5. Using Finite Experiments to Study Asymptotic Performance

Catherine McGeoch[1], Peter Sanders[2,*], Rudolf Fleischer[3], Paul R. Cohen[4], and Doina Precup[4]

[1] Amherst College, Amherst, MA, USA
ccm@cs.amherst.edu

[2] Max-Planck-Institut für Informatik, Saarbrücken, Germany
sanders@mpi-sb.mpg.de

[3] The Hong Kong University of Science and Technology, Hong Kong
rudolf@cs.ust.hk

[4] University of Massachussetts, Amherst, MA
{dprecup,cohen}@cs.umass.edu

Summary.

In the analysis of algorithms we are interested in obtaining closed form expressions for algorithmic complexity, or at least asymptotic expressions in $\mathcal{O}(\cdot)$-notation. It is often possible to use experimental results to make significant progress towards this goal, although there are fundamental reasons why we cannot guarantee to obtain such expressions from experiments alone. This paper investigates two approaches relating to problems of developing theoretical analyses based on experimental data.

We first consider the scientific method, which views experimentation as part of a cycle alternating with theoretical analysis. This approach has been very successful in the natural sciences. Besides supplying preliminary ideas for theoretical analysis, experiments can test falsifiable hypotheses obtained by incomplete theoretical analysis. Asymptotic behavior can also sometimes be deduced from stronger hypotheses which have been induced from experiments. As long as complete mathematical analyses remains elusive, well tested hypotheses may have to take their place. Several examples are given where average complexity can be tested experimentally so that support for hypotheses is quite strong.

A second question is how to approach systematically the problem of inferring asymptotic bounds from experimental data. Five heuristic rules for "empirical curve bounding" are presented, ogether with analytical results guaranteeing correctness for certain families of functions. Experimental evaluations of the correctness and tightness of bounds obtained by the rules for several constructed functions and real datasets are described.

5.1 Introduction

The complexity analysis of algorithms is one of the core activities of computer scientists, especially in the branch of theoretical computer science known as algorithmics. The ultimate goal would be to find closed form expressions for the runtime (or other measures of resource consumption), in terms of

[*] Partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

R. Fleischer et al. (Eds.): Experimental Algorithmics, LNCS 2547, pp. 93–126, 2002.
© Springer-Verlag Berlin Heidelberg 2002

input parameters of interest. Since this is usually too complicated, we are often content with asymptotic expressions for the worst case performance depending on a small number of input parameters like problem size, which are usually presented in $\mathcal{O}(\cdot)$-notation. Even this task can be very difficult so it is important to use all available tools.

In this paper we investigate the empirical version of this primary activity – how to use finite experimental data to shed insight on universal asymptotic properties of algorithms. We illustrate both the promise and the difficulties inherent in the use of experiments to suggest, support, and refute hypotheses about asymptotic behavior. Experimental data can be employed for asymptotic analysis both indirectly – for example, in support of conjectures necessary to theoretical arguments; and directly, by extrapolation of trend data beyond the range of experimentation. In the latter scenario, we consider a specific problem, which we call *empirical curve-bounding*: given a set of data points (N_i, Y_i) obtained from an experiment in which $Y_i = f(N_i)$, for some unknown function $f(n)$, find complexity classes $O(g_u(n))$ and/or $\Omega(g_l(n))$ to which $f(n)$ belongs.

This paper has two goals. The first is to show how, with some care, it is possible to obtain good insights about asymptotic trends, based on analyses of data obtained from experiments. One way to make the meaning of "some care" more precise is to apply the terminology of the scientific method [5.31]. The scientific method views science as a cycle between theory and practice. Theory can inductively or (partially) deductively[1] formulate falsifiable hypotheses which can be tested by experiments. The results may then yield new or refined hypotheses. This mechanism is widely accepted in the natural sciences and is often viewed as a key to the success of these disciplines. We present four examples of ways in which the scientific method can be applied to the use of experimentation to advance the goals of asymptotic algorithm analysis, using problems in parallel disk scheduling, random polling, shellsort, and randomized process allocation.

The second goal is to evaluate a collection of curve-bounding techniques, in order to identify their practical limitations. Unfortunately, no data analysis method for inferring asymptotic trends in data can be guaranteed correct for all data sets: to see this, note that for any finite vector of problem sizes, there are functions of arbitrarily high degree that are indistinguishable from the constant function c at those problem sizes. Therefore any algorithm for this problem must be regarded as a heuristic that sometimes fails. We desire robust heuristics that produce correct bound estimates (or clear indications of failure) for broad classes of functions and for functions that tend to arise in practice.

We describe five simple heuristics (or rules) for curve bounding, and a hybrid rule that handles some specific pathologies. For each of the five rules,

[1] Inductive reasoning draws general conclusions from specific data whereas deductive reasoning draws specific conclusions from general statements.

we present analytical results guaranteeing correctness for certain families of functions. Then, using a variety of algorithmic data sets, we evaluate the rules in "typical" and in near-pathological situations. Negative results concerning two plausible rules that turned out to have high failure rates are also presented.

In our informal and designed experiments with little or no random noise in the data, all the rules generally provide correct asymptotic bounds that are within about a $\sqrt{n}$ factor of the true asymptotic bound. The reliability of the rules deteriorates, however, in the presence of random variation in the data, and/or when too-large constants or negative coefficients appear in second-order terms. Fortunately it is usually easy in algorithmic problems to reduce the noise problem by taking more experiments or applying variance reduction techniques during experimentation. It is of course possible to reduce the effect of large second-order terms by taking larger problem sizes, but the rules can be slow to respond to this type of change. A hybrid diagnostic method described in Section 5.6 can be used with success on such problems.

This explicit study of techniques for curve-bounding appears to be completely new. We can find no techniques in the statistical and data analysis literature specifically designed for finding asymptotic bounds on data, although much is known about fitting curves to data. As we shall demonstrate, good algorithms for curve fitting are not always best for curve bounding, and vice versa.

The importance of experiments in algorithm design and analysis has gained much attention in the past decade. New workshops (ALENEX, WAE) and journals (ACM Journal of Experimental Algorithmics) have been installed, and established conferences (e.g., SODA, ESA) explicitly call for experimental work. Several articles [5.4], [5.19], [5.27], [5.28]) present guidelines for performing experiments on algorithmic research problems, and one book [5.12] presents methods of data analysis in the context of experimentation on heuristic algorithms. Using the scientific method as a basis for algorithmics was proposed by Hooker [5.17], but similar ideas concerning experimental computer science in general can also be found in other papers [5.14, 5.15, 5.3, 5.37, 5.16, 5.23, 5.29, 5.41].

Section 5.2 reviews the main difficulties in experimental algorithmics and explains how to partially solve them. Section 5.3 gives several concrete examples of using experimental results to suggest, support, or to falsify hypotheses about algorithmic performance. The algorithms presented in this section are randomized, with expected resource consumption dependent only on input size so that many repeated experiments give us rather accurate information on average behavior. On the other hand, all the algorithms are nontrivial to analyze analytically. It turns out that in this situation the scientific method with a close, problem specific interaction between theoretical and experimental reasoning yields quite accurate insight on the asymptotic behavior of the algorithm. For example, in Section 5.3.2 we are able to resolve even an additive $\mathcal{O}(\log\log n)$ term.

We then turn to a systematic evaluation of rules for the empirical curve-bounding problem. Section 5.4 presents each rule R, together with a "justification" that describes a class of functions for which the rule is guaranteed correct. Section 5.5 presents an empirical study of the rules using data sets from constructed parameterized functions. We observe that some rules are sensitive to large lower order terms and some to random noise, and some to both. Most of the rules are surprisingly unresponsive to changes in the largest problem size. One rule produces bounds that are rarely incorrect and rarely tight. A second collection of data comes from eight experimental studies of algorithms, to assess performance on "typical" algorithmic problems. In three cases there is at least a logarithmic gap in known analytical bounds, and we show how the rules can (and cannot) be used to support conjectures that tighten the gaps.

Section 5.4 assumes some familiarity with data analysis terms such as *correlation coefficient*, *least-squares regression*, and *residuals*, which may be found in any introductory statistics textbook. For introductions to the curve-fitting methods adapted here for curve-bounding, see Atkinson [5.1], Cohen [5.12], Chambers et al. [5.11], Rawlins [5.33], or Tukey [5.42]. Algorithms for domain-independent function finding [5.36] might be adapted to curve bounding but are not considered here.

Finally, Section 5.7 discusses the role of the scientific method in the context of experimental analysis of data and summarizes our observations about curve-bounding rules.

We emphasize that this work represents a small initial investigation of a potentially large research area. This paper only scratches the surface of a related important methodological topic, namely how to perform experiments on algorithms, and how to evaluate the confidence in our findings statistically. Our analyses are far from complete, and we do not consider here many interesting methodological and statistical questions, function classes, function parameters, rule variations, or multivariate problems.

In specific examples, we mostly consider cases where it is of interest to bound the complexity of algorithms for inputs of size n, using functions of the single parameter n. Later sections emphasizing data analysis use the symbol x in place of n, to refer to the "control parameter" in the experiment, but again we assume that only one such control parameter is present. Issues of experimentation with combinations of control parameters is outside the scope of this paper.

Of course, many problems in experimental evaluation include combinations of parameters (such as problem size n, graph density d, and algorithm tuning parameter p). But these problems can sometimes be studied by varying each parameter in turn while holding others fixed.

5.2 Difficulties with Experimentation

There is no question that experimental analysis of algorithms presents several fundamental problems to the researcher. Some of the major difficulties are surveyed in this section.

