **0.06161**]	[**0.05244**, **0.08337**]*	[**0.07101**, **0.1208**]
	250	[0.05113, 0.08195]	[**0.07232**, **0.1252**]#	[**0.1129**, **0.2152**]#
	275	[0.06258, 0.1061]	[**0.09844**, **0.1843**]#	[**0.1794**, **0.3833**]
	300	[0.07526, 0.1342]	[**0.1322**, **0.2668**]#	[**0.285**, **0.684**]

Table 5.15: 95% bootstrap confidence intervals for predicted median running time of SLS-based solvers solving phase-transition random 4-SAT instances. The instance sizes shown here are larger than those used for fitting the models. Bootstrap intervals on predictions that overlap with the corresponding bootstrap interval on observed data are shown in boldface, those that agree with the observed point estimates are marked by sharps (#), and those that fully contain the confidence intervals on observations are marked by asterisks (*).

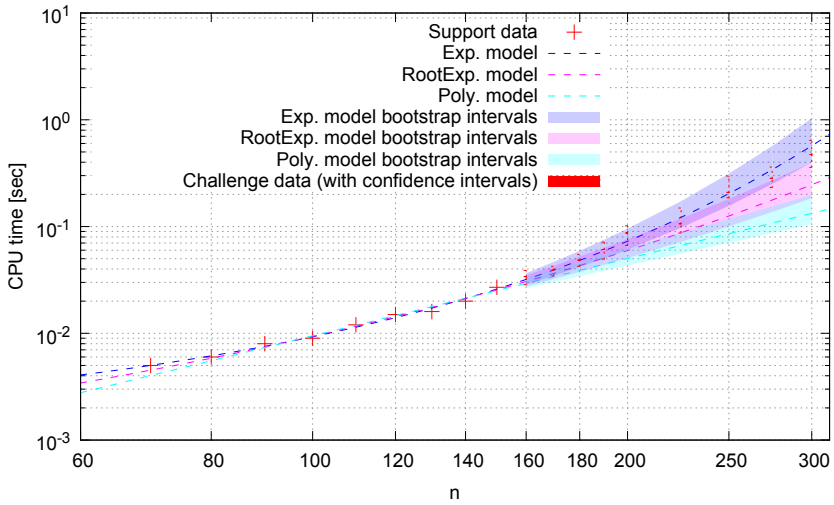


Figure 5.9: Fitted models for median running times of BalancedZ. Both models are fitted to the median running times of BalancedZ solving the 4-SAT instances from the set of phase-transition random instances with 70, 80, ... 150 variables, and are challenged by median running times for instances with 160, 170, ... 200, 225, ... 300 variables.

for BalancedZ.

On the other hand, results for DPLL-based solvers on random 4-SAT are very similar to those on 3-SAT, with exponential models fitting the running times very well. Details of the fitted models and corresponding RMSE values can be found in Table 5.16. Note that we did not present result for polynomial models, because they clearly under-estimate the scaling of the running times, and the Levenberg-Marquardt Algorithm takes extremely long time to fit such models. We again performed bootstrap analysis and obtained the bootstrap confidence intervals for model parameters and for observed and predicted running times shown in Tables 5.17 and 5.18, respectively. Figure 5.10 illustrates the fitted models and corresponding bootstrap confidence intervals for march_hi.

5.4.2 Empirical Scaling of Solver Performance on Less-constrained Random 4-SAT

We also explored less-constrained 4-SAT instances generated according to a formula proposed by Dimitris Achlioptas, namely

$$m = 2^{k-1} \cdot n \tag{5.4}$$

Solver	Model		RMSE (support)	RMSE (challenge)
kcnfs	**Exp. Model**	$\mathbf{1.9621 \times 10^{-5} \times 1.1098^{n}}$	**0.02472**	**689.0**
	RootExp. Model	$5.8865 \times 10^{-11} \times 10.008^{\sqrt{n}}$	0.06817	1085
march_hi	**Exp. Model**	$\mathbf{5.4972 \times 10^{-5} \times 1.1006^{n}}$	**0.01430**	**333.02**
	RootExp. Model	$4.8241 \times 10^{-10} \times 8.285^{\sqrt{n}}$	0.06351	1318
march_br	**Exp. Model**	$\mathbf{3.272 \times 10^{-5} \times 1.1045^{n}}$	**0.07124**	**123.7**
	RootExp. Model	$1.813 \times 10^{-10} \times 8.9758^{\sqrt{n}}$	0.11317	1034

Table 5.16: Fitted models for median running times of the DPLL-based solvers solving phase-transition random 4-SAT instances and the corresponding RMSE values (in CPU sec). The models yielding more accurate predictions (as per RMSEs on challenge data) are shown in boldface.

Solver	Model	Confidence interval of a	Confidence interval of b
kcnfs	Exp.	$[1.1077 \times 10^{-5}, 3.4453 \times 10^{-5}]$	$[1.1047, 1.1149]$
	RootExp.	$[1.8193 \times 10^{-11}, 1.9247 \times 10^{-10}]$	$[8.9979, 11.116]$
march_hi	Exp.	$[2.2428 \times 10^{-5}, 9.0675 \times 10^{-5}]$	$[1.0961, 1.1082]$
	RootExp.	$[7.9199 \times 10^{-11}, 1.3699 \times 10^{-9}]$	$[7.5342, 9.701]$
march_br	Exp.	$[9.033 \times 10^{-6}, 5.9334 \times 10^{-5}]$	$[1.0994, 1.1156]$
	RootExp.	$[1.3416 \times 10^{-11}, 6.0806 \times 10^{-10}]$	$[8.0647, 11.323]$

Table 5.17: Bootstrap intervals of model parameters for median running times of DPLL-based solvers solving phase-transition random 4-SAT instances.

For WalkSAT/SKC, we used $n = 6\,000, 7\,000, \cdots, 15\,000$ as support data, and $n = 16\,000, 17\,000, \cdots, 20\,000, 30\,000, 40\,000$ as challenge data. For kcnfs, we used $n = 300, 350, \cdots, 450$ as support data, and $n = 500, 550, 600$ as challenge data. The sizes were chosen considering the running times of the solvers and our budget of computational resources, and we roughly used half of the sizes for model fitting and the other half for challenging the obtained models. To reduce floor effects, we made sure that median running times for SLS-based solvers are above 0.005 CPU sec, and those for DPLL-based solvers above 0.1 CPU sec. The fitted models and corresponding RMSE values are shown in Table 5.19. Results of bootstrap analysis, including confidence intervals for model parameters, for observed running times and for predicted values, are shown in Tables 5.20, 5.21 and 5.22, respectively. Figure 5.11 illustrates the fitted models of WalkSAT/SKC, which suggests some scaling behaviour lower-bounded by, and close to, polynomial. For kcnfs, the fitted models are shown in Figure 5.12, and the observed running times are surprisingly well consistent with the polynomial model.

To summarise, both WalkSAT/SKC and kcnfs demonstrate significantly better scaling behaviour for solving the less-constrained 4-SAT instances than for phase-transition instances. This confirms earlier observations that phase-transition SAT

Solver	n	Observed median running time (sec)		Predicted confidence intervals (sec)	
		Point estimates	Confidence intervals	RootExp. model	Exp. model
kcnfs	140	39.39	[37.06, 41.39]	[**36.81**, **42.96**]*	[**38.74**, **45.25**]#
	150	115.2	[108.5, 119.9]	[**91.8**, **116.9**]#	[**105**, **134.3**]*
	160	319.6	[306.1, 327.3]	[**221.9**, **306.9**]	[**284.1**, **398**]*
	170	839.7	[783.9, 874.3]	[**521.8**, **785.3**]	[**768.8**, **1183**]*
	180	2331	[2242, 2448]	[1197, 1955]	[**2081**, **3519**]*
	190	6151	[5745, 6447]	[2686, 4749]	[**5632**, $\mathbf{1.046 \times 10^4}$]*
march_hi	140	36.54	[32.77, 39.18]	[**31.81**, **39.33**]*	[**33.43**, **41.37**]#
	150	101.6	[95.02, 105.3]	[**73.85**, **101.8**]#	[**83.98**, **116.5**]*
	160	278.8	[263.4, 291.4]	[166.8, 254.3]	[**211**, **326.4**]*
	170	759.6	[710.9, 803.5]	[367.3, 615.5]	[**530.1**, **910.3**]*
	180	2070	[1965, 2157]	[790.7, 1449]	[**1332**, **2535**]*
	190	5227	[4887, 5537]	[1667, 3331]	[**3346**, **7059**]*
march_br	140	32.6	[28.06, 35.07]	[**31.9**, **38.24**]#	[**33.53**, **40.23**]
	150	94.26	[85.39, 99.02]	[**76.18**, **104.4**]*	[**86.79**, **119.8**]#
	160	261.1	[251.2, 272.4]	[**176.8**, **277.8**]*	[**224.7**, **359.2**]*
	170	714.1	[686.2, 748.8]	[**399.3**, **717**]#	[**580.3**, **1077**]*
	180	1868	[1774, 1962]	[**880**, **1800**]	[**1498**, **3227**]*
	190	4892	[4529, 5237]	[1898, 4396]	[**3865**, **9646**]*

Table 5.18: 95% bootstrap confidence intervals for observed running times of DPLL-based solvers solving phase-transition random 4-SAT instances. The instance sizes shown here are larger than those used for fitting the models. Bootstrap intervals on predictions that overlap with the corresponding bootstrap interval on observed data are shown in boldface, those that agree with the observed point estimates are marked by sharps (#), and those that fully contain the confidence intervals on observations are marked by asterisks (*).

Solver	Model		RMSE (support)	RMSE (challenge)
WalkSAT/SKC	Exp. Model	0.02346×1.00012^n	0.00301	0.76778
	Poly. Model	$\mathbf{2.97526 \times 10^{-7} \times n^{1.35594}}$	**0.00139**	**0.08560**
kcnfs	Exp. Model	$2.47586 \times 10^{-5} \times 1.02815^n$	0.02313	137.183
	Poly. Model	$\mathbf{9.44803 \times 10^{-31} \times n^{11.625}}$	**0.03740**	**2.91674**

Table 5.19: Fitted models for median running times of the solvers solving less-constrained random 4-SAT instances and corresponding RMSE values (in CPU sec). The models yielding more accurate predictions (as per RMSEs on challenge data) are shown in boldface.

Solver	Model	Confidence interval of a	Confidence interval of b
WalkSAT/SKC	Exp.	$[0.02190, 0.02570]$	$[1.00011, 1.00012]$
	Poly.	$[2.12689 \times 10^{-7}, 4.36148 \times 10^{-7}]$	$[1.31430, 1.39195]$
kcnfs	Exp.	$[3.47927 \times 10^{-6}, 0.00012]$	$[1.02419, 1.03295]$
	Poly.	$[3.92302 \times 10^{-36}, 3.17454 \times 10^{-26}]$	$[9.91562, 13.6855]$

Table 5.20: Bootstrap intervals of model parameters for median running time of the solvers solving less-constrained random 4-SAT instances.

Solver	n	Observed median running time (sec)	
		Point estimates	Conf. intervals
WalkSAT/SKC	16000	0.1404	[0.1387, 0.1422]
	17000	0.1540	[0.1521, 0.1558]
	18000	0.1697	[0.1671, 0.1717]
	19000	0.1862	[0.1841, 0.1882]
	20000	0.2040	[0.2014, 0.2075]
	30000	0.4363	[0.4234, 0.4546]
	40000	0.7262	[0.7113, 0.7355]
kcnfs	500	21.3758	[17.8882, 24.9521]
	550	72.2845	[55.7475, 89.4627]
	600	189.071	[153.728, 239.913]

Table 5.21: 95% bootstrap confidence intervals for observed running times on less-constrained random 4-SAT instances. The instance sizes shown here are larger than those used for fitting the models..

Solver	n	Predicted confidence intervals (sec)	
		Exp. model	Poly. model
WalkSAT/SKC	16000	[0.1510, 0.1617]	[0.1462, 0.1519]
	17000	[0.1687, 0.1832]	[0.1583, 0.1652]
	18000	[0.1884, 0.2075]	[**0.1707**, **0.1789**]
	19000	[0.2105, 0.2351]	[**0.1833**, **0.1929**]*
	20000	[0.2351, 0.2663]	[**0.1961**, **0.2072**]#
	30000	[0.7111, 0.9267]	[0.3341, 0.3637]
	40000	[2.1504, 3.2241]	[0.4876, 0.5423]
kcnfs	500	[**18.7603**, **39.0276**]#	[**16.1396**, **32.6376**]*
	550	[**61.7576**, **189.54**]#	[**41.1617**, **115.85**]*
	600	[**208.907**, **957.263**]	[**99.4254**, **379.376**]*

Table 5.22: 95% bootstrap confidence intervals for predicted running times on less-constrained random 4-SAT instances. The instance sizes shown here are larger than those used for fitting the models. Bootstrap intervals on predictions that overlap with the corresponding bootstrap interval on observed data are shown in boldface, those that agree with the observed point estimates are marked by sharps (#), and those that fully contain the confidence intervals on observations are marked by asterisks (*).

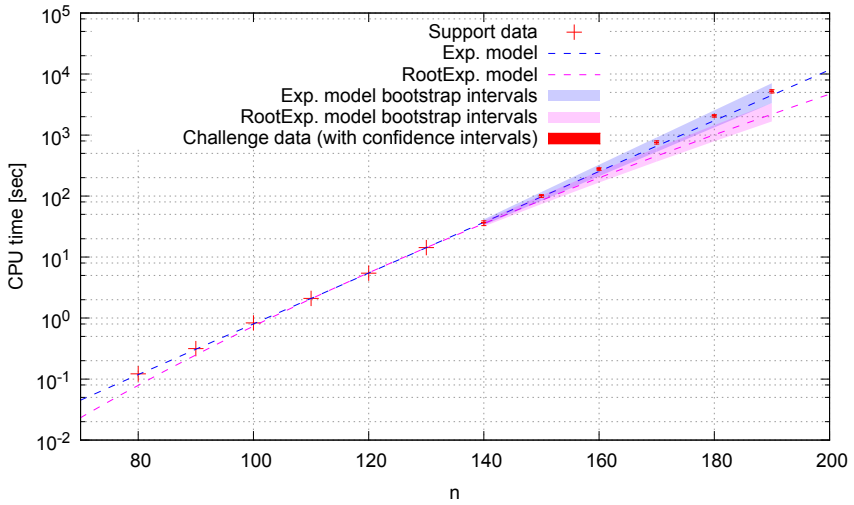


Figure 5.10: Fitted models for median running times of march_hi. Both models are fitted with the median running times of march_hi solving the 4-SAT instances from the set of phase-transition random instances with 40, 50, ... 120 variables, and are challenged by median running times for instances with 130, 140, ... 190 variables.

instances represent a very hard distribution, and the less-constrained instances we studied here are significantly easier than phase-transition 4-SAT instances.