Too Many Inputs. Perhaps the most fundamental problem with algorithmic experimentation is that we can rarely test all possible inputs, even for bounded input size, because there are usually exponentially (or infinitely) many of them. In application-oriented research this problem may be mitigated by collections of test instances which are considered "typical".[2] For example, there is a large class of *oblivious* algorithms where the execution time only depends on a small number of parameters like the input size, for example, matrix multiplication. Although many oblivious algorithms are easy to analyze directly, experiments can sometimes help. Furthermore, there are algorithmic problems with few inputs. For example, the locality properties of several space filling curves were first found experimentally and then proven analytically. Later it turned out that a class of experiments can be systematically converted into theoretical results valid for arbitrary curve sizes [5.30].

But in most cases there are far too many instances to allow exhaustive testing. In these situations, our rich statistical understanding of random sampling makes algorithm randomization and average case analyses most important for experimentation. Randomization can be used to convert a hypothesis about "all instances" into one about behavior "on average," for which experimental approaches are most suited. For example, every sorting algorithm which is efficient on average can be transformed into an algorithm for worst-case instances by permuting the inputs randomly. In this case, a few hundred experimental trials with random inputs can give a reliable picture of the expected performance of the algorithm for inputs of a given size. On the other hand, closed form analyses of randomized algorithms can be very difficult to obtain. For example, the average performance of randomized Shellsort has been open for a long time [5.38]. Section 5.3.3 presents an experimental study of Shellsort.

Unbounded Input Size. Another problem with experiments is that we can only test a finite number of input *sizes*. As a result, no inference about asymptotic behavior is reliable. For example, assume we observe that some sorting algorithm needs an average of $C(n) \leq 3n \log n$ comparisons[3] for $n < 10^6$ elements. We cannot claim that $C(n) \leq 3n \log n$ as a theorem, since quadratic behavior might set in for $n > 42 \cdot 10^6$. Here, the scientific method partially saves the situation. We can formulate the hypothesis $C(n) \leq 3n \log n$, which is scientifically sound since it can be falsified by presenting an instance of size n with $C(n) > 3n \log n$.

[2] For example, a list with 23 collections of problem instances can be found under `http://mat.gsia.cmu.edu/Resources/Problem_Instances/`

[3] Throughout this paper $\log x$ stands for the base two logarithm $\log_2 x$.

Note that not every sound hypothesis is a good hypothesis. For example, we would be cowardly to change the above hypothesis to $C(n) \leq 100000n \log n$, since it would be difficult to falsify it even if it later turns out that the true bound is $C(n) = n \log n + 0.1n \log^2 n$. Issues like accuracy, simplicity, and generality of hypotheses also arise in the natural sciences and should not be obstacles to the use of the scientific method here.

$\mathcal{O}(\cdot)$-s are not Falsifiable. The next problem is that an asymptotic expression cannot be used directly in formulating a scientific hypothesis since it could never be falsified experimentally. For example, if we claim that a certain sorting algorithm needs at most $C(n) \in \mathcal{O}(n \log n)$ comparisons it cannot even be falsified by a set of inputs which clearly indicate quadratic behavior, since we could always claim that this quadratic development would stop for sufficiently large inputs. This problem can be solved by formulating a hypothesis which is stronger than the asymptotic expression we really have in mind. The hypothesis $C(n) \leq 3n \log n$ used above is a trivial example. A less trivial example is given in the study of Shellsort in Section 5.3.3.

Complexity of the Machine Model. Although the actual execution time of an algorithm is perhaps the most interesting subject of analysis, this measure of resource consumption is often difficult to model by closed form expressions. Caches, virtual memory, memory management, compilers, and interference from other processes all influence execution time in ways that are difficult to predict.[4] At some loss of accuracy, this problem can be solved by counting the number of times a certain set of source code operations (which cover all the inner loops of the program) is executed. This count often suffices to capture the asymptotic behavior of the code in a machine-independent way. For example, for comparison-based sorting algorithms it is usually sufficient to count the number of key comparisons.

Finding Hypotheses. Except in very simple cases, it is almost impossible to guess exactly an appropriate formula for a worst case performance, given only measurements, even when the investigated resource consumption only depends on input size. For example, the measured function may be non-monotonic but we are only interested in a monotonic upper bound. There are often considerable contributions of lower order terms for small inputs. Indeed our experience described in later sections shows that simple fitting methods sometimes just won't work, especially if we are interested in fine distinctions like logarithmic factors.

In some cases the scientific method can help to mitigate this difficulty by applying problem-specific information to the study. We may be able to handle a related or simplified version of the system analytically, or we can

[4] Remember that the above is also an argument *in favour* of doing experiments because the full complexity of the hardware is difficult to model theoretically. We only mention it as a problem in the current context of inducing asymptotic expressions from experiments.

make "heuristic" steps in a derivation of a theoretical bound. Although the result is not a theorem about the target system, it is good enough as a hypothesis about its behavior in the sense of the scientific method. Section 5.3 gives several examples of this powerful approach which so far seems to be underrepresented in algorithmics.

5.3 Promising Examples

Our first example in Section 5.3.1 can be viewed as the traditional use of experiments as a method to generate conjectures on the behavior of algorithms — but it has an additional interpretation in the sense that experiment plus theory (on a less attractive algorithm) yields a useful hypothesis. Section 5.3.2 gives an example in the same category but using a less well known approach. Rather than simplifying the algorithm, we simplify the analysis by making simplifying assumptions (independence) in the middle of the derivation. The resulting bound has the status of a theory in the sense of the scientific method and is then validated by simulation. Sections 5.3.3 and 5.3.4 touch on the difficult question of how to use experiments to learn something about the asymptotic complexity of an algorithm. Finally Section 5.3.4 is a good example how experiments can suggest that an analysis can be sharpened.

5.3.1 Theory with Simplifications: Writing to Parallel Disks

Consider the following algorithm, EAGER, for writing D randomly allocated blocks of data to D parallel disks. EAGER is an important ingredient of a general technique for scheduling parallel disks [5.35]. We maintain one queue Q_i for each disk. The queues share a buffer space of size $W \in \mathcal{O}(D)$. We first put all the blocks into the queues and then write one block from each nonempty queue. When the sum of the queue lengths exceeds W, additional write steps are invested. We have no idea how to analyze this algorithm. Therefore, in [5.35] a different algorithm, THROTTLE, is proposed that only admits $(1-\epsilon)D$ blocks per time step to the buffers. Then it is quite easy to show using queuing theory that the expected sum of the queue lengths is close to $D/(2\epsilon)$. Further, it can be shown that the sum of the queue lengths is concentrated around its mean with high probability so that a slightly larger buffer suffices to make waiting steps rare.[5]

Still, in many practical situations EAGER is not only simpler but also somewhat more efficient. Was the theoretical analysis futile and misguided? One of the reasons why we think the theory is useful is that it suggests a nice explanation of the measurements shown in Fig. 5.1. It looks like $1 - D/(2W)$

[5] The current proof shows that $W \in \mathcal{O}(D/\epsilon)$ suffices but we conjecture that this can be sharpened considerably using more detailed calculations.

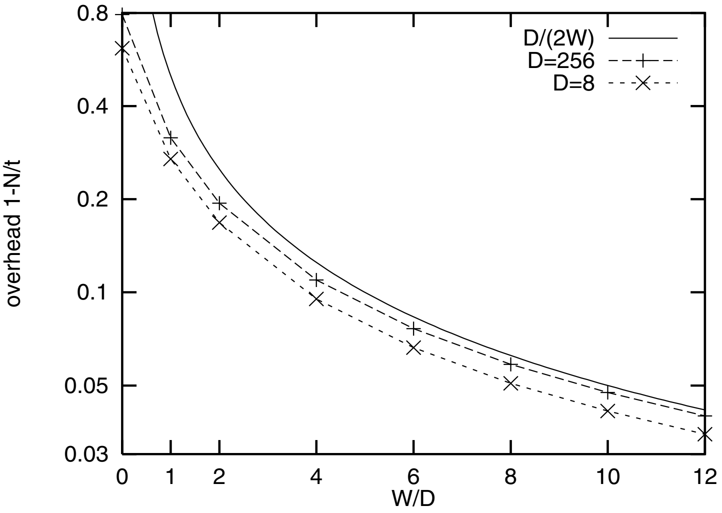

Fig. 5.1. Inefficiency (i.e., 1−efficiency) of EAGER. $N = 10^6 \cdot D$ blocks were written

is a lower bound for the average efficiency of EAGER and a quite tight one for large D. This curve was not found by fitting a curve but by the theoretical observation that algorithm THROTTLE with $\epsilon = D/(2W)$ would have buffer requirement about W.

More generally speaking, the algorithms we are most interested in might be too difficult to understand analytically. In such cases it makes sense to analyze a related and possibly inferior algorithm, and to use the scientific method to develop theoretical insights about the original algorithm. In the next Section we see that rather than simplifying the algorithm we can also simplify the analysis and achieve a similar effect — a theory in the sense of the scientific method.