5.5 Experiments for the Survey Propagation Algorithm

We also tried to experiment with the survey propagation algorithm [11], but encountered some practical problems with the implementation made available by the authors[4]. In particular, the solver turns out to be unable to solve a significant proportion of satisfiable phase-transition random 3-SAT instances for $n = 200, \cdots, 1000$. In our experiments, 5 runs were performed for each instance and we report the number of instances that cannot be solved in at least 3 out of the 5 runs in Table 5.23. It is clear that the proportion of failed runs is quite large and tends to increase with n. For $n = 1000$, more than 70% of the instances could not be solved in 3 out of 5 runs. To further evaluate whether we could restart the algorithm to make it a practical solver for satisfiable instances, we performed 1000 independent runs on 3 instances, and none of these instances was ever solved in any single run of the algorithm.

[4] http://areeweb.polito.it/ricerca/cmp/code/sp

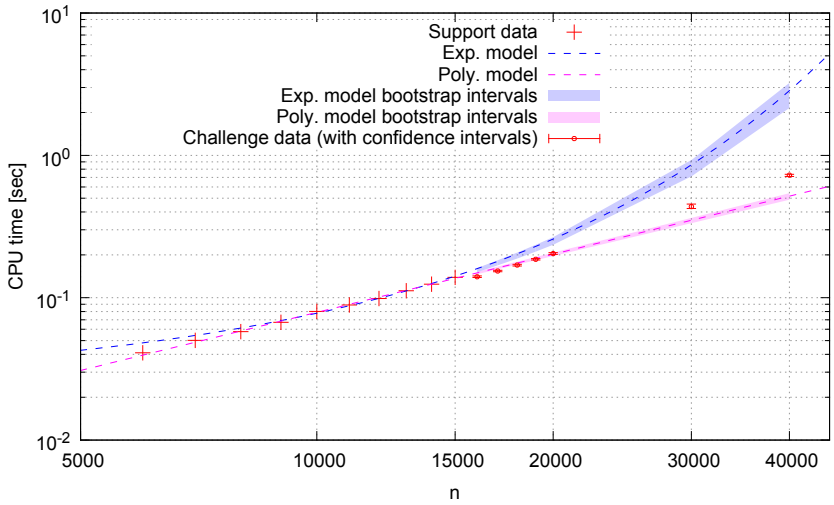


Figure 5.11: Fitted models for the median running times of WalkSAT/SKC. Both models are fitted to the median running times of WalkSAT/SKC solving the less-constrained 4-SAT instances from the set of random instances with $6\,000, 7\,000, \cdots, 15\,000$ variables, and are challenged by median running times with $\geq 16\,000$ variables.

According to Alfredo Braunstein and Riccardo Zecchina[5], the reason for this is that the solver, after some decimation, starts not converging and eventually goes to a singularity of the underlying fixed point equations. Then, the solver presumably makes random choices and eventually gets into a bad partial configuration of variables that leads to a contradiction. To solve this problem, one needs to implement a solver combining survey propagation with backtracking. Due to time constraints, we did not pursue this path to evaluate the survey propagation algorithm empirically. However, our methodology is readily applicable to the survey propagation algorithm, once a practical implementation of the algorithm is available, and such analysis should be of interest to the community.

5.6 Chapter Summary

In this chapter, we presented an empirical analysis of the scaling behaviour of several prominent SAT solvers on phase transition uniform random 3-SAT instances. We were surprised to find solid support for low-degree polynomial scaling of the performance of all three SLS-based SAT solvers we studied, which stands in stark

[5]Personal communications.

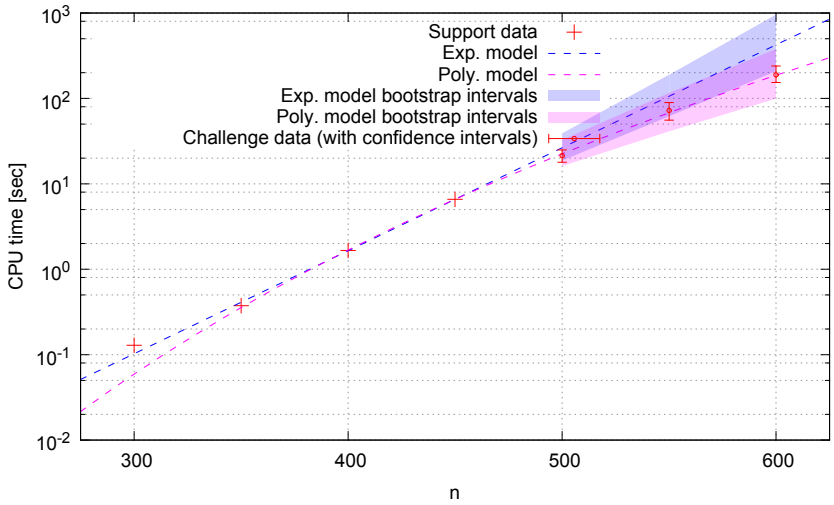


Figure 5.12: Fitted models for the median running times of kcnfs. Both models are fitted to the median running times of kcnfs solving the less-constrained 4-SAT instances from the set of random instances with $300, \cdots, 450$ variables, and are challenged by median running times for instances with $500, 550, 600$ variables.

contrast to the exponential performance scaling of the three DPLL-based solvers we considered, even when these were run on satisfiable instances only. As expected, we did find evidence for significant differences in the performance scaling models of DPLL-based solvers on satisfiable and unsatisfiable instances, but these differences could be attributed to a constant factor, suggesting that they cannot be leveraged to obtain reasonably accurate guesses on the unsatisfiability of instances based on long runs of those solvers. However, the qualitative differences between the scaling of SLS- and DPLL-based solvers can be exploited, in that for growing n, it appears to be possible to separate satisfiable from unsatisfiable instances with high and further increasing accuracy based on long runs of SLS-based or hybrid solvers. Furthermore, the polynomial scaling of even high quantiles of the distribution of running times for SLS-based solvers across instance sets suggests that random 3-SAT instances from the solubility phase transition are likely not capturing the difficulty we expect to encounter in the worst case when solving an $\mathcal{NP}$-hard problem. We note that, considering their sizes, these instances are still very hard compared to almost all types of structured SAT instances and may therefore still be useful as a benchmark.

We also explored the empirical scaling of these solvers on two distributions of random 4-SAT instances. For phase-transition instances, exponential (for BalancedZ) or root-exponential (for the other solvers) models usually fit the running

n	200	250	300	350	400	450
# Instances	581	589	633	558	579	572
# Not solved	211	268	317	285	319	329
% Not solved	36.3%	45.5%	50.1%	51.1%	55.1%	57.5%

n	500	600	700	800	900	1000
# Instances	578	572	607	584	592	593
# Not solved	355	356	398	408	393	437
% Not solved	61.4%	62.2%	65.6%	69.9%	66.4%	73.7%

Table 5.23: Number of the satisfiable instances of different sizes, and the number and percentage of instances that cannot be solved in at least 3 out of 5 independent runs of the survey propagation algorithm.

times of SLS-based solvers better, while DPLL-based solvers demonstrate scaling behaviour well characterised by exponential models. For solving a class of less-constrained instances, we showed that both WalkSAT/SKC and kcnfs scale significantly better than solving phase-transition instances in that a polynomial model is the better fit for both solvers.

To the best of our knowledge, ours is the first study that uses a statistically sound way to assess the scaling of SAT solver performance with instance size, and to discriminate between different scaling models. The empirical scaling analysis we have performed here can easily be applied to other SAT solvers and other distributions of SAT instances (as long as reasonably large sets of instances for each n can be obtained). We believe that doing so can and will produce interesting and useful results that can inspire the design of algorithms and benchmark instance generators as well as, hopefully, theoretical work.

Chapter 6

Empirical Scaling of Running Time of TSP Solvers [6]

In this chapter, we studied the empirical scaling of finding time, that is, the time required by a solver to find an optimal solution of a TSP instance without proving optimality, of prominent TSP solvers. Our analyses were done on the state-of-the-art complete solver, Concorde, and two prominent incomplete solvers, EAX and LKH.

6.1 Experimental Setup

For our analyses, we used the benchmark RUE instances that were previously used and made available to us by Dubois-Lacoste et al. [21] as well as by Hoos and Stützle [35, 36]. We focused on RUE instances, because they represent a widely-studied distribution of hard TSP instances and can be obtained easily for a given set of instance sizes. These instances were generated using the portgen generator from the 8th DIMACS implementation challenge for TSP [38]. It places n points in a $100\,000 \times 100\,000$ square uniformly at random and computes the Euclidean distances between pairs of points. There are 1000 instances for each instance size $n = 500, 600, \cdots, 1500, 2000$ and 100 instances for each $n = 2500, 3000, \cdots, 4500$.

Regarding solvers, we chose three high-performance TSP solvers that are well known in the community, including one complete algorithm and two incomplete algorithms. All algorithms perform well on large TSP instances and play an important role in the development of TSP solving techniques. We thus have reasons to expect that analyses of these solvers would greatly enrich our understanding of empirical time complexity for finding optimal TSP solutions.

For complete algorithm, we concentrated on the long-time state-of-the-art exact TSP solver, Concorde [4]. Mainly based on a branch & cut scheme, it makes use of a multitude of heuristics and is the best-performing exact solver that we are aware of. The solver has been used to solve some largest non-trivial instances.

[6] This chapter covers results that are also reported in two working papers [54, 55]

For incomplete algorithms, we chose the latest versions of LKH and EAX. LKH [29, 30], Helsgaun's variant of the Lin-Kernighan TSP heuristic, is a variable-depth search method that performs sophisticated heuristic-guided local search moves based on sequences of five or more edge exchanges. It can perform restarts based on perturbations of previously found solutions using a variety of strategies. The algorithm arguably represents a milestone in the development of inexact TSP solving and the state-of-the-art for finding high-quality TSP solutions. In this work, we used LKH version 2.0.7, keeping parameters at default except for PATCHING_A and PATCHING_C, which we set to 2 and 3 respectively to include patching of cycles in searching for improving moves. These values were also adopted in the example parameter file for solving TSPLIB instance pr2392 and in earlier work by Dubois-Lacoste et al. [21].

Recently, there has been a fast-developing line of work on evolutionary algorithms for inexact solving of TSP. EAX [57], a very successful genetic algorithm, makes use of improved variants of the edge assembly cross-over recombination operator. It also exploits diversity preservation techniques and carefully initialises the initial population by local optimisation. In the following, we used EAX with default parameters, namely with population size as 100 and number of no-improvement iterations before restart as 30.

For our analyses, we used implementations of LKH and EAX made available to us by Dubois-Lacoste et al. [21]. Their versions of the solvers use a restart mechanism to achieve improved performance. Restart mechanisms are needed because even high-performance local search algorithms can suffer from stagnation behaviour, and restart mechanisms help both algorithms considerably to find the known optimal solution more efficiently.

For our scaling analysis, we considered three parametric models:

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

Similar to our analyses on SAT, we fitted these models on the medians of running times, and computed 95% confidence intervals of the predictions following methodology outlined in Chapter 3.

For running these TSP solvers, we used a cluster of computers, each equipped with a 2.0GHz eight-core AMD 6128 processor, 2×12MB L2/L3 cache and 16GB RAM. The operating system was Cluster Rocks Linux 6.0/CentOS 6.3, and gcc 4.4.6 was used to compile the programs with optimisation flag -O3. To avoid measurement noise caused by memory bottlenecks, we used only one core per CPU to

run our solvers and imposed a memory limit of 1GB per run. We performed 10 independent runs of incomplete TSP solvers, and used the median as well.

6.2 Empirical Scaling of Running Time of Concorde for Finding Optimal Solutions

In this section, we study empirical scaling of finding time, namely the time required to find an optimal solution of a given TSP instance, of Concorde, the state-of-the-art complete solver for TSP. We further compared the scaling results with those of overall running time of Concorde and obtained interesting results. Our work complement previous analyses on empirical scaling of TSP, as discussed in Section 2.2.

6.2.1 Treatments of running time data

When studying the running times for finding optimal solutions (finding times) compared to those including proving optimality (overall running times), care needs to be taken in the treatment of timed-out runs. In the Concorde case, this concerns runs on instances for which only pseudo-optimal solutions or not even pseudo-optimal solutions are available. These solutions were found through multiple runs of EAX and LKH. For a subset of these instances, EAX and LKH reached the same best solution in every single run; we conjecture that these best solutions are in fact optimal and refer to them as pseudo-optimal. When comparing the best solution found during a run of Concorde to the pseudo-optimal solution or the best solution we know for an instance in case these are not considered pseudo-optimal, we could verify that for all except one such solution Concorde did not find within the 7 CPU days such solutions, that is, we can verify that the finding times of Concorde are larger than 7 CPU days and, hence, larger than the median times on which we base the analysis given below. It thus enables us to compute the median finding time in a definite manner, despite that finding times for these instances are unknown. For the remaining instance on which Concorde found the same solution as the best known one, the computation time to match the best solution known was larger than the respective median time required on instances for which proven optimal solutions are already known.

6.2.2 Scaling models, RMSEs and bootstrap intervals

We first fitted parametric models on the finding times of Concorde, and obtained three models together with corresponding RMSE values as in Table 6.1. According

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

Table 6.1: Fitted models for the medians of the running times of Concorde and the corresponding RMSE values (in CPU sec). The models yielding more accurate predictions (as per RMSEs on challenge data) are shown in boldface.

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

Table 6.2: 95% bootstrap confidence intervals for the parameters of the scaling models for Concorde's running times to find optimal solutions of RUE instances.

to the RMSE values on support and challenge data, the root exponential model gives the best fit.

To assess the confidence we should have in these models, we used the bootstrap re-sampling method described in Section 3.1 to evaluate the models, which gives us the confidence intervals for model parameters (Table 6.2) and for observed/predicted running times (Tables 6.3 and 6.4 respectively). The results, also clearly seen from Figure 6.1, indicate that the root-exponential model with a base ≈ 1.25 is a very good fit for the finding times of Concorde.