5.3.2 "Heuristic" Deduction: Random Polling

Let us consider the following simplified model for the startup phase of *random polling dynamic load balancing* [5.21, 5.9, 5.34] which is perhaps the best available algorithm for parallelizing tree shaped computations of unknown structure: There are n processing elements (PEs) numbered 0 through $n-1$. At step $t = 0$, a random PE is busy while all other PEs are idle. In step t, a random shift $k \in \{1, \ldots, n-1\}$ is determined and the idle PE with number i asks PE $i + k \bmod n$ for work. Idle PEs which ask idle PEs remain idle; all others are busy now. How many steps T are needed until all PEs are busy? A trivial lower bound is $T \geq \log n$ steps since the number of busy PEs can

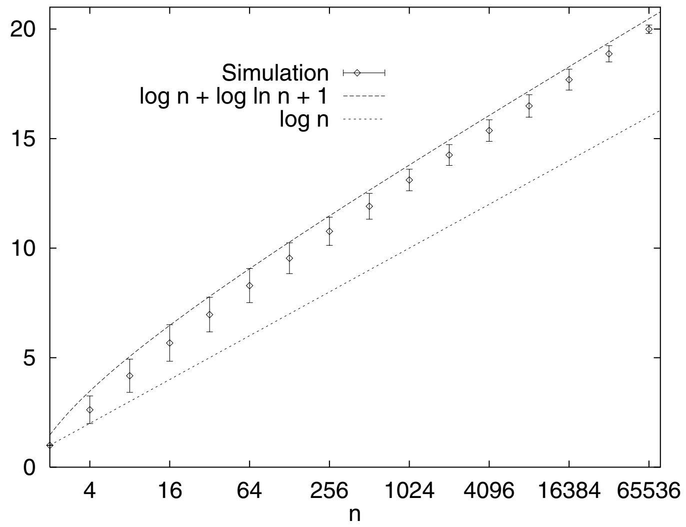

Fig. 5.2. Number of random polling steps to get all PEs busy: Hypothesized upper bound, lower bound and measured averages with standard deviation

at most double in each step. An analysis for a more general model yields an $E[T] \in \mathcal{O}(\log n)$ upper bound [5.34].

We will now argue that there is a much tighter upper bound of $E[T] \leq \log n + \log \ln n + 1$. We start with a theoretical analysis and get stuck half way. We then make a simplifying assumption (independence) that allows us to complete the analysis. The hypothesis generated in this way is then validated experimentally.

Define the 0/1-random variable X_{ik} to be 1 iff PE i is busy at the beginning of step k. For fixed k, these variables are identically distributed and $P[X_{i0} = 1] = 1 - 1/n$. Let $U_k = \sum_{i<n} X_{ik}$. We have

$$E(U_k) = E(\sum_{i<n} X_{ik}) = \sum_{i<n} P[X_{ik} = 1] = nP[X_{ik} = 1].$$

Since the X_{ik} are not independent even for fixed k, we are stuck with this line of reasoning. However, if we (falsely) assume independence, we get

$$P[X_{i,k+1} = 0] = P[X_{ik} = 0] \sum_{j \neq i} \frac{1}{n-1} P[X_{jk} = 0] = P[X_{ik} = 0]^2,$$

and, by induction,

$$P[X_{ik} = 0] = (1 - 1/n)^{2^k} \leq e^{-2^k/n}.$$

Therefore, $E(U_k) \geq n(1 - e^{-2^k/n})$ and for $k = \log n + \log \ln n$, $E(U_k) \geq n - 1$. One more step must get the last PE busy.

We have tested the hypothesis by simulating the process 1000 times for $n = 2^j$ and $j \in \{1, \ldots, 16\}$. Fig. 5.2 shows the results. On the other hand, the measurements do exceed $\log n + \log \ln n$. We conjecture that our results can be verified using a calculation which does not need the independence assumption.

5.3.3 Shellsort

Shellsort [5.39] is a classical sorting algorithm which has been widely studied. Given an increasing integer sequence of offsets h_i with $h_0 = 1$, the following pseudo-code describes Shellsort.

```
for each offset $h_k$ in decreasing order do
    for $j := h_k$ to $n$ step $h_k$ do
        $x :=$ data[$j$]
        $i := j - h_k$
        while $i \geq 0 \wedge x <$data[$i$] do
            data[$i + h_k$] := data[i]
            $i := i - h_k$
        od
        data[$i + h_k$] := x
```

Despite its long history, Shellsort still poses several open problems. For example, let $T(n)$ denote the average number of key comparisons performed by Shellsort for n inputs. It is unknown whether there is an offset sequence which yields a sorting algorithm with $T(n) \in \mathcal{O}(n \log n)$ or even one with $T(n) \in o(n \log^2 n)$ [5.38, 5.18]. It is known that any algorithm with $T(n) = \mathcal{O}(n \log n)$ must use $\Theta(\log n)$ offsets [5.18]. Previous experiments with many carefully constructed offset sequences led to the conjecture that no sequence yields $T(n)$ close to $\mathcal{O}(n \log n)$ [5.45].

Motivated by the successful use of randomness for sorting networks [5.22, Section 3.5.4] where no comparably good deterministic alternatives are known, we asked ourselves whether *random* offsets might work well for Shellsort. For our experiments we used offsets which are the product of random numbers. The situation now is more difficult than in Section 5.3.2 where the theory gave us a very accurate hypothesis. Now we have little information about the dependence of the performance on n. Still, we should put the little things we do know into the measurements. First, by counting comparisons we can avoid the pitfalls of measuring execution time directly. Furthermore, we can divide these counts by the lower bound $\log(n!) \approx n \log n - n/\ln(2)$ for comparison based sorting algorithms. The difficult part is to find an adequate model for the resulting quotient plotted in Fig. 5.3. According to the conjecture in [5.45] the quotient should follow a power law. In a semilogarithmic plot this should be an exponentially growing curve. So this conjecture is not a good model at least for realistic n (also remember that Shellsort is usually

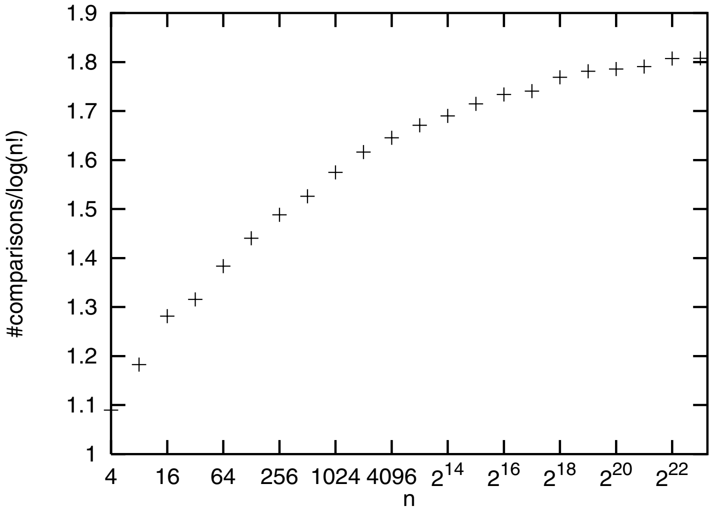

Fig. 5.3. Ratio of the average number of key comparisons of random offset Shellsort compared to the information theoretic lower bound $\log(n!)$. We used $h_i := \lfloor h_{i-1} \cdot f_i + 1 \rfloor$ where f_i is a random factor from the interval $[0, 4]$. Averages are based on 1000 repetitions for $n \leq 2^{13}$ and 100 repetitions for larger inputs

not used for large inputs). A sorting time of $\mathcal{O}(n \log^a n)$ for any constant $a > 1$ would result in a curve converging to a straight line in Fig. 5.3. Indeed, the curve gets flatter and flatter and its inclination might even converge to zero.

We might be tempted to conjecture that $T(n) = \mathcal{O}(n \log^{1+o(1)} n)$. But we must be careful here, because assertions like "$T(n) = O(f(n))$" or "the inclination of $g(n)$ converges to zero" are not experimentally falsifiable.

5.3.4 Sharpening a Theory: Randomized Balanced Allocation

Consider the following load balancing algorithm known as *random allocation*: m jobs are independently assigned to n processing elements (PEs) by choosing a target PE uniformly at random. Using Chernoff bounds, it can be seen that the maximum number of jobs assigned to any PE is

$$l_{\max} = m/n + \mathcal{O}(\sqrt{(m/n) \log n} + \log n)$$

with high probability (*whp*). For $m = n$,

$$l_{\max} = \Theta(\log(n)/\log\log n)$$

whp can be proven.

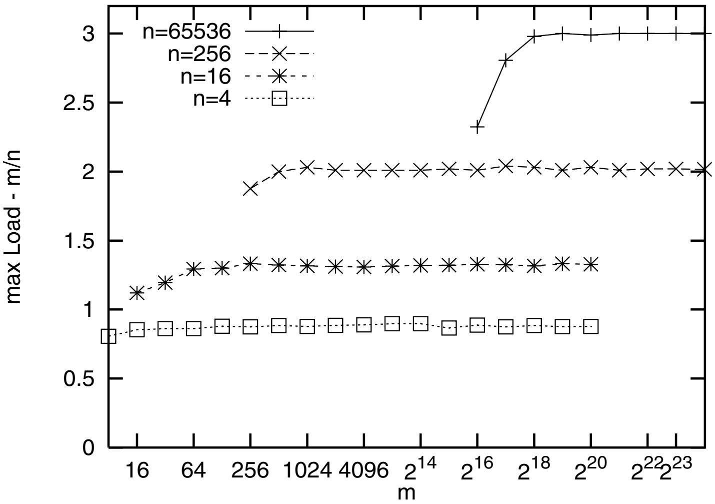

Fig. 5.4. Excess load for randomized balanced allocation as a function of n for different n. The experiments have been repeated at least sufficiently often to reduce the *standard error* $\sigma/\sqrt{\text{repetitions}}$ [5.32] below one percent of the average excess load. In order to minimize artifacts of the random number generator, we have used a generator with good reputation and very long period $(2^{19937}-1)$[5.24]. In addition, we have repeated some experiments with the Unix generator `srand48` leading to almost identical results

Now consider the slightly more adaptive approach called *balanced random allocation*. Jobs are considered one after the other. Two random possible target PEs are chosen for each job and the job is allocated on the PE with lower load. Azar et al. [5.2] have shown that

$$l_{\max} = \mathcal{O}(m/n) + (1 + o(1)) \log \ln n$$

whp for $m = n$. Interestingly, this bound shows that balanced random allocation is exponentially better than plain random allocation. However, for large m their methods of analysis yield even weaker bounds than that for plain random allocation. Fig. 5.4 shows that a simple experiment predicts that $l_{\max} - m/n$ cannot depend much on m. Recently[6] Berenbrink et al. [5.8] have published a proof (using quite nontrivial arguments) that indeed,

$$l_{\max} = m/n + (1 + o(1)) \log \ln n.$$

Our experiments were done before the theoretical solution. For other examples, we could have picked one of the other open problems in the area of balls into bins games. For example, Vöcking [5.43] recently proved that an

[6] After our experiments were done.

asymmetric placement rule for breaking ties can significantly reduce $l_{\max}$ for $m = n$ but nobody seems to know how to generalize this result for general m.

5.4 Empirical Curve Bounding Rules

We now develop several heuristic rules for finding asymptotic trends in data sets. To emphasize the general applicability of these techniques of data analysis, and to achieve some notational compatibility with related works in data analysis, we use the symbol x rather than n to refer to the parameter that is controlled during experimentation.

We begin with some notation and a precise specification of the problem. The cost of algorithm A is described by an unknown exact function $f(x)$, where x may denote problem size. An experiment produces a pair of vectors X, Y such that $Y[i] = F(X[i])$; in cases with randomized inputs and/or randomized algorithms, the experiment produces X, Y such that $E(Y[i]) = f(X[i])$ (that is, f is a function describing the average behavior of the algorithm). By convention, the vector X is assumed to contain k distinct nonnegative values arranged in increasing order.

The complexity class $\mathcal{O}(g(x))$ denotes a set of functions: we have $f(x) \in \mathcal{O}(g(x))$ if there exist positive constants c_u, x_u such that $0 \leq f(x) \leq c_u g(x)$ for all $x \geq x_u$. Similarly, $f(x)$ is in the set $\Omega(g(x))$ if there exist positive constants c_l, x_l such that $0 \leq c_l g(x) \leq f(x)$ for all $x \geq x_l$.

By convention, a complexity class is always labeled by the "simplest" member of the set; thus while $\mathcal{O}(3x^2 + 4x)$ is technically correct, we would use $\mathcal{O}(x^2)$ to denote this class. Throughout, $g(x)$ and $\bar{g}(x)$ are assumed to be simple functions labeling complexity classes, while $f(x)$ and $\bar{f}(x)$ may be arbitrary functions. The bar notation denotes functions that are estimates, and functions without bars denote (typically unknown) target functions.

Each heuristic rule takes X, Y, and reports a class estimator $\bar{g}(x)$ together with a bound type, either *upper*, *lower*, or *close*. *Upper* signifies a claim that $f(x) \in O(\bar{g}(x))$, and *lower* signifies a claim that $f(x) \in \Omega(\bar{g}(x))$. A rule will report a bound of *close* when the data is "too close to call" with respect to the upper/lower bound criteria being used. Usually this occurs when the data is indeed very close to the estimate, but in some cases a *close* result is returned because of some unexpected property of the data set.

An upper bound estimate $\mathcal{O}(\bar{g}(x))$ is *correct* if in fact $f(x) \in \mathcal{O}(g(x))$. A correct upper bound is *exact* if it labels the smallest correct class that holds the target function. Analogous definitions hold for lower bound estimates. Some heuristics iteratively generate internal *guess functions* $\bar{f}(x)$ stopping when come criteron is met and then reporting the corresponding estimate $\bar{g}(x)$ obtained from the leading term of $\bar{f}(x)$.

We consider the five strategies outlined below.

- The *Guess-Ratio* (GR) rule "guesses" a function $\bar{f}(x)$ and evaluates the guess according to the apparent convergence of the ratios $Y/\bar{f}(X)$.

- The *Guess-Difference* (GD) rule also guesses a function $\bar{f}(x)$, but evaluates the differences $\bar{f}(X) - Y$ rather than ratios.
- The *Power* (PW) rule combines log-log transformation of X and Y, linear regression on the transformed data, and residuals analysis. Two variations PW3 and PWD are introduced that improve this method for curve-bounding problems.
- The *Box Cox* (BC) rule combines a parametric transformation of Y values with linear regression and residuals analysis.
- The *Difference* (DF) rule generalizes Newton's divided difference method for polynomial interpolation. The generalization ensures that the method is defined and terminates for any data set.

Oracle Functions. In general, the rules can be viewed as interactive tools or as offline algorithms. To accommodate both views, we describe the algorithms in terms of a small set of *oracle functions* which decide, for example, whether "residuals are concave upwards." When the rules are used interactively, a human provides the oracle values; when the rules are offline, simple computations are used for each oracle function.

Trend($\boldsymbol{X, Y, c_r}$). Returns a value indicating whether Y appears to be *increasing* with X, *decreasing*, or *neither.* Our implementation compares the correlation coefficient r, computed on X and Y, to a cutoff parameter c_r which is 0.1 by default.

Concavity ($\boldsymbol{X, Y, s}$). This function performs a linear regression on X and Y, smooths the residuals, and examines the signs of the smoothed residuals. It returns "concave upward" if signs obey the regular expression $(+)^+(-)^+(+)^+$ (at least one plus, followed by at least one minus, followed by at least one plus); it returns "concave downward" if they obey $(-)^+(+)^+(-)^+$; and otherwise the function returns "neither." The parameter s can be used to adjust the smoothing operation; the default low setting produces "less smooth" residuals and more frequent "neither" results.

DownUp($\boldsymbol{X, Y, s}$). The DownUp oracle examines smoothed Y values to determine whether Y appears to be first decreasing and then increasing within its range. If successive differences in smoothed Y values obey the regular expression $(-)^+(+)^+$, the function returns `True`; otherwise it returns `False`. The default low setting of parameter s (identical in purpose to the one for Concavity) produces less smooth values and more frequent `False` results.

NextCoef(*f*, *direction*, *cstep*) and NextOrder(*f*, *direction*, *estep*). Rules that iterate over several guesses require an oracle to supply the next guess. Our implementation constructs functions $f(x) = ax^b$ for positive rationals a and b. *NextCoef* changes a according to *direction* (up or down) and the *cstep* size. If a decrement of size *cstep* would give a negative coefficient, then *cstep* is reset to *cstep*/10 before decrementing. $NextOrder$ changes the exponent b according to the *estep* size. In our tests the default *estep* is .001 for all but one rule, and the initial *cstep* value is .01.

The remainder of this section presents a "justification" for each rule in the form of a family of functions for which the rule is guaranteed to produce correct results.

5.4.1 Guess Ratio

To justify the Guess Ratio (GR) rule, let the set F_{GR} contain functions of the form $f(x) = a_1 x^{b_1} + a_2 x^{b_2} + \cdots + a_t x^{b_t}$, with rationals a_i positive, and rationals b_i such that $b_1 > 0$, $b_i \geq 0$, and $b_i > b_{i+1}$. Let the guess function be of the form $\bar{f}(x) = x^b$. Then the ratio $f(x)/\bar{f}(x)$ has the following properties: (1) When $f_1(x) \in O(\bar{f}(x))$, the ratio decreases to a nonnegative constant as x increases; (2) When $f_1(x) \notin O(\bar{f}(x))$ the ratio eventually increases and has a unique minimum point at some location x_r. If $x_r > 0$, then the ratio shows an initial decrease followed by an eventual increase. These properties are established by an application of Descartes' Rule of Signs [5.44] which (when extended from polynomials to functions in F_{GR} having rational exponents and coefficients) bounds the number of sign changes in the derivative of the ratio.

The Guess Ratio rule exploits this property by guessing a function $\bar{f}(x)$ and examining the ratio obtained for the finite sample X, Y. If a plot of X vs $Y/\bar{f}(X)$ shows an eventual increasing trend (perhaps with an initial decrease at low X values), then case (2) must hold. If only a decrease is observed in the plotted values, then cases (1) and (2) cannot be distinguished.

The Guess Ratio rule begins with a constant guess function $\bar{f}(x) = x^0$, and increments the exponent b using the NextOrder oracle, iterating until the ratios $Y/\bar{f}(X)$ do not appear to eventually increase. The Trend oracle is used to determine whether the ratios increase. The largest guess $\bar{f}'(x)$ for which an eventual increase is observed is reported as a "greatest lower bound" on the target $f(x)$: thus this rule always generates a *lower* claim that $f(x) = \Omega(\bar{g}_l(x))$, using the estimate $\bar{g}_l(x) = \bar{f}'(x)$.

When $f(x) \in F_{GR}$ and $k \geq 2$, the correctness of GR can be guaranteed simply by defining "eventual increase" as $Y[k-1] < Y[k]$ (recall that k is the size of X). However our implementation uses the Trend oracle (which calculates the correlation coefficient) for this test because of possible random noise in Y. Thus for any data set (X, Y) and for our Trend oracle, the rule must eventually terminate, but cannot be guaranteed correct.