6.2.3 Comparison with scaling for overall running time of Concorde

Previously, Hoos and Stützle [35] have shown that the overall running times (for finding optimal solutions and proving optimality) of Concorde also scale as a root-exponential model. Here, we repeated the analysis with newer overall running time data that we collected together with finding times. The results we thus obtained confirmed a root-exponential scaling model for the overall running time of Concorde, with a in the interval $[0.11658, 0.35323]$ and b in the interval $[1.2126, 1.2522]$. Note that the confidence interval for b is very close to that obtained for finding time, which is $[1.2287, 1.2793]$. Moreover, the predictions made by the root-exponential model of overall running time, as shown in Table 6.5, are quite consistent with observed finding times. A closer look at the data, however, reveals that the observed finding times are usually closer to the lower end of the corresponding bootstrap confidence intervals, while the observed overall running times are usually closer

Solver	n	Observed median finding time	
		Point estimates	Confidence intervals
Concorde	2000	1969	$[1739, 2222]$
	2500	6149	$[4084, 8812]$
	3000	1.84×10^4	$[1.332 \times 10^4, 2.669 \times 10^4]$
	3500	3.246×10^4	$[2.581 \times 10^4, 5.038 \times 10^4]$
	4000	1.312×10^5	$[7.073 \times 10^4, 2.024 \times 10^5]$
	4500	2.633×10^5	$[1.73 \times 10^5, 4.419 \times 10^5]$

Table 6.3: 95% bootstrap confidence intervals for the medians of observed running times for Concorde to find optimal solutions of RUE instances. The instance sizes shown here are larger than those used for fitting the models.

Solver	n	Predicted confidence intervals		
		Exp. model	RootExp. model	Poly. model
Concorde	2000	$[\mathbf{1988}, \mathbf{3179}]$	$[\mathbf{1528}, \mathbf{2269}]$*	$[\mathbf{1228}, \mathbf{1795}]$
	2500	$[\mathbf{8718}, \mathbf{1.884 \times 10^4}]$	$[\mathbf{4536}, \mathbf{8335}]$#	$[\mathbf{2737}, \mathbf{4771}]$
	3000	$[3.853 \times 10^4, 1.103 \times 10^5]$	$[\mathbf{1.212 \times 10^4}, \mathbf{2.694 \times 10^4}]$*	$[5252, 1.057 \times 10^4]$
	3500	$[1.698 \times 10^5, 6.479 \times 10^5]$	$[\mathbf{3.001 \times 10^4}, \mathbf{7.925 \times 10^4}]$#	$[9149, 2.069 \times 10^4]$
	4000	$[7.5 \times 10^5, 3.809 \times 10^6]$	$[\mathbf{6.95 \times 10^4}, \mathbf{2.163 \times 10^5}]$*	$[1.477 \times 10^4, 3.708 \times 10^4]$
	4500	$[3.301 \times 10^6, 2.245 \times 10^7]$	$[\mathbf{1.528 \times 10^5}, \mathbf{5.563 \times 10^5}]$*	$[2.248 \times 10^4, 6.205 \times 10^4]$

Table 6.4: 95% bootstrap confidence intervals for the medians of the running time predictions for Concorde to find optimal solutions of RUE instances. The instance sizes shown here are larger than those used for fitting the models. Bootstrap intervals on predictions that are weakly consistent with the observed data are shown in boldface, those that are strongly consistent are marked by sharps (#), and those that fully contain the confidence intervals on observations are marked by asterisks (*).

Solver	n	Predicted confidence intervals	Observed median overall running time	
		RootExp. model	Point estimates	Confidence intervals
Concorde	2000	$[\mathbf{1962}, \mathbf{2736}]$#	2508	$[2197, 2760]$
	2500	$[\mathbf{5431}, \mathbf{8914}]$#	7899	$[4886, 9789]$
	3000	$[\mathbf{1.366 \times 10^4}, \mathbf{2.612 \times 10^4}]$#	2.064×10^4	$[1.492 \times 10^4, 2.795 \times 10^4]$
	3500	$[\mathbf{3.181 \times 10^4}, \mathbf{6.994 \times 10^4}]$#	4.057×10^4	$[2.586 \times 10^4, 5.719 \times 10^4]$
	4000	$[\mathbf{6.987 \times 10^4}, \mathbf{1.753 \times 10^5}]$#	1.377×10^5	$[8.236 \times 10^4, 2.108 \times 10^5]$
	4500	$[\mathbf{1.462 \times 10^5}, \mathbf{4.154 \times 10^5}]$#	3.264×10^5	$[1.953 \times 10^5, 5.087 \times 10^5]$

Table 6.5: 95% bootstrap confidence intervals for the medians of the observed and predicted overall running times for Concorde to solve RUE instances. Bootstrap intervals on predictions that are weakly consistent with the observed finding times are shown in boldface, those that are strongly consistent are marked by sharps (#), and those that fully contain the confidence intervals on observations are marked by asterisks (*).

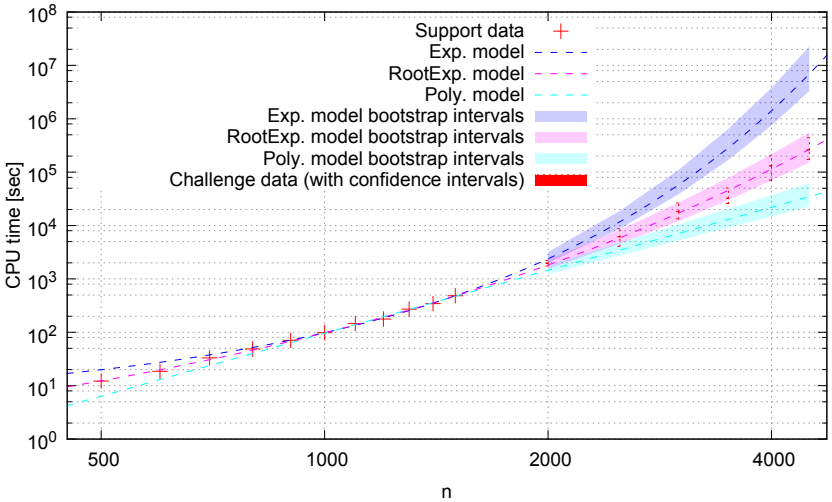


Figure 6.1: Fitted models for the medians of the running times for Concorde to find optimal solutions of RUE instances without proving optimality. All models are fitted with the medians of the running times of Concorde solving the RUE instances from the set of instances $500 \leq n \leq 1500$ variables, and are challenged by the medians of the running times of $2000 \leq n \leq 4500$ variables.

to the higher end. In other words, we do not have strong evidence that the finding times scale differently compared to the overall running times, but more experiments may narrow down the confidence intervals and help us better understand the scaling behaviour of Concorde.

To further test whether the difference between finding and overall running times is explained by a constant factor, we fitted a one-parameter root-exponential model of the form $a \cdot b_{proving}^{\sqrt{n}}$ on the finding times of Concorde, where $b_{proving} = 1.2281$ from the root-exponential model for Concorde overall running time. Our results show that the model obtained in this way tends to under-estimate the finding times, and imply that the finding times scale slightly worse than overall running times. Note that this is also suggested by the difference in the confidence intervals for b in the two models. Moreover, it was also observed by Hoos and Stützle that, as n increases, the percentage of running times spent on finding optimal solutions goes up and dominates the running time for large n [36].

So, do Concorde finding times scale worse than overall running times? While this is not clear from cross-checking model predictions with observed finding times, our analysis using the one-parameter root-exponential model clearly suggests an affirmative answer. We argue that two reasons cause our analysis with the two-parameter model of overall running time to fail to illustrate this: 1) the two-

parameter model represents a large family of models, and the bootstrap confidence intervals tend to be wide; 2) the fitted root-exponential model for overall running time has a larger a, which causes an illusion that overall running times scale worse than finding times. Given the relationship between finding and overall running times, though, we believe that more sophisticated models than the two-parameter root-exponential model we use here would be required in order to explain the increase of fraction of running time spent on finding optimal solutions.

6.3 Empirical Scaling of Running Time of EAX and LKH

In this section, we study empirical scaling of running time of EAX and LKH, two prominent incomplete solvers for TSP. We further compare the scaling results with those of findings time of Concorde and obtain interesting results. Our work complements previous analyses on empirical scaling of TSP, as discussed in Section 2.2.

6.3.1 Details of instances

As described in detail before, Concorde does not manage to solve all instances within the allowed time. Because EAX and LKH cannot prove optimality, they need to be given the optimal solution in advance in order to terminate once the optimal solution is reached. To make more instances available for EAX and LKH to solve, we ran Concorde on the previously unsolved instances with different seeds and/or on (limited number of) faster machines. In addition, we performed multiple runs of EAX and LKH on those instances that remain unsolved by Concorde. For a subset of these instances, EAX and LKH reached the same best solution in every single run; we conjecture that these best solutions are in fact optimal and refer to them as pseudo-optimal. In the results to follow, we include data for both optimal and pseudo-optimal instances. We note that only using data for optimal instances gives qualitatively similar conclusions.

For the six instances (two of size 4000 and four of size 4500) for which we did not succeed in establishing pseudo-optimal solutions, we took special care in how to treat them. As found by Dubois-Lacoste et al. [21], performance correlations between each pair of solvers are all very low. Thus, the instances for which we do not have optimal or pseudo-optimal solutions may actually be easy for the incomplete solvers. Thus, they are treated using an optimistic/pessimistic estimation as done by Dubois-Lacoste et al. [21]. More precisely, we treat these instances as easy with smaller-than-the-median running times in the optimistic estimation, and

Solver	Model		RMSE (support)	RMSE (challenge)
EAX	Exp.	1.6512×1.0017^{n}	0.80329	$[1513.3, 1566.2]$
	RootExp.	$\mathbf{0.24795 \times 1.1230^{\sqrt{n}}}$	**0.45614**	$[\mathbf{44.77}, \mathbf{88.739}]$
	Poly.	$1.9196 \times 10^{-5} \times n^{1.9055}$	0.10699	$[235.79, 287.6]$
LKH	Exp.	$0.56147 \times 1.0025^{x}$	0.51265	$[18213, 18330]$
	RootExp.	$0.030075 \times 1.1879^{\sqrt{x}}$	0.38383	$[797.7, 909.51]$
	Poly.	$\mathbf{1.15 \times 10^{-8} \times x^{2.9297}}$	**0.50193**	$[\mathbf{282.12}, \mathbf{378.53}]$

Table 6.6: Fitted models for the medians of the running times of EAX and LKH and the corresponding RMSE values (in CPU sec). The models yielding more accurate predictions (as per RMSEs on challenge data) are shown in boldface.

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

Table 6.7: 95% bootstrap confidence intervals for the model parameters of the scaling models for EAX's and LKH's running times to find optimal solutions of RUE instances.

as timed-out instances in the pessimistic treatment. This gives us intervals for the medians for those sizes where there are instances with unknown (pseudo-)optimal solutions. We note that these intervals are not confidence intervals, but bounds on the median running times, as the true median must be contained within these intervals.

6.3.2 Scaling models, RMSEs and bootstrap intervals

Similarly, we first fitted parametric models to the running times of EAX and LKH, resulting in the models and corresponding RMSE values shown in Table 6.6. Judged from the RMSE values, the running times of EAX and LKH are best approximated by a root-exponential and a polynomial model, respectively.

We also assessed the fitted models with bootstrap re-sampling, using the method described in Section 3.1. In particular, we calculated the confidence intervals for model parameters (Table 6.7) and for observed/predicted running times (Tables 6.8 and 6.9). The results for EAX, also presented graphically in Figure 6.2, indicate that the root-exponential model with a base ≈ 1.123 is a fairly good fit. On the other hand, the results for LKH, also shown in Figure 6.3, suggest a scaling behaviour bounded from below and above by a polynomial and a root-exponential

Solver	n	Observed median run-time	
		Point estimates	Confidence intervals
EAX	2000	41.24	[40.03, 42.26]
	2500	73.19	[61.11, 118]
	3000	172.2	[155.7, 223.2]
	3500	239.9	[220, 357.7]
	4000	[483.2, 547.4]	[370.2, 649.8]
	4500	[611.7, 727.7]	[520, 877.6]
LKH	2000	62.64	[58.03, 69.61]
	2500	137	[108.5, 199.7]
	3000	249.4	[201.3, 372.2]
	3500	382.8	[260.8, 648.8]
	4000	[891, 907.3]	[551.5, 1207]
	4500	[1059, 1352]	[808.2, 2203]

Table 6.8: 95% bootstrap confidence intervals for the medians of the observed running times for EAX and LKH to find optimal solutions of RUE instances. The instance sizes shown here are larger than those used for fitting the models.

Solver	n	Predicted confidence intervals		
		Exp. model	RootExp. model	Poly. model
EAX	2000	[53.08, 54.75]	[43.77, 45.01]	[37.02, 37.98]
	2500	[125.9, 131.9]	[**80.33**, **83.51**]	[56.43, 58.35]
	3000	[298.7, 317.8]	[139, 146]	[79.63, 82.87]
	3500	[709.2, 765.6]	[**230.3**, **244**]#	[106.5, 111.5]
	4000	[1682, 1845]	[**368.4**, **393.6**]	[137.1, 144.1]
	4500	[3991, 4445]	[**572.8**, **616.7**]+	[171.3, 180.8]
LKH	2000	[**62.15**, **95.52**]#	[**58.8**, **73.94**]#	[**48.05**, **59.71**]
	2500	[**174.5**, **360.2**]	[**138.1**, **193.6**]	[**88.54**, **119.8**]
	3000	[490, 1361]	[**299.3**, **462.5**]	[**145.9**, **211.4**]
	3500	[1376, 5154]	[**609.3**, **1030**]	[**222.4**, **342.4**]
	4000	$[3863, 1.954 \times 10^4]$	[**1176**, **2178**]	[320.7, 519.9]
	4500	$[1.085 \times 10^4, 7.412 \times 10^4]$	[**2173**, **4391**]	[442.8, 751.6]

Table 6.9: 95% bootstrap confidence intervals for the medians of the running time predictions for EAX and LKH to find optimal solutions of RUE instances. The instance sizes shown here are larger than those used for fitting the models. Bootstrap intervals on predictions that are weakly consistent with the observed data are shown in boldface, those that are consistent are marked by plus signs (+), those that are strongly consistent are marked by sharps (#), and those that fully contain the confidence intervals on observations are marked by asterisks (*).