5.4.2 Guess Difference

The Guess Difference (GD) rule also iterates over several guess functions $\bar{f}(x)$, evaluating differences $\bar{f}(X) - Y$ rather than ratios. It produces an upper rather than a lower bound estimate.

This rule is guaranteed correct for the set F_{GD} which contains functions $f(x) = cx^d + e$ where c, d and e are positive rationals, by the following argument. Let the guess function have the form $\bar{f}(x) = ax^b$, and consider the *difference curve* $\bar{f}(x) - f(x)$. When $\bar{f}(x) \notin O(f(x))$, this curve must eventually increase (when x is "large enough"), and it must have a unique minimum at some location x_d. Also, note that x_d is inversely related to the coefficient a in the guess: for large a the difference curve increases everywhere ($x_d = 0$), but for small a there might be an initial decrease at small x. In the latter case we say the curve has the *DownUp* property.

The GD rule starts with an upper bound guess $\bar{f}(x) = ax^b$ and searches for a difference curve having the DownUp property by adjusting the coefficient a. If a DownUp curve is found, the rule concludes that $\bar{f}(x)$ overestimates the order of $f(x)$, so it decrements the exponent b and tries adjusting a again. The lowest b for which the rule finds a DownUp curve is reported as a "least upper bound" found. Thus if the rule stops at $\bar{f}'(x) = a'x^{b'}$, it reports an upper bound $f(x) = O(\bar{g}_u(x))$ with $\bar{g}_u(x) = x^{b'}$.

Using an analysis similar to that for GR, we can show that when $f(x) \in F_{GD}$ and X is fixed and when $k \geq 4$, then there exists an a such that $\bar{f}(X) - f(Y)$ will have the DownUp property. If the rule is able to find the a that produces a DownUp curve in its finite sample, then the upper bound it returns must be correct. In our implementation, if the rule is unable to find an initial DownUp curve within preset limits on iteration, the rule stops and reports the original guess provided by the user.

Note that Guess Difference rule cannot be guaranteed correct for functions from F_{GR} (defined for the Guess Ratio rule), because these functions may have several non-constant terms. If t is the number of terms in $f(x)$, and if $\bar{f}(x)$ over-estimates the order of $f(x)$, then the difference curve $\bar{f}(x) - f(x)$ can have at most $t - 1$ local minimal points (down-up-down-up-down-up) before its eventual increase. A DownUp curve in the plot for the finite sample may only be some initial fluctuation at small x, and it is not necessarily the case that $\bar{f}(x)$ overestimates $f(x)$.

5.4.3 The Power Rule

Power Rule (PW) modifies a standard data analysis technique for fitting curves to data. Suppose that the set F_P contains functions $f(x) = cx^d$ for positive rationals c and d. Let $y = f(x)$. Applying the logarithmic transformation $x' = \ln(x)$ and $y' = \ln(y)$, we obtain $y' = dx' + c$. Now y' is linear in x', and the slope obtained by a linear regression fit of x' to y' is equal to d, the exponent in the original function.

The Power Rule applies this log-log transformation to the data sets X and Y and then reports d, the slope of a linear regression fit on the transformed data. Since we are interested in bounds rather than fits, the Concavity oracle is applied to residuals from the linear regression fit. If the residuals appear to be concave upward, then the rule concludes that the data is growing faster than the fit, and returns a "lower" bound claim. If the residuals are concave downwards, the the rule returns "upper." If the residuals do not meet the convexity criteria for these two claims, the oracle returns “neither” and the Power Rule returns “close.”

If $Y = f(X)$ and $f(X) \in F_P$ then the Power rule finds the exponent d exactly. If Y is a random variate such that $Y = f(X) \cdot \epsilon$ and the random noise component ϵ obeys standard assumptions of independence and lognormality, then confidence intervals on the estimate of d can be derived by standard techniques (see [5.33] for details).

High-End Power Rule (PW3). When $f(x)$ contains low-order terms (such as $ax^b + e$), the log-log transformed points do not lie on a straight line. In this case, a linear regression using only the transformed points at the j highest X values might give a better asymptotic bound than one using all k points. The PW3 variation on the Power Rule applies the Power rule to the three highest data points corresponding to $X[k-2]$, $X[k-1]$, and $X[k]$.

Power Rule with Differences (PWD). The *differencing* variation on the Power rule attempts to straighten out plots under log-log transformation by removing constant terms. This variation can be applied when the X values are chosen such that $X[i] = \Delta \cdot X[i-1]$ for a positive constant Δ (for example, if $\Delta = 2$ then the X values are obtained by successive doubling. This variation applies the Power rule to *successive differences* in adjacent Y values, rather than to Y values alone.

To justify this rule, suppose F_{PWD} contains $f(x) = cx^d + e$ where c, d and e are positive rationals, and let $Y = f(X)$. Set $Y'[i] = Y[i+1] - Y[i]$ and $X'[1..k-1] = X[1..k-1]$.

Then we have

$$\begin{aligned} Y'[i] &= f(X[i+1]) - f(X[i]) \\ &= cX[i+1]^d + e - cX[i]^d - e \\ &= c(\Delta X[i])^d - cX[i]^d \\ &= c(\Delta)^d X[i]^d - cX[i]^d \\ &= X[i]^d (c\Delta^d - c) \end{aligned}$$

Now $Y' = c'X'^d$: that is, the exponent is the same as in the original, there is a new coefficient, and the constant e has been removed. The Power rule is then applied to Y' and X' in order to bound the exponent d. If $f(x) \in F_{\mathrm{PWD}}$, $Y = f(X)$ and $k > 4$, then the PWD rule is guaranteed to find d exactly.

Note that it is straightforward to extend this result to show that taking differences on Y twice will remove a logarithmic term.

5.4.4 The BoxCox Rule

To generalize the power rule, a standard approach in curve-fitting is to find transformations on Y or on X, or both, that produce a straight line in the transformed scale, and then to invert the transformation to obtain an estimate of the original curve. For example, if $Y = X^2$, then a plot of X vs $\sqrt{Y}$ would produce a straight line, as would a plot of X^2 vs Y. One difficulty with the general approach is that it can be hard to find a good statistic to compare the quality of different transformations because the transformation changes the scale of the data points.

The Box-Cox ([5.1, 5.10]) curve-fitting method applies a transformation on Y that is parameterized by λ, and defines a "straightness" statistic that permits comparisons of transformations across different parameter levels. The transformation is as follows:

$$Y^{(\lambda)} = \begin{cases} \frac{Y^{\lambda}-1}{\lambda \bar{Y}^{\lambda-1}} & \text{if } \lambda \neq 0 \\ \bar{Y}\ln(Y) & \text{if } \lambda = 0 \end{cases}$$

where $\bar{Y}$ is the geometric mean of Y, equal to exp(mean(ln (Y))). The "straightest" transformation in this family minimizes the Residual Sum of Squares (RSS) statistic which is calculated from X and Y^{λ}.

Our BC rule iterates over a range of guesses $\bar{f}(x) = x^b$ generated by the NextOrder oracle (with the range specified by the user). The rule evaluates $Y^{(\lambda)}$ with $\lambda = 1/b$ at each iteration, and the b' that produces the minimum RSS statistic is returned as the complexity class estimate $\bar{g}(x) = x^{b'}$. The Concavity oracle is then applied to residuals from the linear regression fit under the transformation, to determine the type of bound claimed (upper, lower, close).

When $f(x) = F_{PW}$, $Y = f(X)$, $k > 2$, and when NextGuess oracle includes $f(x)$, this rule is guaranteed to finds the function exactly. With standard normality assumptions about an additive random error term, it is possible to calculate confidence intervals for the estimate on exponent b: see [5.1] or [5.10] for details.

5.4.5 The Difference Rule

The **Difference** heuristic extends Newton's divided difference method for polynomial interpolation (see [5.40] for an introduction) This method calculates $Y^1 = \text{diff}(Y)/\text{diff}(X)$, where diff$(Y)$ denotes the differences between successive values in Y (and is therefore of length $k-1$), and $X^1 = X[1 \ldots k-1]$. If after d such calculations the resulting Y^d values are all equal, then we can conclude that $f(x)$ is a polynomial of degree d.

The extension used here applies when when Y contains random noise and nonpolynomial terms. The method iterates numerical differentiation on X and Y until the data "appears to be non-increasing," according to the Trend oracle. The number of iterations d required to obtain this condition provides an upper bound guess $\bar{g}(x) = x^d$. If $f(x)$ is a positive increasing polynomial of degree d, and if $k > d$, and $Y = f(X)$, then this method is guaranteed correct. Much is known about numerical robustness, best choice of design points, and (non)convergence when $k \leq d$.

5.4.6 Two Negative Results

A basic requirement is that a curve-bounding heuristic be internally consistent. For example, it should not be possible to reach the contradictory conclusions "Y is growing faster than X^2" and "Y is growing more slowly than X^2" on the same data set, merely by applying variations on the heuristic rule. Surprisingly, two plausible approaches included in our initial study turned out to have exactly this failure.

The first, perhaps the most obvious approach to the problem of bounding empirical curves, is to use general (nonlinear) regression to fit a multi-term function $\bar{f}(x)$ to the data set. The leading term of $\bar{f}(x)$ would provide the complexity class estimate, and the curvature of the residuals from regression analysis would provide the upper/lower bound claim.

Several general regression methods are known in the literature. These methods can be viewed as simple types of heuristic search, where a "step" from the current model $\bar{f}_i(x)$ to the next involves the addition or removal (or both) of an additive term, and the objective function (to be minimized) is a goodness-of-fit statistic such as the residual sum of squares (RSS).

In preliminary tests we found the RSS to be woefully inadequate for curve-bounding problems, in the sense that the statistic was quite oblivious to how close the leading term of $\bar{f}(x)$ was to that of true function $f(x)$. Nor were we able to discover a substitute statistic that could distinguish between a variety of guesses having different leading terms. As a result, when experimenting with this general regression method there was no sense of "convergence" towards a correct answer, and our "final" results were primarily artifacts of the stepping rule applied during the heuristic search. It seems an interesting problem for future research to determine whether general regression can be adapted to the curve-bounding problem.

The second approach is based on Tukey's [5.42] "ladder of transformation" technique, by which the X or Y values (or both), are transformed according to functions along the scale

$$\ldots x^{-1}, x^{-1/2}, \log(x), x^{1/2}, x^1, x^2 \ldots,$$

until the transformed data appears as a straight line. The best transformation on X, or inverse of the best transformation on Y, produces the asymptotic bound $g(x)$.

We implemented two versions of this approach, one which systematically applies transformations to Y, and one which transforms X. The straightness of each transformation was assed by the RSS statistic with respect to a linear regression on the transformed data; the upper/lower bound was determined by the Concavity oracle (or by visual inspection).

Our preliminary investigation showed that this approach frequently gives contradictory results depending on whether the transformation is applied to Y or X. The problem is that the correct transformation for the leading term of $f(X)$ can be difficult to find when a large (or even moderately-sized) second-order term is present, and the importance of the second-order term varies considerably depending on whether Y or X is transformed. In our early tests these two rules frequently gave contradictory bound claims, such as both $\Omega(x^{2.2})$ and $O(x^{1.8})$.

As a result of these early failures, these two approaches were abandoned prior to the developement of the designed experiments, and are not considered further here. Note that the BoxCox curve-fitting method can be seen as a formalization of Tukey's transformation ladder (restricted to Y transformations), and some of the difficulties that we observe for BC may have similar foundation.

5.5 Experimental Results

The rules have been implemented in the S language [5.5], which is supported by the Splus software package designed for statistical and graphical computations. The main set of experiments were carried out on a Sun SPARCstation ELC, using functions running within Splus; some supporting experiments were conducted using the Lisp-based CLASP statistical/graphics package. Timing statistics would be very misleading in this context and are not reported in detail.

Roughly, however, the three Power rules required a few microseconds, and two of the iterative rules (Guess Ratio, BoxCox) usually took no more than a few seconds per trial (each trial corresponding to around 20-50 iterations of guess function generation). The Guess Difference rule iterates over two parameters (e and c), and was significantly slower than the other iterative rules; therefore a coarser *estep* value in the NextOrder oracle (0.01 instead of 0.001) was adopted to produce comparable wall clock times for this heuristic.

5.5.1 Parameterized Functions

The first experiment uses constructed functions $f(x) = ax^b + cx^d$, with $b > d$, with a positive, and with no randomization. To illustrate the sensitivity of the rules to low-order terms that may dominate at small x, this experiment varies the relative magnitudes of a to c and of b to d. Here the input vector X is small, containing powers of two ranging between 16 and 128.

Note that all of the successful examples in Section 5.3 use much larger problem sizes than are presented here. At any given maximum problem size, any curve-bounding rule will have no difficulty detecting asymptotic trends on "easy" functions having $b >> d$ and $a > c$. Similarly, any curve-bounding rule will fail on "hard" functions with $b \approx d$ and/or $a < c$. The goal of this experiment is to "stress" the rules and find the limits of successful applicability by using and difficult test functions for the given problem sizes.

To that end, the parameter values used in this experiment were selected (from the enormous space of possible combinations) after several weeks of informal testing in order to locate the boundaries between easy and hard functions and problem sizes vfor these rules. Each parameter is allowed to vary within a range that causes some rules to move from success to failure. Curve-bounding rule that fail here will also tend to fail on harder functions and/or smaller problem sizes.

The exponent b takes three values $[0.2, 0.8, 1.2]$. Our initial exploration suggested that functions with exponents above two are generally quite easy to bound. Also, many open problems of interest to algorithm analyzers involve functions with exponents below two (see Section 5.3). Non-integer exponents were chosen here to avoid "lucky guesses" in our parallel tests using human oracles (since people tend to start guessing with integers). Similarly, the fixed coefficient $a = 3$ was chosen because people tend to guess one and ten first.

For each b value the second exponent d is set to $[0, 0.2, b - 0.2]$, subject to the restriction that $d < b$. The zero provides a constant second term, the 0.2 gives a second term which is "small" compared to b, and the third exponent is "near" b. For $d = 0$, the constant c is set to 10^4, and when $d > 0$ the coefficient c takes values from $[1, -1, 10^4]$ (small, negative, and large).

Figure 5.5 presents raw results from an experiment using all combinations of b, c, and d described above, plus three extra tests identified as functions 1, 2, and 11 (to illustrate some observations made below). In function 11 the constant 10^6 is added to ensure that all y values are positive, because some rules cannot handle negative y values.

The table shows the leading exponents that were returned by the rules. On functions 1 through 3, the correct exponent is 0.2; on functions 4 through 11 it is 0.8; and on functions 12 through 17 the exponent is 1.2. The notations (**l, u**) indicate the type of bound reported by the rule, either **lower** or **upper**. These numerical results have been rounded to two decimal places – lower bounds were rounded down, and upper bounds were rounded up. An underline marks a bound that is incorrect. A * marks a case where the heuristic failed to return an answer, usually because of lack of convergence.

Many intriguing observations arise.

The Guess Ratio (GR) rule, possibly the most widely-used curve-bounding technique in the folklore, performs surprisingly poorly. While it is frequently correct and close, it never dominates the three Power rules, and it always fails on functions having negative second terms (6, 9 and 16), even when the magnitude of the second term is small. This rule begins with a low guess function and iterates, increasing guesses, until the Trend oracle reports the ratio is "not increasing." With a negative second order term, the true function approaches its asymptote from above, which fools the oracle. A more sophisticated termination test might reduce this problem; but on the other hand we note in Section 5.6.1 that using a human to provide the termination test gives worse results in general.

Note that GR tends to "track" large positive second terms, producing correct, but less tight bounds, when the second term dominates the data.

	Function	GR	GD	PW	PW3	PWD	BC	DF
1	$3x^{0.2} + 1$	0.17l	0.24u	0.17l	0.17l	0.20u	0.17l	1u
2	$3x^{0.2} + 10^2$	0.01l	0.24u	0.01l	0.01l	0.20l	0.01l	1u
3	$3x^{0.2} + 10^4$	0.00l	0.24u	0.00l	0.00l	0.20l	*	1u
4	$3x^{0.8} + 10^4$	0.00l	*	0.00l	0.00l	0.80l	*	1u
5	$3x^{0.8} + x^{0.2}$	0.77l	*	0.77l	0.78l	0.79l	0.79l	1u
6	$3x^{0.8} - x^{0.2}$	$\underline{\text{0.82l}}$	*	0.83u	0.82u	0.81u	0.81u	1u
7	$3x^{0.8} + 10^4x^{0.2}$	0.20l	*	0.20l	0.20l	0.20l	0.20l	1u
8	$3x^{0.8} + x^{0.6}$	0.77l	*	0.77l	0.77l	0.77l	0.77l	1u
9	$3x^{0.8} - x^{0.6}$	$\underline{\text{0.83l}}$	0.88u	0.85u	0.84u	0.83u	$\underline{\text{0.81l}}$	1u
10	$3x^{0.8} + 10^4x^{0.6}$	0.60l	*	0.60l	0.60l	0.60l	0.60l	1u
11	$3x^{0.8} - 10^4x^{0.6}$ $+10^6$	$\underline{\text{-0.01l}}$	*	$\underline{\text{-0.06u}}$	$\underline{\text{-0.09u}}$	*	*	$\underline{\text{0u}}$
12	$3x^{1.2} + 10^4$	0.03l	1.3u	0.03l	0.05l	1.2l	*	2u
13	$3x^{1.2} + x^{0.2}$	1.18l	1.22u	1.18l	1.19l	1.19l	1.2u	2u
14	$3x^{1.2} + 10^4x^{0.2}$	0.21l	*	0.21l	0.22l	0.26l	0.23l	$\underline{\text{1u}}$
15	$3x^{1.2} + x^1$	1.17l	1.3u	1.17l	1.17l	1.17l	$\underline{\text{1.18u}}$	2u
16	$3x^{1.2} - x^1$	$\underline{\text{1.23l}}$	1.27u	1.25u	1.24u	1.24u	$\underline{\text{1.22l}}$	2u
17	$3x^{1.2} + 10^4x^1$	1.00l	*	1.00l	1.00l	1.00l	1.0l	$\underline{\text{1u}}$

Fig. 5.5. Parameterized nonrandom functions. The numbers indicate the leading exponents returned by the rules. The notations **l, u**, indicate whether a **lower** or **upper** bound was returned. These numbers have been rounded to two decimal places – lower bounds were rounded down and upper bounds were rounded up. An underline marks a bound that is incorrect. The starred entries (*) mark cases where the rule failed to return a result

On functions 1, 2, and 3, for example, the bound actually decreases as the constant term becomes more important. Similarly, functions 3, 4, and 12 have the same constant second term, and in these three cases the bound returned by GR fails to follow the leading exponent. Finally, notice that performance deteriorates with respect to the function pairs (5 and 7), (13 and 14), and (15 and 17), which differ only in the coefficient on the second term.