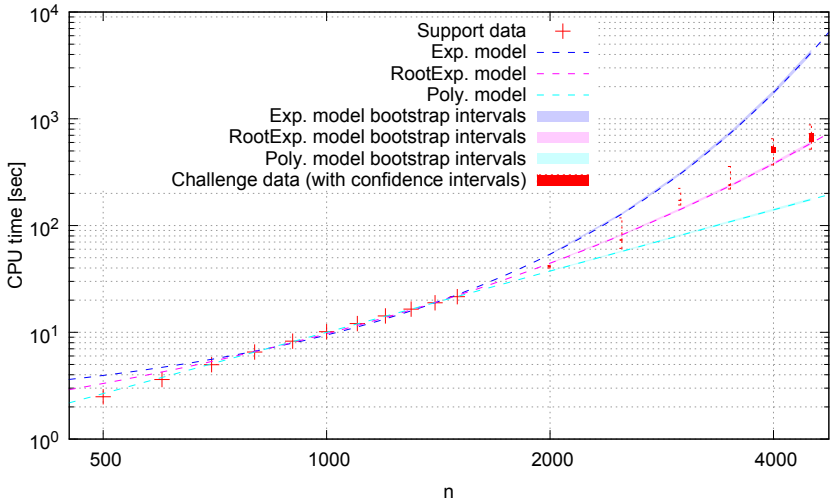


Figure 6.2: Fitted models for the medians of the running times for EAX to find optimal solutions of RUE instances. All models are fitted with the medians of the running times of EAX solving the set of RUE instances of $500 \leq n \leq 1500$ variables, and are challenged by the medians of the running times of $2000 \leq n \leq 4500$ variables.

model, respectively.

6.3.3 Comparison with scaling of Concorde for finding optimal solutions

To compare the scaling of Concorde with that of EAX and LKH, we first analysed the bootstrap intervals of the values of the parameter b in the obtained root-exponential models. For Concorde, $b \in [1.2255, 1.2747]$, which is significant larger than that for EAX ($[1.1219, 1.1242]$) and for LKH ($[1.1749, 1.2006]$). This observation suggests that Concorde scales significantly worse than EAX and LKH.

To further test this, we fitted two one-parameter models of the form $a \cdot b_{Concorde}^{\sqrt{n}}$ to the running times of EAX and LKH, where $b_{Concorde} = 1.2503$. The obtained models, shown in Figure 6.4, clearly over-estimate the running times of EAX and LKH, which confirms our observation that the finding times of Concorde scale significantly worse than those of EAX and LKH.

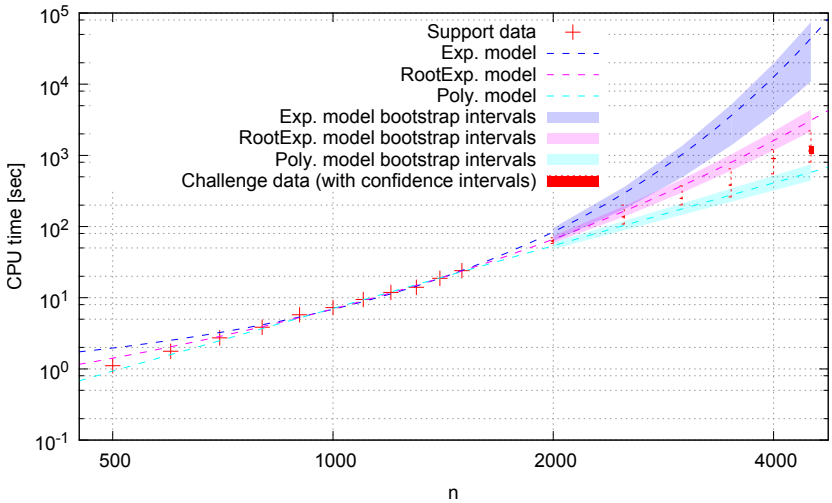


Figure 6.3: Fitted models for the medians of the running times for LKH to find optimal solutions of RUE instances. All models are fitted with the medians of the running times of LKH solving the set of RUE instances of $500 \leq n \leq 1500$ variables, and are challenged by the medians of the running times of $2000 \leq n \leq 4500$ variables.

6.4 Impact of Automated Configuration on Scaling of EAX and LKH

We also performed several experiments investigating the impact of automated configuration on scaling behaviour of incomplete TSP solvers. Automated configuration involves the use of an automated procedure to determine good parameter values for a given algorithm. In this work, we used SMAC [37], the prominent sequential model-based algorithm configurator, to configure EAX and LKH, and assessed the impact of algorithm configuration on their scaling behaviour. Following the standard configuration protocol, we performed 25 independent runs of SMAC for each scenario, selected the best parameter setting according to performance on the given set of training instances and report scaling results for that setting.

6.4.1 Impact of Automated Configuration on Scaling of EAX

We first present results for EAX, which has only two exposed parameters. One key parameter is fNumOfPop, the population size whose default value is 100, and the other is fNumOfKids, the number of no-improvement iterations before the search restarts, for which the default value is 30. In our experiments, we enforced a cur-

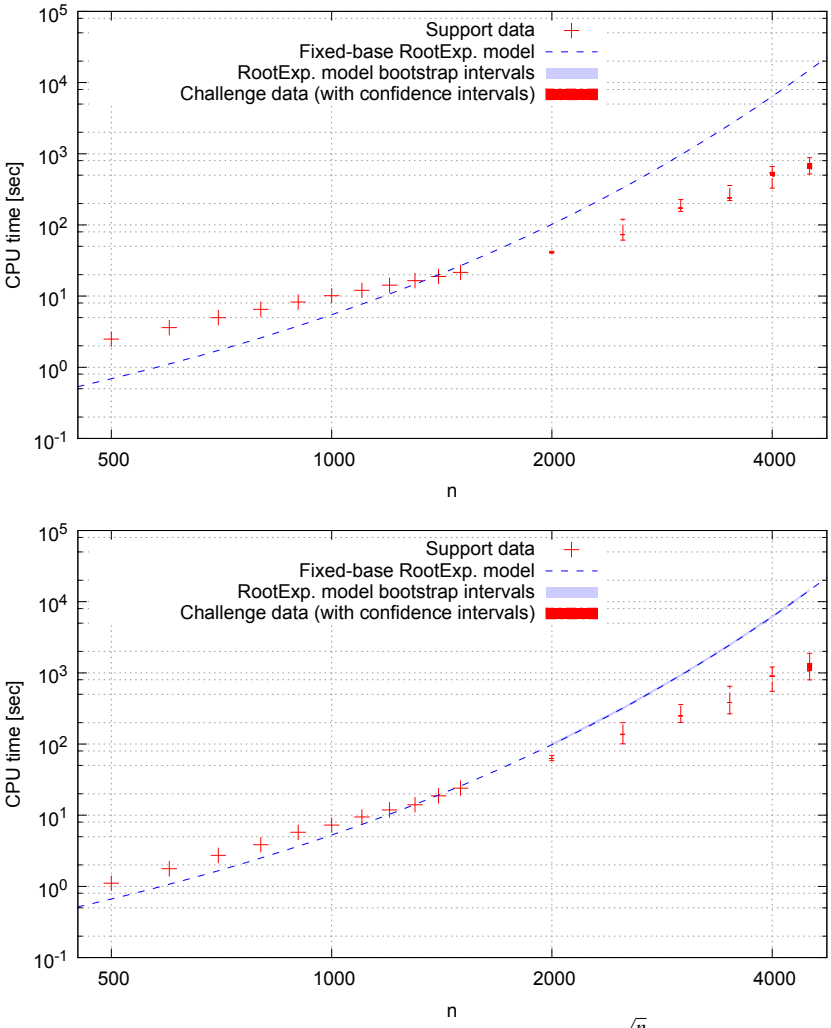


Figure 6.4: Fitted one-parameter models (of the form $a \cdot b_{Concorde}^{\sqrt{n}}$) for the medians of the running times for EAX (top) and LKH (bottom) to find optimal solutions of RUE instances. All models are fitted with the medians of the running times of the solvers solving the RUE instances from the set of instances of $500 \leq n \leq 1500$ variables, and are challenged by the medians of the running times of $2000 \leq n \leq 4500$ variables.

Solver	Model		RMSE (support)	RMSE (challenge)
EAX	Exp.	1.2511×1.0017^{n}	0.4711	$[1085, 1097.8]$
	RootExp.	$\mathbf{0.1991 \times 1.1193^{\sqrt{n}}}$	**0.22625**	$[\mathbf{22.278}, \mathbf{32.899}]$
	Poly.	$2.0916 \times 10^{-5} \times n^{1.8464}$	0.050459	$[124.04, 135.61]$

Table 6.10: Fitted models for the medians of the running times of EAX with configured parameters and the corresponding RMSE values (in CPU sec). The models yielding more accurate predictions (as per RMSEs on challenge data) are shown in boldface.

Solver	Model	Confidence interval of a	Confidence interval of b
EAX	Exp.	$[1.2249, 1.2709]$	$[1.0017, 1.0017]$
	RootExp.	$[0.19024, 0.20586]$	$[1.1182, 1.1208]$
	Poly.	$[1.7804 \times 10^{-5}, 2.3636 \times 10^{-5}]$	$[1.8288, 1.8693]$

Table 6.11: 95% bootstrap confidence intervals for the parameters of the scaling models for EAX's running times (with configured parameters) to find optimal solutions of RUE instances.

off of 1 CPU day for each SMAC run. To ensure that SMAC could perform at least 1000 runs of EAX, we further enforced a cut-off time of 86 sec for each EAX run. After training on a set of RUE instances with 1500 cities, SMAC picked a parameter setting with larger population sizes (167 vs. 100) and smaller number of no-improvement iterations (20 vs. 30).

To investigate the scaling of EAX with optimised parameters, we used the methodology described in Section 3.1. The fitted models are presented in Table 6.10 and illustrated in Figure 6.5 together with the confidence intervals obtained from bootstrap analysis. Similar to results for the default version of EAX, root-exponential characterises the scaling the best, while the best exponential and polynomial models can be rejected with 95% confidence. From the confidence intervals of the model parameters, as shown in Table 6.11, there is evidence that algorithm configuration improves the scaling of EAX, since it reduces the value of b in root-exponential models from 1.123 to 1.119 with completely disjoint confidence intervals ($[1.1219, 1.1242]$ vs. $[1.1182, 1.1208]$). Following the method described in Section 3.2, we cross-checked the predictions of the root-exponential model for the default version of EAX against the observed running times for the optimised version of EAX. Our results, illustrated in Figure 6.6, clearly show that the model over-estimate the running times for the optimised version of EAX.

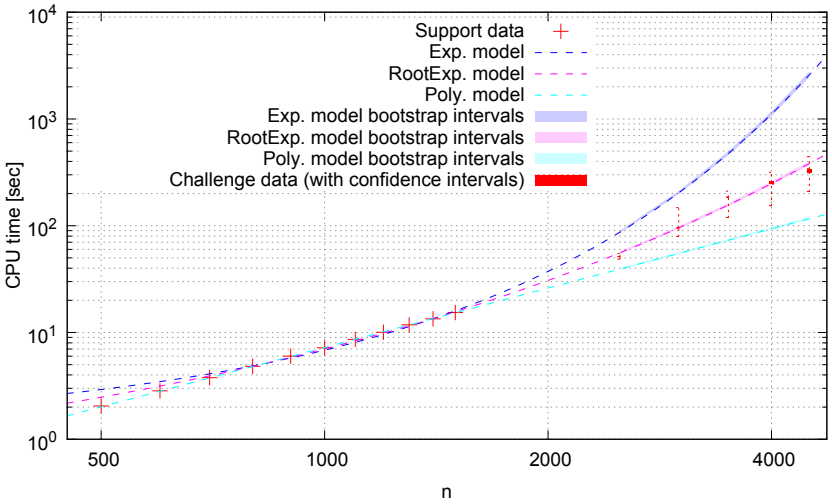


Figure 6.5: Fitted models for the medians of the running times for EAX with configured parameters to find optimal solutions of RUE instances. The parameters of EAX are set to the optimised values determined by configuration on instances with 1500 variables using SMAC. All models are fitted with the medians of the running times of EAX solving the SAT instances from the set of RUE instances of $500 \leq n \leq 1500$ variables, and are challenged by the medians of the running times of $2000 \leq n \leq 4500$ variables.

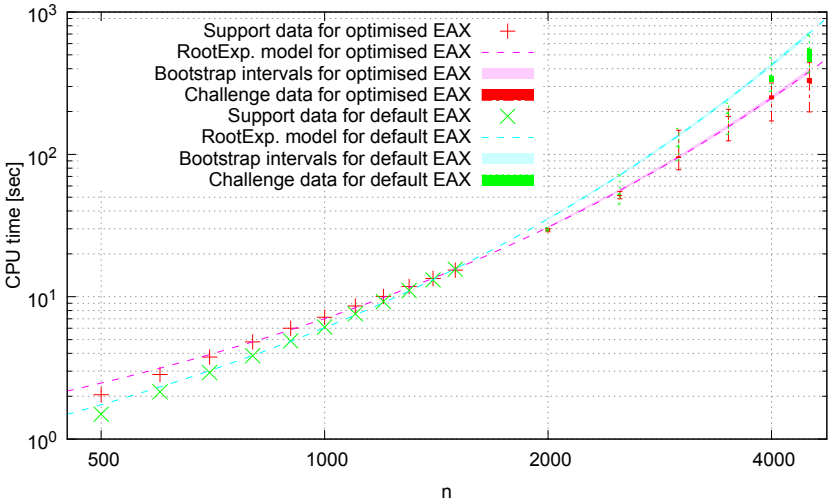


Figure 6.6: Best fitted models for the medians of the running times for EAX with configured and default parameters to find optimal solutions of RUE instances. All models are fitted with the medians of the running times of EAX solving the SAT instances from the set of RUE instances of $500 \leq n \leq 1500$ variables, and are challenged by the medians of the running times of $2000 \leq n \leq 4500$ variables.

6.4.2 Effect of Population Size for EAX

Considering the impact of the parameters, it is natural to increase the population size when solving a larger instance size. Thus, we also performed experiments to quantify the impact of varying population size with instance size. In particular, we set the population size as a multiple of the instant size, namely,

$$ps(n) = \alpha \cdot n,$$

where $ps(n)$ is the population size for instance size n.