The Guess Difference (GD) column contains several starred entries that mark cases where the rule failed to find an initial DownUp curve. In cases it returned the user-supplied starting guess, which was either $1x^1$ (functions 1 through 11) or $1x^2$ (functions 12 through 17). It appears that the performance of GD is quite sensitive to the choice of initial guess and step sizes: further exploration here suggests that the failures in functions 4 through 11, for example, are caused by an initial guess $1x^1$ that is too close to the true function $3x^{0.8}$. A higher initial guesses does allow the rule to get started and to find a tighter bound. Function 14 represents a different kind of failure – in this trial the GD routine was canceled after about 60 minutes of processing, at which time it was working on a guess of $1502.2x^{0.56}$, approaching the second order term from above.

However, when GD is able to get started, its estimates are surprisingly tight – much better than other rules in some cases. GD shows less sensitivity to large second terms than does GR, but the rule is not impervious to second-order interference, as function 14 indicates.

The Power rules are close to one another, and also surprisingly close to GR in performance. However unlike GR, the three Power rules remain correct on functions 6, 9 and 16 (with negative second terms) by switching from "lower" to "upper" bound claims. Both PW3 and PWD give slightly tighter bounds than PW. Not only does PWD successfully eliminate the constant terms, producing exact bounds in functions 1–4 and 12, but it is slightly better than PW and PW3 even when the second term is not constant.

The BC rule returns bounds similar to those for GR and the Power rules. This rule provides very competitive bounds when it works, but it fails to converge on functions 3, 4, 11, and 12. These functions have a very large constant as a second term: it turns out that the failure of BC here is an intrinsic property of the λ transformation. That is, if the data is nearly constant, then the "straightest" transformation, having minimum RSS value, is obtained by the transformation $Y^{1/b}$ with $b = 0$. The rule iterates towards ever-smaller b values until the calculation of $1/b$ produces a numeric error.

Large increasing second terms (functions 7, 10, 14, 17) present no such termination problems for BC, although the rule does tends to track the second term. On functions 9, 15, and 16 the bound is incorrect although the estimate is close to those obtained by other rules. This appears to be due to interactions between the λ transformation and our Concavity function.

As is the case with PWD, the differencing operation performed by the DF rule eliminates the effect of large constant terms. Recall that this rule can only return integer exponents, which are often correct but rarely close to the selected functions. This rule fails on functions 11, 14, and 17.

Function 11 is disasterous for all the rules because the large negative second term causes Y to be decreasing within its range. As a general rule, these rules do not work well on functions that are decreasing or even temporarily decreasing within their range.

Increasing the Largest Problem Size. The obvious remedy to the problem of a dominant second-order term is to use larger problem sizes. The second experiment uses functions identical to those of the previous section, but X takes values at powers of two in the range $8 \ldots 256$ rather than $8 \ldots 128$ thereby doubling the largest problem size.

The results in Figure 5.6 are very similar to those in in the previous chart, suggesting that in general the rules respond very slowly to changes in the largest input values. In particular, doubling the largest problem size has very little effect on the bounds returned by Guess Ratio and the three Power Rules. The observed changes in estimates were generally only in the third or higher decimal places, and incorrect bounds remain incorrect.

The Guess Ratio rule could be made more responsive to changes in problem size if a different Trend oracle were used to provide the stopping condition: instead of calculating the correlation coefficient, an oracle that concen-

	Function	GR	GD	PW	PW3	PWD	BC	DF
1	$3x^{0.2} + 1$	0.17l	0.23u	0.17l	0.17l	0.20u	0.18l	1u
2	$3x^{0.2} + 10^2$	0.01l	0.23u	0.01l	0.01l	0.20l	0.01l	1u
3	$3x^{0.2} + 10^4$	0.00l	0.23u	0.00l	0.00l	0.20l	*	1u
4	$3x^{0.8} + 10^4$	0.00l	0.83u	0.00l	0.01l	0.80l	0.00l	1u
5	$3x^{0.8} + x^{0.2}$	0.77l	0.82u	0.77l	0.78l	0.79l	0.79l	1u
6	$3x^{0.8} - x^{0.2}$	$\underline{\text{0.82l}}$	0.83u	0.83u	0.82u	0.81u	0.81u	1u
7	$3x^{0.8} + 10^4x^{0.2}$	0.20l	*	0.20l	0.20l	0.20l	0.20l	1u
8	$3x^{0.8} + x^{0.6}$	0.77l	0.80u	0.77l	0.77l	0.77l	0.78c	1u
9	$3x^{0.8} - x^{0.6}$	$\underline{\text{0.83l}}$	0.85u	0.84u	0.83u	0.83u	0.82c	1u
10	$3x^{0.8} + 10^4x^{0.6}$	.60l	*	0.60l	0.60l	0.60l	0.60l	1u
11	$3x^{0.8} - 10^4x^{0.6}$ $+10^6$	-0.01l	*	$\underline{\text{-0.07u}}$	$\underline{\text{-0.15u}}$	*	*	$\underline{\text{0u}}$
12	$3x^{1.2} + 10^4$	0.06l	1.22u	0.05l	0.11l	1.20l	*	2u
13	$3x^{1.2} + x^{0.2}$	1.19l	1.22u	1.18l	1.19l	1.19l	1.20u	2u
14	$3x^{1.2} + 10^4x^{0.2}$	0.22l	*	0.21l	0.23l	0.29l	0.25l	$\underline{\text{1u}}$
15	$3x^{1.2} + x^{0.8}$	1.17l	1.20u	1.17l	1.18l	1.18l	$\underline{\text{1.19u}}$	2u
16	$3x^{1.2} - x^{0.8}$	1.22l	1.24u	1.24u	1.23u	1.23u	$\underline{\text{1.21l}}$	2u
17	$3x^{1.2} + 10^4x^{0.8}$	0.80l	*	0.80l	0.80l	0.80l	0.80c	$\underline{\text{1u}}$

Fig. 5.6. Doubling the largest problem size. The numerical values show the leading exponent returned by the rule. The notations **l, u, c**, indicate the type of bound reported by the rule, either **lower**, **upper**, or **close**. These results are rounded to two decimal places: lower bounds are rounded down, upper bounds are rounded up and close bounds are rounded to the nearest decimal. An $\underline{\text{underline}}$ marks a bound that is incorrect. A * marks a rule that failed to return an answer

trates on the high end of the data set might be more successful here. It is surprising that PW3 does not respond much to the change in problem size, because only the highest three data points are checked each time. One would expect the new point to have much greater leverage for this rule.

The greatest improvement is found in the Guess Difference (GD) rule on functions 4 through 9 (excepting 7). In the previous experiment the rule failed to find an initial DownUp curve at all—now the rule is able to find an initial curve, and iterate to find upper bounds within 0.05 of the true exponent. The BC rule also shows some very slight improvement: in two cases the rule produces *close* bound claims where previously the claim had been incorrect.

It is a problem for future research to how best to design rules that respond to significant changes in problem sizes. For now, it remains important in any algorithmic experiment to obtain results using the largest problem sizes possible, especially when the underlying function has low exponents.

Adding Random Noise. The previous two experiments use functions with no random noise in the data. In the third experiment we add a random term to three functions (1, 5, and 13) that were easy for all rules, to learn how rule performance degrades with increased variance. We let $Y = \bar{f}(X) + \epsilon_i$ with $i = 1, 2, 3$. The random variates ϵ_i are drawn independently from a normal distribution with mean 0 and standard deviation set to constants 1

($i = 1$) and 10 ($i = 2$), and to the function means $\bar{f}(X[j])$ ($i = 3$). We ran two independent trials for each i, in order to check for spurious positive and negative results. A table of results appears in Figure 5.7.

Not surprisingly, the quality of results returned by all rules degrades as dramatically as random variation increases. The replication of tests in each category demonstrates that many correct bounds are in fact spurious. Conversely, of course, rule performance improves when variance in the data decrease: This is good news for experimentors because is often possible to reduce variance in experimental data, either by increasing the number of trials or by applying one of several variance reduction techniques known in the literature (see [5.25]). Note that variance is less of a problem when the first term exponent is large enough.