We experimented with two different ways of determining α. The first way is to choose α based on the default parameter settings, namely, fNumOfPop $= 100$ (for training instances with instance size $n = 1500$) and fNumOfKids $= 30$. This gives $\alpha = 0.067$. We then followed the methodology described in Section 3.1 to examine the scaling of EAX with these optimised parameters. The fitted models are presented in Table 6.12 and illustrated in Figure 6.7. Surprisingly, the best-fitting model is now the polynomial model, while the best and root-exponential models are rejected with 95% confidence. This stands, as illustrated in Figure 6.8, in contrast with the scaling of the default version of EAX, which is well characterised by a root-exponential model. We also performed bootstrap analysis described

Solver	Model		RMSE (support)	RMSE (challenge)
EAX	Exp.	$0.34117 \times 1.0026^{n}$	0.37901	$[13976, 13987]$
	RootExp.	$0.017513 \times 1.1914^{\sqrt{n}}$	0.17295	$[840.11, 850.63]$
	Poly.	$\mathbf{4.7691 \times 10^{-9} \times n^{2.9918}}$	**0.099509**	$[\mathbf{11.214}, \mathbf{21.427}]$

Table 6.12: Fitted models for the medians of the running times of EAX with default parameters and varying population size and the corresponding RMSE values (in CPU sec). The models yielding more accurate predictions (as per RMSEs on challenge data) are shown in boldface.

Solver	Model	Confidence interval of a	Confidence interval of b
EAX	Exp.	$[0.32156, 0.36342]$	$[1.0025, 1.0026]$
	RootExp.	$[0.01539, 0.01993]$	$[1.1870, 1.1957]$
	Poly.	$[2.9101 \times 10^{-9}, 7.6663 \times 10^{-9}]$	$[2.9241, 3.0607]$

Table 6.13: 95% bootstrap confidence intervals for the parameters of the scaling models for EAX's running times (with default parameters and varying population size) to find optimal solutions of RUE instances.

in Section 3.1, which results in the confidence intervals for the model parameters shown in Table 6.13.

The second way is to configure α (hence fNumOfPop) and fNumOfKids using SMAC. Following the same configuring protocol in the previous configuration experiment with EAX, we arrived at the optimised setting with $\alpha = 0.111$ and fNumOfKids $= 20$. We then followed the same methodology to examine the scaling of EAX with these optimised parameters. The fitted models are presented in Table 6.14 and illustrated in Figure 6.9. Similar to the previous experiment, the best-fitting model is now the polynomial model, while the best and root-exponential models are rejected with 95% confidence. This also stands, as illustrated in Figure 6.10, in contrast with the scaling of the default version of EAX. We also performed bootstrap analysis described in Section 3.1, which results in the confidence intervals for the model parameters shown in Table 6.15.

To summarise, both experiments indicate that varying population size can significantly improve the scaling behaviour of EAX. Whether the formula for the population size is determined based on default parameter settings or based on automated configuration of α (together with fNumOfKids), the scaling of EAX is best captured by a polynomial model. Cross-checking the two polynomial models obtained above using the method described in Section 3.2, we found that the difference between them is not significant.

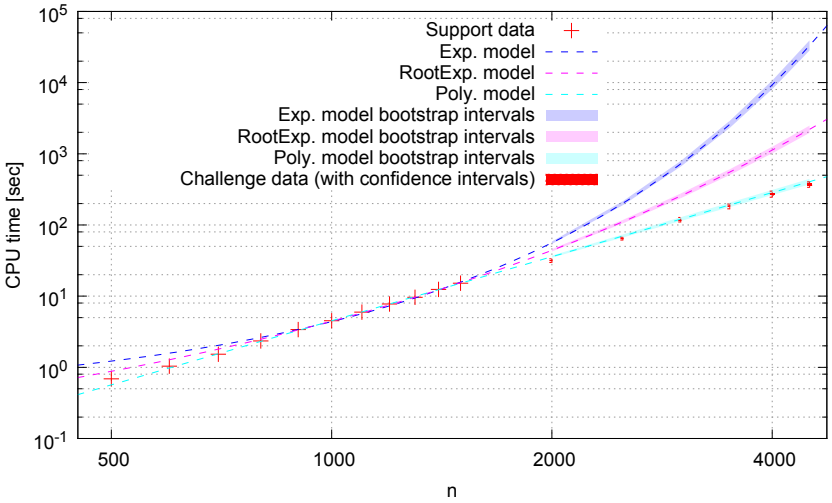


Figure 6.7: Fitted models for the medians of the running times of EAX to find optimal solutions of RUE instances. The parameters of EAX are set to the default values, and the population size is a simple function of instance size. All models are fitted with the medians of the running times of EAX solving the SAT instances from the set of RUE instances of $500 \leq n \leq 1500$ variables, and are challenged by the medians of the running times of $2000 \leq n \leq 4500$ variables.

Solver	Model		RMSE (support)	RMSE (challenge)
EAX	Exp.	$0.43879 \times 1.0024^{n}$	0.48413	[10931, 10940]
	RootExp.	$0.02628 \times 1.1816^{\sqrt{n}}$	0.24178	[667.42, 676.81]
	Poly.	$\mathbf{1.6194 \times 10^{-8} \times n^{2.8364}}$	**0.027288**	[**37.049**, **44.847**]

Table 6.14: Fitted models for the medians of the running times of EAX with configured parameters and varying population size and the corresponding RMSE values (in CPU sec). The models yielding more accurate predictions (as per RMSEs on challenge data) are shown in boldface.

Solver	Model	Confidence interval of a	Confidence interval of b
EAX	Exp.	[0.42779, 0.44748]	[1.0024, 1.0025]
	RootExp.	[0.024996, 0.027408]	[1.1801, 1.1833]
	Poly.	$[1.3335 \times 10^{-8}, 1.9003 \times 10^{-8}]$	[2.8132, 2.8636]

Table 6.15: 95% bootstrap confidence intervals for the parameters of the scaling models for EAX's running times (with configured parameters and varying population size) to find optimal solutions of RUE instances.

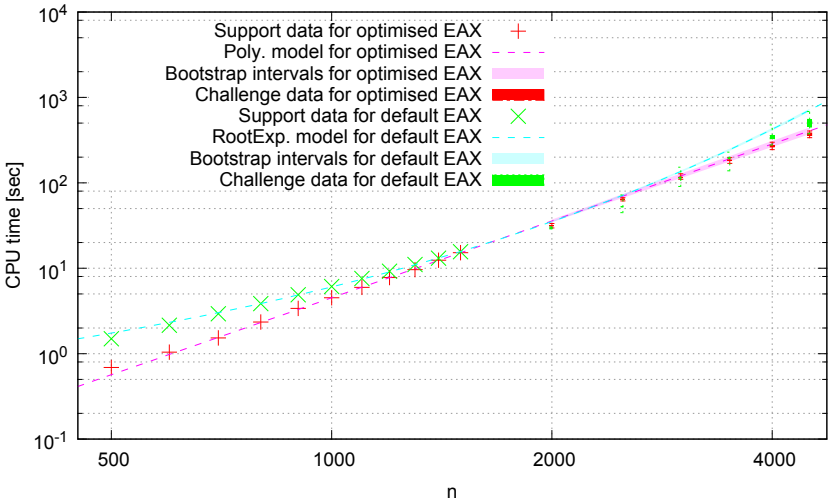


Figure 6.8: Best fitted models for the medians of the running times for EAX with configured and default parameters to find optimal solutions of RUE instances. All models are fitted with the medians of the running times of EAX solving the SAT instances from the set of RUE instances of $500 \leq n \leq 1500$ variables, and are challenged by the medians of the running times of $2000 \leq n \leq 4500$ variables.

6.4.3 Impact of Automated Configuration on Scaling of LKH

We also attempted to configure LKH for scaling performance. With over 40 parameters, LKH is arguably much more configurable than EAX. Out of these parameters, some require additional information like initial tours or sub-division of a given TSP instance, which are not available in our case. Thus, we analysed the details of the parameters, and selected 21 parameters that we could configure without additional information or code modification. These parameters are listed in Table 6.16 in detail. Out of them, 12 are numerical paramters and 9 are categorical. We kept all parameter at default values specified in the user guide, except for PATCHING_A and PATCHING_C. As mentioned earlier, we used 2 and 3 respectively for these two parameters in our earlier experiments, and continued using these values as default in our configuration experiments. The ranges of all parameters were determined based on the user guide. When in doubt, we tried to use large ranges and leave larger configuration space to SMAC.

We first followed the standard protocol, performing 25 SMAC runs to configure LKH using a set of 100 instances of size $n = 1500$ randomly selected from the larger set. We enforced a cur-off of 2 CPU days for each SMAC run. To ensure that SMAC could perform at least 1000 runs of LKH, we further enforced a cut-off

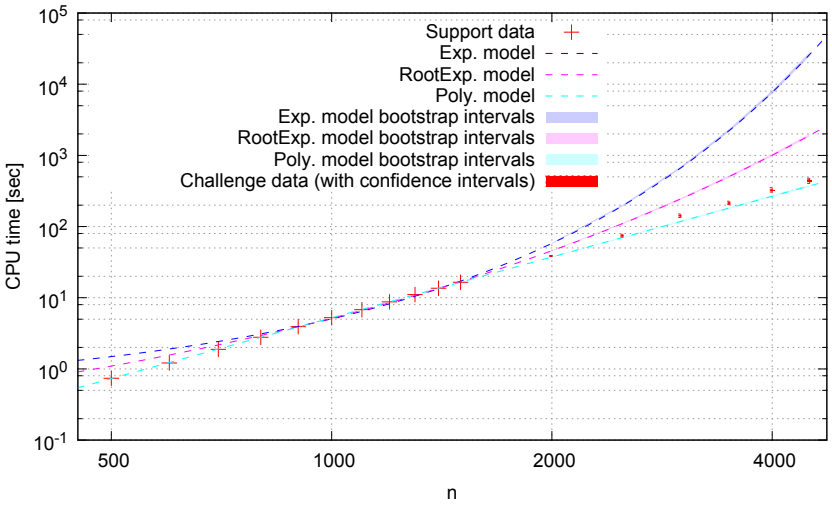


Figure 6.9: Fitted models for the medians of the running times of EAX to find optimal solutions of RUE instances. The parameters of EAX are set to the optimised values, and the population size is a simple function of instance size. All models are fitted with the medians of the running times of EAX solving the SAT instances from the set of RUE instances of $500 \leq n \leq 1500$ variables, and are challenged by the medians of the running times of $2000 \leq n \leq 4500$ variables.

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

Table 6.16: List of numerical (N) and categorical (C) parameters for LKH that are configurable.

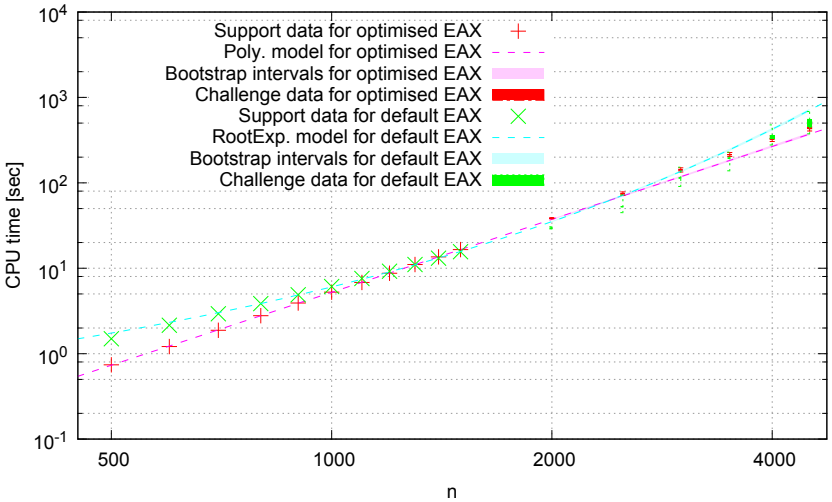


Figure 6.10: Best fitted models for the medians of the running times for EAX with configured and default parameters to find optimal solutions of RUE instances. All models are fitted with the medians of the running times of EAX solving the SAT instances from the set of RUE instances of $500 \leq n \leq 1500$ variables, and are challenged by the medians of the running times of $2000 \leq n \leq 4500$ variables.

time of 172 sec for each LKH run. Out of the 25 parameter settings, we selected the best parameter setting based on training performance on the same set. Analysing the scaling of LKH with this parameter setting, we obtained the three models shown in Table 6.17. Based on RMSE (challenge), the root-exponential model provides the best fit, but has a larger b than that for LKH with default parameters (1.2452 vs 1.1879). We also performed the bootstrap analysis described in Section 3.1. From the bootstrap confidence intervals of the model parameters shown in Table 6.18, the confidence interval of b in the root-exponential model is also larger than that for LKH with default parameters ($[1.2213, 1.2720]$ vs. $[1.1749, 1.2006]$). Moreover, the root-exponential model actually under-estimates the observed running times on challenge data, as clearly seen in Figure 6.11. Examining the running times in detail, we saw that configuration does bring down the running times of LKH for $n \leq 2000$, but causes it to perform worse for larger instances. In other words, the configuration procedure seems to overfit on smaller instance sizes.

We then followed a protocol proposed by Styles et al. [66], performing 25 SMAC runs to configure LKH using a set of 100 instances with instance size $n = 1000$, and selected the best parameter setting based on validation performance on a set of 50 instances with instance size $n = 1500$ randomly selected from the larger set (referred to as scaling protocol hereafter). The underlying idea is to select

Solver	Model		RMSE (support)	RMSE (challenge)
LKH	Exp.	$0.10374 \times 1.0031^{n}$	0.24213	[32342, 44308]
	RootExp.	$\mathbf{0.0022353 \times 1.2452^{\sqrt{n}}}$	**0.17628**	[**5682.2**, **17808**]
	Poly.	$6.8001 \times 10^{-12} \times n^{3.8412}$	0.19239	[7758.7, 19825]

Table 6.17: Fitted models for the medians of the running times of LKH with configured parameters from experiments following the standard protocol and the corresponding RMSE values (in CPU sec). The models yielding more accurate predictions (as per RMSEs on challenge data) are shown in boldface.

Solver	Model	Confidence interval of a	Confidence interval of b
LKH	Exp.	[0.071619, 0.14743]	[1.0028, 1.0034]
	RootExp.	[0.0010571, 0.004436]	[1.2213, 1.272−]
	Poly.	$[3.4774 \times 10^{-13}, 9.5553 \times 10^{-11}]$	[3.4693, 4.2573]

Table 6.18: 95% bootstrap confidence intervals for the parameters of the scaling models for EAX's running times (with configured parameters) to find optimal solutions of RUE instances.

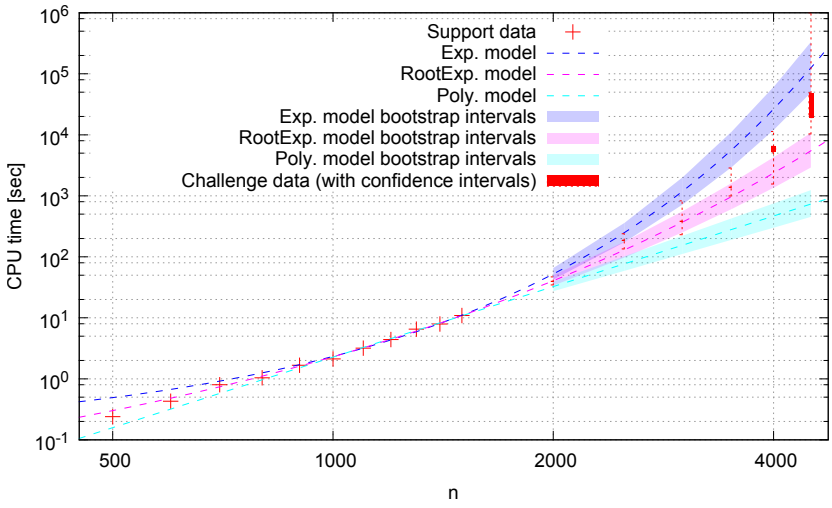


Figure 6.11: Fitted models for the medians of the running times for LKH with configured parameters from experiments following the standard protocol. All models are fitted with the medians of the running times of LKH solving the set of RUE instances of $500 \leq n \leq 1500$ variables, and are challenged by the medians of the running times of $2000 \leq n \leq 4500$ variables.

Solver	Model		RMSE (support)	RMSE (challenge)
LKH	Exp.	$0.17326 \times 1.0027^{n}$	0.25216	$[12219, 12917]$
	RootExp.	$\mathbf{0.0066414 \times 1.2077^{\sqrt{n}}}$	**0.13341**	$[\mathbf{533.13}, \mathbf{1165.9}]$
	Poly.	$4.7378 \times 10^{-10} \times n^{3.2473}$	0.090054	$[1281.8, 1941.8]$

Table 6.19: Fitted models for the medians of the running times of LKH with configured parameters from experiments following the scaling protocol and the corresponding RMSE values (in CPU sec). The models yielding more accurate predictions (as per RMSEs on challenge data) are shown in boldface.

Solver	Model	Confidence interval of a	Confidence interval of b
LKH	Exp.	$[0.13477, 0.22338]$	$[1.0025, 1.0029]$
	RootExp.	$[0.0040961, 0.010936]$	$[1.1908, 1.2245]$
	Poly.	$[7.2956 \times 10^{-11}, 2.9719 \times 10^{-9}]$	$[2.9843, 3.5068]$

Table 6.20: 95% bootstrap confidence intervals for the parameters of the scaling models for EAX's running times (with configured parameters) to find optimal solutions of RUE instances

parameter settings that tend to perform well for larger instances. We again analysed the scaling of LKH with the parameter setting such obtained, and obtained the three models shown in Table 6.19. Based on the RMSE (challenge), the root-exponential model is still the best fit, with a smaller b (1.2077) than the previous configuration (1.2452) but still larger than the default (1.1879). We also performed the bootstrap analysis described in Section 3.1. From the bootstrap confidence intervals of the model parameters shown in Table 6.20, the confidence interval of b in the root-exponential model is also slightly larger than that for LKH with default parameters ($[1.1908, 1.2245]$ vs. $[1.1749, 1.2006]$). As seen in Figure 6.12, the root-exponential model is a very good fit up to $n = 3500$. Investigating the running times more closely, we see that this configuration brings down the running times of LKH compared to the default up to instance size $n = 3500$. In other words, the new configuration protocol seems to less overfit on the smaller instances, but still runs into problems for large instances.

Comparing the two resulted parameter settings, we believed that the configured parameter settings help LKH do a more careful check of possible moves on smaller instances, but that does not pay off for larger instances. Seeing this, we fixed KICKS, MOVE_TYPE, POPULATION_SIZE and RESTRICTED_SEARCH to their default values and performed another round of configuration following the scaling protocol. The result parameter setting was also analysed using the methodology described in Section 3.1, resulting in the three models as shown Table 6.21. We also performed the bootstrap analysis described in Section 3.1. From

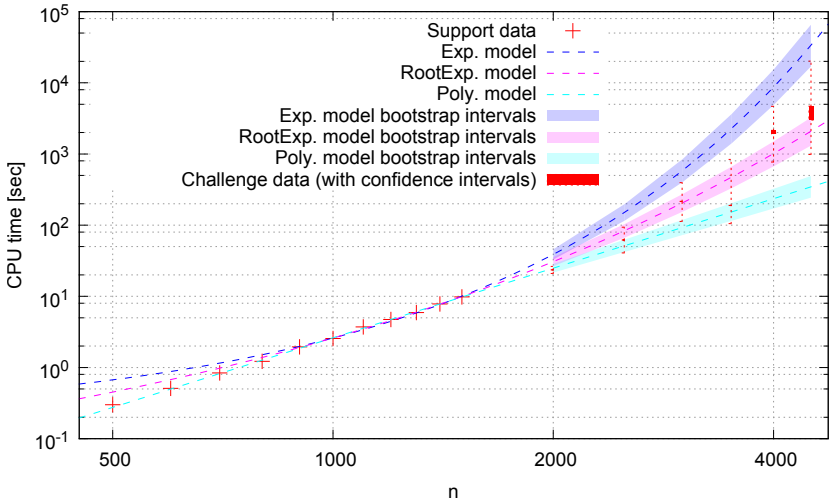


Figure 6.12: Fitted models for the medians of the running times of LKH with configured parameters from experiments following the scaling protocol. All models are fitted with the medians of the running times of LKH solving the set of RUE instances of $500 \leq n \leq 1500$ variables, and are challenged by the medians of the running times of $2000 \leq n \leq 4500$ variables.

Solver	Model		RMSE (support)	RMSE (challenge)
LKH	Exp.	0.1084×1.0027^{n}	0.14318	[4910.6, 5596.5]
	RootExp.	$\mathbf{0.0048097 \times 1.2010^{\sqrt{n}}}$	**0.15081**	[**1394.8**, **2090.6**]
	Poly.	$6.0534 \times 10^{-10} \times n^{3.1401}$	0.19236	[1764.1, 2460.9]

Table 6.21: Fitted models for the medians of the running times of LKH with configured parameters from experiments following the scaling protocol but with less parameters and the corresponding RMSE values (in CPU sec). The models yielding more accurate predictions (as per RMSEs on challenge data) are shown in boldface.

Solver	Model	Confidence interval of a	Confidence interval of b
LKH	Exp.	$[0.079825, 0.14304]$	$[1.0024, 1.0029]$
	RootExp.	$[0.0027107, 0.008814]$	$[1.1802, 1.2210]$
	Poly.	$[6.2838 \times 10^{-11}, 6.1249 \times 10^{-9}]$	$[2.8125, 3.4561]$

Table 6.22: 95% bootstrap confidence intervals for the parameters of the scaling models for EAX's running times (with configured parameters) to find optimal solutions of RUE instances.

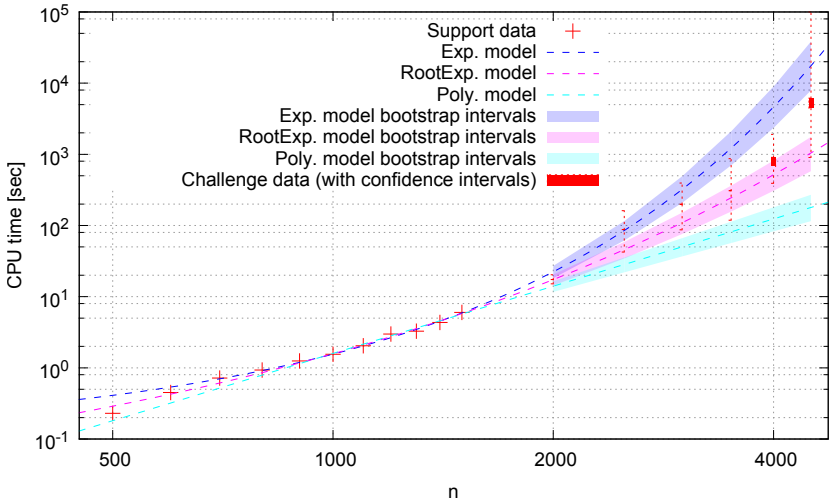


Figure 6.13: Fitted models for the medians of the running times of LKH with configured parameters from experiments following the scaling protocol but with less parameters. All models are fitted with the medians of the running times of LKH solving the set of RUE instances of $500 \leq n \leq 1500$ variables, and are challenged by the medians of the running times of $2000 \leq n \leq 4500$ variables.

the bootstrap confidence intervals of the model parameters shown in Table 6.22, the confidence interval of b in the root-exponential model is also slightly larger than that for LKH with default parameters ($[1.1802, 1.2210]$ vs. $[1.1749, 1.2006]$). Again, the root-exponential model gives the best fit according to RMSE (challenge), and fits the running times, as seen in Figure 6.13, well up to $n = 4000$. The b in the model (1.2010) is very similar to before (1.2077), with similar confidence intervals ($[1.1802, 1.2210]$ vs. $[1.1908, 1.2245]$), but remains larger than the default. Examining the running times, we noticed that the new configuration further decreases running times for $n \leq 4000$, but performs even worse for $n = 4500$. In other words, the new configuration protocol seems to even less overfit on the smaller instances, but still runs into problems for the largest instance size in our analysis

and presumably even larger sizes.

To sum up, we have attempted to configure LKH in several different settings and observed clear impact of configuration on the scaling of LKH. However, all of our attempts overfit the running times for smaller instances and lead to worse performance for instances larger than some threshold that varied with the configuration protocol used. This calls for more experiments on LKH to study the impact of its key parameters on the scaling of the solver and to examine if we can improve its scaling by adapting certain parameters with instance size. Another direction is the development of more sophisticated protocols for algorithm configuration that incorporate automated scaling analysis. Such protocols can be useful in configuring algorithms with a large number of parameters for scaling performance. We elaborate more on this idea in Section 7.2, where we discuss potential future work.

6.5 Chapter Summary

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

For Concorde, a root-exponential model of the form $a \cdot b^{\sqrt{n}}$, with b being around 1.25, was found to best describe the scaling behaviour. Compared to the scaling models for overall running times (for finding optimal solutions and proving optimality), we found that the finding times of Concorde scale slightly worse than the overall running times, which can be seen in that b is slightly larger for the model of finding time (1.25032 *vs* 1.2281); the effect is also seen in the bootstrap intervals for b ($[1.2287, 1.2793]$ *vs* $[1.2126, 1.2522]$). While this is consistent with the conclusion that the fraction of finding time goes up and approaches 1 as n increases Hoos and Stützle [36], we note that better capturing this property may require more sophisticated models with second-order terms.

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

We also investigated the impact of parameter settings and automated configuration on the scaling of algorithms. For EAX, algorithm configuration helps improve the scaling, which can be further improved by adapting the population size with instance size. For LKH, we have observed significant impact of parameter settings

on its scaling, but the state-of-the-art algorithm configurator SMAC tends to overfit the running times for smaller instances and thus produces configurations for which LKH scales worse. This calls for thorough studies on the parameters of LKH, and for the development of better configuration protocols for scaling performance.

Overall, our work complements earlier work on the scaling of state-of-the-art TSP solvers [35, 36, 21] and deepens our understandings of them. It also indicates potential for improvements in scaling behaviour through automated algorithm configuration and through setting certain parameters in dependence of features of the problem instance to be solved.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this work, we further developed a framework to analyse the empirical scaling of the performance of algorithms in a statistically sound way. The original methodology was introduced by Hoos [33], who proposed to challenge obtained models by extrapolation and to use bootstrap re-sampling to assess these models. Here, we further expanded the use of bootstrap intervals to observed running times of challenge data, and proposed a systematic way to compare scaling models for different algorithms. The former extension allows us to better capture variability of observed running time data, while the latter helps compare scaling behaviour of algorithms to draw useful conclusions. To facilitate the broad use of the methodology, we designed ESA, an automated empirical scaling analysis tool, which can be easily used as a web service or a command-line tool. ESA takes a file of running times as input, performs scaling analysis as described in Chapter 3, and generates a technical report on the scaling of the target algorithm.

We applied and extended the methodology for empirical scaling analysis to two prominent problems, SAT and TSP. For phase-transition random 3-SAT, we investigated the time complexity of three SLS-based (WalkSAT/SKC, BalancedZ and probSAT) and three DPLL-based solvers (kcnfs, march_hi and march_br). Our results show that the running times of SLS-based solvers are surprisingly well captured by polynomial models, while DPLL-based solvers clearly scale exponentially. We also compared the scaling models within each category of solvers, and found no significant difference between the SLS-based solvers we studied, but showed that the two march variants scale significantly better than kcnfs. We also demonstrated that DPLL-based solvers are faster by only a constant factor for solving satisfiable instances compared to their performance on unsatisfiable instances. In addition, we explored two distributions of 4-SAT instances. For phase-transition instances, the performance of SLS-based solvers are consistent with exponential (for BalancedZ) or root-exponential (for the other two solvers) models, while DPLL-based solvers show clear exponential scaling. For a class of less-constrained instances, the performance of both WalkSAT/SKC and kcnfs is well characterised by polynomial models.

For TSP, we investigated the scaling of running times of state-of-the-art complete solver, Concorde, as well as incomplete solvers, EAX and LKH, for finding optimal solutions without proving optimality. For Concorde, we found that the finding times scale root-exponentially, and the finding times scale slightly worse than overall running times, but more sophisticated models may be needed to better capture this. For EAX and LKH, we showed that their running times are consistent with or upper bounded by a root-exponential model, respectively, and that they scale significantly better than Concorde. We also studied the impact of automated configuration on the scaling of the two incomplete solvers and obtained positive results for EAX. For LKH, we encountered overfitting, which calls for more studies on its parameters and for the development of new configuration protocols for scaling performance.

In summary, we showed that the empirical scaling analysis methodology can be successfully applied to high-performance solvers for prominent decision or optimisation problems like SAT and TSP, and that such analysis can produce interesting and even surprising results. We note that the methodology is also applicable to other problems, and we have designed ESA to facilitate such applications. We hope that empirical scaling analysis can play a more important role in the design and analysis of algorithms and instance generators, and can even inspire new lines of theoretical analysis.

7.2 Future Work

We see several areas for future work, which potentially will lead to enhanced methodology and improved tool for empirical scaling analysis, as well as to deeper understanding of target algorithms and problems. For SAT, we see some interest in further investigating the survey propagation algorithm, which is of interest to many researchers in the community. Also of interest, as pointed out by the anonymous reviewers for our paper [52], is the empirical scaling analysis for structured SAT instances. More generally, it would be very interesting to develop empirical scaling analysis methods for classes of instances for which no instance generator is available. Such instances can be of substantial industrial or practical interest, like structured SAT and real-world non-Euclidean TSP instances. In such cases, we may have only one or a few instances for each size, and much fewer instances overall. As a result, estimations of medians or other statistics will be of much lower quality. Presumably, outlier detection will be an interesting and difficult task for scaling analysis in this context. On the methodology side, how to handle estimations of varying quality for different instance sizes is also a challenging problem. Currently, the methodology treats the estimation of all support sizes as of equal

quality, and needs to be extended if the assumption does not hold. Intuitively, the fitted model should be less influenced by an estimation obtained from fewer instances and thus being of lower quality. Bayesian methods can potentially address this situation, but further thought and thorough evaluation is clearly needed when extending the methodology to handle such situations.

Earlier work on empirical scaling analysis has involved evaluating curve bounding techniques with artificial data [48]. McGeoch et al. [49] also reported poor results when using non-linear regression for curve bounding of slow-growing functions. We suspect that this results from their focus on slow-growing functions and on curve bounding rather than fitting. Thus, it will be interesting to evaluate our methodology with this type of data. It will also be interesting to explore how to extend our methodology to empirically bounding asymptotic growth of algorithms.

One future direction for the development of our methodology and of ESA is to use nested bootstrap re-sampling when dealing with randomised algorithms and multiple runs per problem instance. Currently, the methodology only re-samples the running times for different instances. For randomised algorithms, multiple runs are performed for each instance and the running time for one instance is actually a statistic of multiple running times for the same instance. Thus, re-sampling can also be done on multiple running times for the same instance in order to capture the variability of running time on the same instance. We call this nested bootstrap re-sampling, since it involves two nested steps of bootstrap re-sampling.

Another future direction is to automatically select models from a large family of functions based on input data. This can also facilitate fitting of models with lower-order terms. One possible approach is to repeatedly fit models, first on the original data, then on the residues, in order to obtain a model with several terms. In this way, the tool will be easily applicable to an even broader range of algorithms with little human input, and may potentially produce even more interesting results. Potentially, such models can make more accurate predictions on running times for large instances, and can help bring us to another level of understanding of the algorithms. One particular example is the investigation of the scaling of finding and overall running times of Concorde, where models with lower-order terms are believed to be needed in order to model the observed running times more accurately.

In addition, we see great potential in developing automated configuration procedures for better scaling behaviour. Such procedures can make algorithm configuration even more applicable to real-world situations, as problem instances from the target distribution can take a long time to solve. This makes it infeasible to run algorithm configuration directly on these instances, because an algorithm configuration usually requires many runs of the target algorithms with different parameter settings. Some previous work has shown that new protocols are needed to configure algorithms for better scaling performance [66, 65] . As we observed when

configuring LKH, an algorithm configurator may overfit the support data. A promising area for the development of configurators is to incorporate empirical scaling analysis into algorithm configuration. To achieve that goal, our automated scaling analysis tool, ESA, should provide a solid basis. The major challenge comes from the fact the scaling analysis involves solving a large number of problem instances of different sizes and takes a long time to complete. Thus, it will be important to design a way to reduce the time, possibly by leveraging previously fitted models. One possible way is to incorporate Bayesian methods into empirical scaling analysis, with a previous model acting as the prior for model fitting.

Application of the methodology to other problems and instance distributions is also a promising area for future research. As seen in this thesis and in previous work, empirical scaling analysis has revealed interesting and surprising properties of high-performance algorithm for SAT and TSP, and we believe other problems can also benefit from such analysis. Some example problems include AI planning and scheduling, for which international competitions are regularly organised to evaluate solvers, but the methodology is applicable other problems as well. We note that empirical analysis is also of value for polynomial-time solvable problems, particularly in understanding the impact of features other than the size of instances on the empirical complexity of algorithms. Empirical investigation of the complexity of prominent algorithms for these domains can potentially enrich our understanding of these algorithms and inspire the design of new algorithms.

Finally, we envision that our methodology can be adapted to analyse properties beyond time complexity of algorithms, as long as the change in the property with some aspect (e.g., size) of problem instances is expected to follow some parametric functions. For instance, one may use similar ways to model the scaling of quality of algorithm outcomes. A more concrete example is modelling of learning curves of machine learning algorithms.

Bibliography

[1] Dimitris Achlioptas and Yuval Peres. The threshold for random k-SAT is $2^k \log 2 - O(k)$. *Journal of the American Mathematical Society*, 17(4):947–973, 2004.

[2] Rosalía Aguirre-Hernández, Holger H Hoos, and Anne Condon. Computational RNA secondary structure design: Empirical complexity and improved methods. *BMC Bioinformatics*, 8(1):34, 2007.

[3] David L Applegate, Robert E Bixby, Vasek Chvátal, and William J Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.

[4] David L Applegate, Robert E Bixby, Vasek Chvátal, and William J Cook. The traveling salesman problem, concorde TSP solver. 2012. URL `http://www.tsp.gatech.edu/concorde`. Last visited on 24 October 2015.

[5] Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5): 753–782, 1998.

[6] Adrian Balint and Uwe Schöning. Choosing probability distributions for stochastic local search and the role of make versus break. In *Proceedings of SAT 2012*, pages 16–29. Springer, 2012.

[7] Adrian Balint and Uwe Schöning. probSAT and pprobSAT. In *Proceedings of SAT Competition 2014*, page 63, 2014.

[8] Adrian Balint, Anton Belov, Matti Järvisalo, and Carsten Sinz. SAT challenge 2012 random SAT track: Description of benchmark generation. In *Proceedings of SAT Challenge 2012*, page 72, 2012.

[9] Jon L Bentley and M Douglas McIlroy. Engineering a sort function. *Software – Practice and Experience*, 23(11):1249–1265, 1993.

[10] Paul Biggar, Nicholas Nash, Kevin Williams, and David Gregg. An experimental study of sorting and branch prediction. *Journal of Experimental Algorithmics*, 12:1–8, 2008.

[11] Alfredo Braunstein, Marc Mézard, and Riccardo Zecchina. Survey propagation: An algorithm for satisfiability. *Random Structures & Algorithms*, 27(2): 201–226, 2005.

[12] bzip.org. bzip2 project homepage. URL `http://www.bzip.org/`. Last visited on 24 October 2015.

[13] Peter Cheeseman, Bob Kanefsky, and William M Taylor. Where the really hard problems are. In *Proceedings of IJCAI 1991*, pages 331–337, 1991.

[14] Cristian Coarfa, Demetrios D Demopoulos, Alfonso San Miguel Aguirre, Devika Subramanian, and Moshe Y Vardi. Random 3-SAT: The plot thickens. *Constraints*, 8(3):243–261, 2003.

[15] Amin Coja-Oghlan. The asymptotic k-SAT threshold. In *Proceedings of STOC 2014*, pages 804–813. ACM, 2014.

[16] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of STOC 1971*, pages 151–158. ACM, 1971.

[17] James M Crawford and Larry D Auton. Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence*, 81(1):31–57, 1996.

[18] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[19] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[20] Gilles Dequen and Olivier Dubois. Kcnfs: An efficient solver for random k-SAT formulae. In *Proceedings of SAT 2004*, pages 486–501. Springer, 2004.

[21] Jérémie Dubois-Lacoste, Holger H Hoos, and Thomas Stützle. On the empirical scaling behaviour of state-of-the-art local search algorithms for the Euclidean TSP. In *Proceedings of GECCO 2015*, pages 377–384. ACM, 2015.

[22] Bradley Efron and Robert J Tibshirani. *An Introduction to the Bootstrap, vol. 57 of Monographs on Statistics and Applied Probability*. New York: Chapman and Hall, 1993.

[23] John Franco and Marvin Paull. Probabilistic analysis of the Davis Putnam procedure for solving the satisfiability problem. *Discrete Applied Mathematics*, 5(1):77–87, 1983.

[24] Michael R Garey and David S Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. San Francisco, CA: Freeman, 1979.

[25] Ian P Gent and Toby Walsh. Towards an understanding of hill-climbing procedures for SAT. In *Proceedings of AAAI 1993*, pages 28–33, 1993.

[26] Ian P Gent, Ewan MacIntyre, Patrick Prosser, and Toby Walsh. The scaling of search cost. In *Proceedings of AAAI/IAAI 1997*, pages 315–320, 1997.

[27] gnu.org. gcov documentation. URL `http://gcc.gnu.org/onlinedocs/gcc/Gcov.html`. Last visited on 24 October 2015.

[28] Simon F Goldsmith, Alex S Aiken, and Daniel S Wilkerson. Measuring empirical computational complexity. In *Proceedings of ESEC/FSE 2007*, pages 395–404. ACM, 2007.

[29] Keld Helsgaun. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

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

[31] Marijn JH Heule. march_br. In *Proceedings of SAT Competition 2013*, page 53, 2013.

[32] Marjin JH Heule and Hans van Marren. march_hi: Solver description. In *Proceedings of SAT Competition 2009*, pages 23–24, 2009.

[33] Holger H Hoos. A bootstrap approach to analysing the scaling of empirical run-time data with problem size. Technical report, TR-2009-16, Department of Computer Science, University of British Columbia, 2009.

[34] Holger H Hoos and Thomas Stützle. Local search algorithms for SAT: An empirical evaluation. *Journal of Automated Reasoning*, 24(4):421–481, 2000.

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

[36] Holger H Hoos and Thomas Stützle. On the empirical time complexity of finding optimal solutions vs proving optimality for Euclidean TSP instances. *Optimization Letters*, 9:1247–1254, 2015.

[37] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of LION 2011*, pages 507–523. Springer, 2011.

[38] David Johnson, Lyle McGeoch, Cesar Rego, and Fred Glover. 8th DIMACS implementation challenge. 2001. URL `http://dimacs.rutgers.edu/Challenges/TSP/`. Last visited on 24 October 2015.

[39] Scott Kirkpatrick and Bart Selman. Critical behavior in the satisfiability of random boolean expressions. *Science*, 264(5163):1297–1301, 1994.

[40] Lefteris M Kirousis, Evangelos Kranakis, Danny Krizanc, Yannis C Stamatiou, et al. Approximating the unsatisfiability threshold of random formulas. *Random Structures and Algorithms*, 12(3):253–269, 1998.

[41] Lars Kotthoff, Pascal Kerschke, Holger Hoos, and Heike Trautmann. Improving the state of the art in inexact tsp solving using per-instance algorithm selection. In *Proceedings of LION 2015*. Springer, 2015.

[42] Daniel Kunkle. Empirical complexities of longest common subsequence algorithms. Technical report, Computer Science Department, Rochester Institute of Technology, NY, USA, 2002.

[43] Kenneth Levenberg. A method for the solution of certain problems in least squares. *Quarterly of Applied Mathematics*, 2:164–168, 1944.

[44] Chumin Li, Chong Huang, and Ruchu Xu. Balance between intensification and diversification: a unity of opposites. In *Proceedings of SAT Competition 2014*, pages 10–11, 2014.

[45] Xiaoming Li, María Jesús Garzarán, and David Padua. A dynamically tuned sorting library. In *Proceedings of CGO 2004*, pages 111–122. IEEE, 2004.

[46] Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Csch-based portfolio using ccsat and march. In *Proceedings of SAT Competition 2013*, page 28, 2013.

[47] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics*, 11 (2):431–441, 1963.

[48] Catherine McGeoch, Doina Precup, and Paul R Cohen. How to find big-oh in your data set (and how not to). In *Proceedings of IDA 1997*, pages 41–52. Springer, 1997.

[49] Catherine McGeoch, Peter Sanders, Rudolf Fleischer, Paul R Cohen, and Doina Precup. Using finite experiments to study asymptotic performance. In *Experimental Algorithmics*, pages 93–126. Springer, 2002.

[50] Stephan Mertens, Marc Mézard, and Riccardo Zecchina. Threshold values of random k-SAT from the cavity method. *Random Structures and Algorithms*, 28(3):340–373, 2006.

[51] David Mitchell, Bart Selman, and Hector Levesque. Hard and easy distributions of SAT problems. In *Proceedings of AAAI 1992*, pages 459–465, 1992.

[52] Zongxu Mu and Holger H Hoos. On the empirical time complexity of random 3-SAT at the phase transition. In *Proceedings of IJCAI 2015*, pages 367–373, 2015.

[53] Zongxu Mu and Holger H Hoos. Empirical scaling analyser: An automated system for empirical analysis of performance scaling. In *Companion Publication of GECCO 2015*, pages 771–772, 2015.

[54] Zongxu Mu, Jérémie Dubois-Lacoste, Holger H Hoos, and Thomas Stützle. On the empirical scaling for finding optimal solutions of the TSP problem. *In preparation*, 2015.

[55] Zongxu Mu, Holger H Hoos, and Thomas Stützle. The impact of automated algorithm configuration on the scaling behaviour of state-of-the-art incomplete tsp solvers. *In preparation*, 2015.

[56] Yuichi Nagata. Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem. In *Proceedings of ICGA 1997*, pages 450–457, 1997.

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

[58] Christos H Papadimitriou. The euclidean travelling salesman problem is np-complete. *Theoretical Computer Science*, 4(3):237–244, 1977.

[59] Andrew J Parkes and Joachim P Walser. Tuning local search for satisfiability testing. In *Proceedings of AAAI/IAAI 1996*, pages 356–362, 1996.

[60] Olivier Roussel. Controlling a solver execution with the runsolver tool system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7: 139–144, 2011.

[61] Peter Sanders and Rudolf Fleischer. Asymptotic complexity from experiments? A case study for randomized algorithms. In *Algorithm Engineering*, pages 135–146. Springer, 2001.

[62] SatCompetition.org. The international SAT competitions web page, 2014. URL `http://www.satcompetition.org/`. Last visited on 24 October 2015.

[63] Bart Selman, Hector J Levesque, David G Mitchell, et al. A new method for solving hard satisfiability problems. In *Proceedings of AAAI 1992*, pages 440–446, 1992.

[64] Bart Selman, Henry A Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proceedings of AAAI 1994*, pages 337–343, 1994.

[65] James Styles and Holger Hoos. Ordered racing protocols for automatically configuring algorithms for scaling performance. In *Proceedings of GECCO 2013*, pages 551–558. ACM, 2013.

[66] James Styles, Holger H Hoos, and Martin Müller. Automatically configuring algorithms for scaling performance. In *Proceedings of LION 2012*, pages 205–219. Springer, 2012.

[67] K Subramani and Dejan Desovski. On the empirical efficiency of the vertex contraction algorithm for detecting negative cost cyles in networks. In *Proceedings of ICCS 2005*, pages 180–187. Springer, 2005.

[68] Makoto Yokoo. Why adding more constraints makes a problem easier for hill-climbing algorithms: Analyzing landscapes of CSPs. In *Proceedings of CP 1997*, pages 356–370. Springer, 1997.

Appendix A

LaTeX Template for ESA

```
%% template-AutoScaling.tex
%% Author: Zongxu Mu
%% This is the LaTeX template file for Empirical Scaling
   Analyser (ESA).
%% ESA takes the template, replace variables with their
   corresponding values,
%%      and generates an output file named AutoScaling.
   tex.
%% You may compile this file alone without running ESA to
    see how the output looks like.
%% Variables are surrounded by ``@@''s
%% Supported variable names include:
%%      - algName, e.g., ``WalkSAT/SKC''
%%      - instName, e.g., ``random 3SAT at phase
   transition''
%%      - models, e.g., ``\\begin{itemize}\n\\item $Exp\\
   left[a,b\\right]\\left(n\\right)=a\\cdot b^{n}$ \\
   quad{}(2-parameter exponential);\n\\end{itemize}''
%%      - numBootstrapSamples, the number of bootstrap
   samples used in the analysis, e.g., 1000
%%      - numSizes, the number of sizes used in the
   analysis, e.g., 12
%%      - largestSupportSize, e.g., 500
%%      - table-Details-dataset, e.g., ``\\input{
   table_Details-dataset}''
%%      - table-Details-dataset, e.g., ``\\input{
   table_Details-dataset}''
%%      - table-Fitted-models, e.g., ``\\input{
   table_Fitted-models}''
%%      - figure-fittedModels, e.g., ``\\includegraphics[
   width=0.8\textwidth]{fittedModels_loglog}''
%%      - supportSizes, the sizes used for fitting the
   models, e.g., ``200, 250, 300, 350, 400, 450, 500''
%%      - challengeSizes, e.g., ``600, 700, 800, 900,
   1000''
%%      - table-Bootstrap-intervals-of-parameters, e.g.,
```

```
    ``\\input{table_Bootstrap-intervals-of-parameters}''
%%      - table-Bootstrap-intervals, e.g., ``\\input{
    table_Bootstrap-intervals}''
%%      - analysisSummary, e.g., ``observed median
    running times are consistent with the polynomial
    scaling model''
\documentclass[british]{article}
\usepackage[T1]{fontenc}
\usepackage[latin9]{inputenc}
\usepackage{geometry}
\geometry{verbose,tmargin=3.5cm,bmargin=3.5cm,lmargin=3cm
    ,rmargin=3cm}
\usepackage{array}
\usepackage{multirow}
\usepackage{amstext}
\usepackage{graphicx}
\usepackage{color}

\newcommand{\medianInterval}[1]{}
@@customCommands@@

\makeatletter

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% LyX specific LaTeX
    commands.
%% Because html converters don't know tabularnewline
\providecommand{\tabularnewline}{\\}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% User specified LaTeX
    commands.

\title{On the empirical scaling of running time of
    @@algName@@ for solving @@instName@@}
\author{Empirical Scaling Analyser}

\makeatother

\usepackage{babel}
\begin{document}
\maketitle %


\section{Introduction}

This is the automatically generated report on the
```

```
    empirical scaling
of the running time of @@algName@@ for solving
    @@instName@@.


\section{Methodology}

\label{sec:Methodology}

% models, model fitting
For our scaling analysis, we considered the following
    parametric models:
@@models@@
% \begin{itemize}
% \item $Exp\left[a,b\right]\left(n\right)=a\cdot b^{n}$
    \quad{}(2-parameter exponential);
% \item $RootExp\left[a,b\right]\left(n\right)=a\cdot b
    ^{\sqrt{n}}$ \quad{}(2-parameter root-exponential);
% \item $Poly\left[a,b\right]\left(n\right)=a\cdot n^{b}$
     \quad{}(2-parameter polynomial).
% \end{itemize}
Note that the approach could be easily extended to other
    scaling models.
For fitting parametric scaling models to observed data,
    we used the
non-linear least-squares Levenberg-Marquardt algorithm.

% what we fitted the models to, how we assessed model fit
Models were fitted to performance observations in the
    form of @@statistic@@s
of the distributions of running times over sets of
    instances for given
$n$, the instance size.
%Compared to the mean, the median has two
%advantages: it is statistically more stable and immune
    to the presence
%of a certain amount of timed-out runs.
To assess the fit of a given
scaling model to observed data, we used root-mean-square
    error (RMSE).

\medianInterval{
Due to the instances for which the running times are
    unknown,
there is uncertainty about the
```

```
precise location of the @@statistic@@s of the running
   time distributions
at each such $n$, and we can only provide bounds on
those @@statistic@@s instead. Closely following \cite{
   dubois2015on}, we calculate
these bounds based on the best
and worst case scenarios, in which all instances with
   unknown running times
are easiest or hardest, respectively.
We note that these are not confidence
intervals, since we can guarantee the actual
   @@statistic@@ running
times to lie within them.
We also calculate RMSEs and confidence intervals based on
    these bounds.
}

% bootstrap confidence intervals
Closely following \cite{hoos2009bootstrap,
   hoos2014empirical}, we
computed 95\% bootstrap confidence intervals for the
   performance predictions
obtained from our scaling models, based on
   @@numBootstrapSamples@@ bootstrap samples
per instance set and @@numBootstrapSamples@@
   automatically fitted variants of each scaling
model.
In the following, we say that a scaling model is in-
consistent with observed data if the bootstrap confidence
    interval for the observed data
is disjoint from the bootstrap confidence interval for
predicted running times;
\medianInterval{we say that a scaling model is consistent
with observed data, if the interval for observed medians
overlaps with, but is not fully contained within the
bootstrap confidence interval for predicted running times
   ;}
we say that a scaling model is strongly consistent with
   observed
data, if the observed median is fully contained
within the bootstrap confidence interval for predicted
running times.
Also, we define residue of a model at a given size as the
    observed
point estimate less the predicated value.
```

```
\section{Dataset Description}

The dataset contains running times of the @@algName@@
   algorithm solving
@@numSizes@@ sets of instances of different sizes. We
   split the running times into
two categories, support ($n\leq@@largestSupportSize@@$)
   and challenge ($n>@@largestSupportSize@@$). The
details of the dataset can be found in Tables \ref{tab:
   Details-dataset-support}
and \ref{tab:Details-dataset-challenge}.
\begin{table*}
\noindent \begin{centering}
@@table-Details-dataset-support@@
\par\end{centering}

\caption{\label{tab:Details-dataset-support} Details of
   the running time dataset used as support data for
   model fitting.}
\end{table*}

\begin{table*}
\noindent \begin{centering}
@@table-Details-dataset-challenge@@
\par\end{centering}

\caption{\label{tab:Details-dataset-challenge} Details of
    the running time dataset used as challenge data for
   model fitting.}
\end{table*}

%
% Figure \ref{fig:CDFs} shows the distributions of the
   running times of
% @@algName@@ solving @@instName@@.
% \begin{figure*}[tb]
% \begin{centering}
% @@figure-cdfs@@
% % \includegraphics[width=0.8\textwidth]{cdfs}
% \par\end{centering}
%
% \noindent \centering{}\caption{\label{fig:CDFs}
   Distribution of running times across instance sets
   for
```

```
% @@algName@@.}
% \end{figure*}
%
%

\section{Empirical Scaling of Solver Performance}

\label{sec:Results}

We first fitted our parametric scaling models to the
   @@statistic@@s of the running times
of @@algName@@, as described in Section \ref{sec:
   Methodology}. The
models were fitted using the @@statistic@@s of the
   running times for $@@supportSizes@@$
(support) and later challenged with the @@statistic@@s of
    the running times for $@@challengeSizes@@$.
This resulted in the models, shown along with RMSEs on
   support and
challenge data, shown in Table~\ref{tab:Fitted-models}.
\begin{table}[tb]
\begin{centering}
@@table-Fitted-models@@
% \input{table_Fitted-models}
\par\end{centering}

\caption{\label{tab:Fitted-models}Fitted models for the
   @@statistic@@s of the running times and RMSE
values (in CPU sec). The models yielding more
accurate predictions (as per RMSEs on challenge data) are
    shown in
boldface.}
\end{table}
In addition, we illustrate the fitted models of
   @@algName@@ in Figure~\ref{fig:Fitted-models},
and the residues for the models in Figure~\ref{fig:Fitted
   -residues}.
\begin{figure}[tb]
\noindent \begin{centering}
@@figure-fittedModels@@
% \includegraphics[width=0.8\textwidth]{fittedModels}
\par\end{centering}

\caption{\label{fig:Fitted-models} Fitted models for the
   @@statistic@@s of the running times.
```

```
The models are fitted with the @@statistic@@s of the
   running times of
@@algName@@ solving the set of @@instName@@
of $@@supportSizes@@$ variables, and are challenged by
   the @@statistic@@s of the
running times of $@@challengeSizes@@$ variables.}
\end{figure}


\begin{figure}[tb]
\noindent \begin{centering}
@@figure-fittedResidues@@
% \includegraphics[width=0.8\textwidth]{fittedResidues}
\par\end{centering}

\caption{\label{fig:Fitted-residues} Residues of the
   fitted models for the @@statistic@@s
of the running times. }
\end{figure}


But how much confidence should we have in these models?
   Are the RMSEs
small enough that we should accept them? To answer this
   question,
we assessed the fitted models using the bootstrap
   approach outlined
in Section~\ref{sec:Methodology}. Table~\ref{tab:
   Bootstrap-intervals-of-parameters}
shows the bootstrap intervals of the model parameters,
and Table~\ref{tab:Bootstrap-intervals-support}
contains the bootstrap intervals for the support data.
Challenging the models with extrapolation, as shown in
Table~\ref{tab:Bootstrap-intervals-challenge}, it is
   concluded that
@@analysisSummary@@
(as also illustrated in Figure~\ref{fig:Fitted-models}).
\begin{table*}[tb]
\noindent \begin{centering}
@@table-Bootstrap-intervals-of-parameters@@
% \input{table_Bootstrap-intervals-of-parameters}

\par\end{centering}
\caption{\label{tab:Bootstrap-intervals-of-parameters}
   95\% bootstrap intervals
```

```
of model parameters for the @@statistic@@s of the running
    times}
\end{table*}
\begin{table*}[tb]
\noindent \begin{centering}
@@table-Bootstrap-intervals-support@@
% \input{table_Bootstrap-intervals}
\par\end{centering}

\caption{\label{tab:Bootstrap-intervals-support} 95\%
   bootstrap confidence intervals
for the @@statistic@@s of the running time predictions
   and observed running times on @@instName@@.
The instance sizes shown here are those used for fitting
   the models.
Bootstrap intervals on predictions that are weakly
   consistent
with the observed point estimates are shown in boldface,
those that are consistent are marked by plus signs ({+}),
and those that fully contain the confidence intervals on
observations are marked by asterisks ({*}).}
\end{table*}

\begin{table*}[tb]
\noindent \begin{centering}
@@table-Bootstrap-intervals-challenge@@
% \input{table_Bootstrap-intervals}
\par\end{centering}

\caption{\label{tab:Bootstrap-intervals-challenge} 95\%
   bootstrap confidence intervals
for the @@statistic@@s of the running time predictions
   and observed running times on @@instName@@.
The instance sizes shown here are larger than those used
   for fitting the models.
Bootstrap intervals on predictions that are weakly
   consistent
with the observed data are shown in boldface,
\medianInterval{those that are consistent are marked by
   plus signs ({+}),}
those that are strongly consistent are marked
by sharps ({\#}),
and those that fully contain the confidence intervals on
observations are marked by asterisks ({*}).}
\end{table*}
```

```
\section{Conclusion}

In this report, we presented an empirical analysis of the
    scaling
behaviour of @@algName@@ on @@instName@@. We found
@@analysisSummary@@.

\bibliographystyle{plain}
\begin{thebibliography}{1}

\bibitem{dubois2015on}
J{\'e}r{\'e}mie Dubois-Lacoste, Holger~H. Hoos, and
   Thomas St{\"u}tzle.
\newblock On the empirical scaling behaviour of state-of-
   the-art local search
  algorithms for the euclidean tsp.
\newblock In {\em GECCO}. ACM, 2015.

\bibitem{hoos2009bootstrap}
Holger~H Hoos.
\newblock A bootstrap approach to analysing the scaling
   of empirical run-time
  data with problem size.
\newblock Technical report, Technical Report TR-2009-16,
   University of British
  Columbia, 2009.

\bibitem{hoos2014empirical}
Holger~H Hoos and Thomas St{\"u}tzle.
\newblock On the empirical scaling of run-time for
   finding optimal solutions to
  the travelling salesman problem.
\newblock {\em European Journal of Operational Research},
    238(1):87--94, 2014.

\end{thebibliography}

\end{document}
```

Appendix B

Gnuplot Templates for ESA

B.1 Gnuplot Template for Plotting Models

```
#!/gnuplot

set terminal pdfcairo dashed fontscale 0.5
set termopt enhanced
set logscale
set xlabel 'n'
set ylabel 'CPU time [sec]'
set format y "10^{%T}"
set key top left
set xtics auto
set grid xtics ytics mxtics mytics lc rgb '#999999' lw 1
   lt 0
set style fill transparent solid 0.2 noborder
```

B.2 Gnuplot Template for Plotting Residue Curves

```
#!/gnuplot

set terminal pdfcairo dashed fontscale 0.5
set termopt enhanced
set print "plot-residues.log"

set xlabel 'n'
set ylabel 'Residue [sec]'
set xtics auto
set key left bottom
set grid xtics ytics mxtics mytics lc rgb '#999999' lw 1
   lt 0
set style fill transparent solid 0.2 noborder
```