The GR rule responds strangely to random data, returning negative bounds and lower bounds of 2.98 and even 25.7 [sic] on these functions. Not surprisingly, PW3 is frequently wrong – when random variation is present, it seems wise to make use of all the data, rather than just part of it. As

Function	GR	GD	PW	PW3	PWD	BC	DF
$3x^{0.2} + 1$	0.173l	0.23u	0.17l	0.17l	0.2c	0.18l	1u
$3x^{0.2} + 1 + \epsilon_1$	0.12l	*	0.15c	$\underline{\text{-0.00u}}$	$\underline{\text{0.05u}}$	0.90u	1u
$3x^{0.2} + 1 + \epsilon_1$	0.10l	*	0.10c	0.34u	-0.02l	0.40u	1u
$3x^{0.2} + 1 + \epsilon_2$	$\underline{\text{25.7l}}$	0.57u	0.97u	0.67u	-0.5c	*	1u
$3x^{0.2} + 1 + \epsilon_2$	0.90l	*	0.63c	$\underline{\text{0.40l}}$	0.19l	*	2u
$3x^{0.2} + 1 + \epsilon_3$	-0.1l	*	-0.01c	$\underline{\text{-0.55u}}$	0.93l	0.41c	$\underline{\text{0u}}$
$3x^{0.2} + 1 + \epsilon_3$	-0.01l	*	-0.05c	-0.34l	0.03c	1.00c	$\underline{\text{0u}}$
$3x^{0.8} + x^{0.2}$	0.77l	0.82u	0.77l	0.78l	0.79l	0.79l	1u
$3x^{0.8} + x^{0.2} + \epsilon_1$	0.77l	0.83u	0.77l	0.77l	0.80u	0.78c	1u
$3x^{0.8} + x^{0.2} + \epsilon_1$	0.76l	$\underline{\text{0.78u}}$	0.76c	0.81u	0.77l	0.81c	1u
$3x^{0.8} + x^{0.2} + \epsilon_2$	0.71l	*	0.75c	$\underline{\text{0.77u}}$	0.78c	0.69c	1u
$3x^{0.8} + x^{0.2} + \epsilon_2$	0.69l	*	0.68c	0.73l	0.89c	0.81c	1u
$3x^{0.8} + x^{0.2} + \epsilon_3$	$\underline{\text{1.50l}}$	*	1.34c	1.03u	0.91u	*	2u
$3x^{0.8} + x^{0.2} + \epsilon_3$	$\underline{\text{1.08l}}$	*	1.01u	$\underline{\text{-0.35u}}$	1.98u	*	1u
$3x^{1.2} + x^{0.2}$	1.19l	1.22u	1.18l	1.19l	1.19l	1.20u	2u
$3x^{1.2} + x^{0.2} + \epsilon_1$	1.18l	1.22u	1.18l	1.19l	1.21u	1.20c	2u
$3x^{1.2} + x^{0.2} + \epsilon_1$	1.18l	1.22u	1.18l	1.19l	1.19l	1.20c	2u
$3x^{1.2} + x^{0.2} + \epsilon_2$	1.18l	1.22u	1.17l	1.20u†	$\underline{\text{1.19u}}$	1.19c	2u
$3x^{1.2} + x^{0.2} + \epsilon_2$	1.15l	1.30u	1.14l	1.18l	1.22c	1.22c	2u
$3x^{1.2} + x^{0.2} + \epsilon_3$	0.10l	1.99u	$\underline{\text{1.25l}}$	$\underline{\text{2.20l}}$	$\underline{\text{1.83l}}$	*	$\underline{\text{1u}}$
$3x^{1.2} + x^{0.2} + \epsilon_3$	$\underline{\text{2.98l}}$	2.00u	1.58u	$\underline{\text{0.39u}}$	0.94l	2.59u	$\underline{\text{1u}}$

Fig. 5.7. Adding random noise. The numbers show the exponents returned by the rules. The notations **l, u, c**, indicate the type of bound reported by the rule, either **lower**, **upper**, or **close**. These results are shown rounded to two decimal places: lower bounds are rounded down, upper bounds are rounded up, and close bounds are rounded to the nearest decimal. The † marks a case where rounding changed an originally incorrect upper bound (1.194u) to a correct one (1.2u). An $\underline{\text{underline}}$ marks a bound that is incorrect. The starred entries (*) mark cases where the rule failed to return a bound

variance in Y increases, the Power and the BoxCox rules more frequently return claims of *close*. We do not know how to interpret these results to obtain bounds (upper or lower) on function growth; therefore these rules may be less useful for curve-bounding problems when large variance is present.

5.5.2 Algorithmic Data Sets

The experiment in this section applies the rules to eight data sets taken from previous computational experiments by the first author. The data sets were originally developed in the context of experimental research on algorithms, and not for testing curve-bounding heuristics. Thus the performance of the heuristics on these data sets may give more realistic indications of their performance in practice. On the other hand, since these data sets are from research problems, we don't always know the true underlying function $\bar{f}(x)$, and can't always tell when the rules are correct.

The results appear in Figure 1.8. The left column gives the best analytical bounds known for each function. The entries NA for PWD mark cases where this rule was not applied because design points were not in required format (with X increasing by constant multiples).

Data sets 1 and 2 represent the expected costs of Quicksort and Insertion Sort, formulas for which are known exactly (see for example [5.20]). The X values are [10, 20, 30, ..., 1000] for Quicksort, and [10, 20, 30 ..., 500] for Insertion sort. These data sets were generated from the formulas with no random noise. An experimental study of these algorithms would produce random variation in the data, but because these algorithms are extremely efficient it would be possible to make the variace quite small by taking large batches of trials. For Quicksort the asymptotic leading term (i.e. the "correct answer" is $\Theta(x \log x)$; for Insertion sort the leading term is $\Theta(x^2)$.

Sets 3 through 6 are from experiments on heuristics for one-dimensional bin packing [5.6], [5.7]. In these experiments X takes values $[200, 400, 800, \dots,$ $128000]$ (doubling each time). Set 3 shows measurements of *bin count* and Set 4 measures *empty space*, for First Fit Decreasing rule. Sets 5 and 6 show measurements of empty space for the First Fit rule under two different parameter settings. In all four cases, each Y value represents the mean of 25 independent trials. Variance in the four data sets is, respectively, about 0.3x, 40x, 1x ,0.1x (times) the mean. The formulas shown on the left represent the best analytical bounds known for the functions generating these data.

Sets 7 and 8 are from experiments on distances in random complete graphs having weights drawn from a uniform distribution on $(0, 1]$ [5.26]. In both cases X = [200, 400, 600, ..., 1400] and each Y value represents the mean of 50 independent trials. In Set 7 variance is about 2x mean, and in Set 8 variance is a constant near 1000.

Contrary to experience with the constructed functions, the Guess Ratio rule (GR) obtains a correct and tight bound when a negated second term

	Known	GR	GD	PW	PW3	PWD	BC	DF
1	$(x+1)(2H_{x+1}-2)$	$\underline{\text{1.20l}}$	1.24u	1.23u	1.19u	NA	1.18c	2u
2	$(x^2-x)/4$	2.00l	2.03u	3.01u	3.01u	NA	2.00l	2u
3	$x/2+O(1/x^2)$	0.99l	*	0.99l	1.00u†	1.00c	1.20c	2u
4	$\Theta(x^{0.5})$	$\underline{\text{0.52l}}$	*	0.55c	0.58u	0.78c	1.00c	1u
5	$O(x^{2/3}(\log x)^{1/2})$, $\Omega(x^{2/3})$	$\underline{\text{0.68l}}$	0.72u	0.69c	0.69u	0.69c	0.69c	1u
6	$y \leq 0.68x$	0.90l	1.00u	0.89l	0.95l	$\underline{\text{1.26l}}$	0.98c	1u
7	$x-1 \leq y$ $\leq 13.5x \ln x$	$\underline{\text{1.13l}}$	1.18u	1.15u	$\underline{\text{1.12l}}$	NA	1.11c	2u
8	$x \ln x < y < 1.2x^2$	1.30l	1.47u	1.32u	1.20l	NA	1.20c	2u

Fig. 5.8. Data from algorithmic experiments. The numbers give the leading exponents returned by the rules. The notations **l, u, c**, indicate the type of bound reported, either **lower**, **upper**, or **close**. The numbers are rounded to two decimal places: lower bounds are rounded down, upper bounds are rounded up, and close bounds are rounded to the nearest decimal. The † marks a case where rounding changed an incorrect result (0.999u) to a correct one (1.00u). An $\underline{\text{underline}}$ marks a bound which is known to be incorrect, and * marks a case where the rule failed to return an answer. In some cases (NA) the PWD rule was not applied because the X values in the data did not increase by constant factors

is present (Set 2). However in four cases (Sets 1, 4, 5, and 7), GR produces lower bound claims that violate the known bounds.

For Set 1 (and possibly for Sets 5, 7, and 8), the leading term contains a logarithmic factor, which is not generated by our NextOrder function. From additional tests that include logarithmic terms as guess functions, we observe that none of the rules is able to distinguish logarithms from low-order exponents such as $x^{0.2}$ with any degree of reliability. Since logarithms do tend to occur in many algorithmic research problems, it would be useful to develop some techniques that can be applied specifically to this problem.

The Guess Difference rule and the Power Rules rarely violate known bounds on the data sets, although without better analyses it is impossible to tell whether the rules are correct in all cases. Note that BC nearly always returns a "close" report, which is very difficult to evaluate. Interestingly, every incorrect bound produced by these rules is a lower bound.

Data Sets 5 through 8 have gaps between the known lower and upper bounds. In these cases we might hope that the heuristic rules can provide some insight to direct future analytical research: does the upper bound need to be lowered, or does the lower bound need to be raised (or both)?

In Sets 5 and 7, the $(\log x)^{0.5}$ and $c \log x$ gaps are too small to be distinguishable by these rules. In Set 6, however, the rules provide consensus support for a conjecture that the true function $\bar{f}(x)$ is closer to linear $\Theta(x^1)$ than, say, to a square-root function $\Theta(x^{0.5})$. In Set 8 the results are even stronger. Given the above observation that logarithmic terms tend to be indistinguishable from terms near $x^{0.2}$, we have much greather support for a conjecture $\bar{f}(x) = \Theta(x \log x)$ than than $\bar{f}(x) = \Theta(x^2)$ although the true answer may be somewhere in between. (In this case there is external supporting evidence that the lower bound is tight.)

5.6 A Hybrid Iterative Refinement Method

In our informal explorations and designed experiments with little or no random noise in the data, all the rules generally can get within a linear or sometimes $\sqrt{x}$ factor of the exact bound, except when they become "fooled" by very large second-order terms. It is possible to reduce the effect of large second-order terms by taking larger problem sizes, but the rules are surprisingly slow to respond to this type of change. In this section we describe a hybrid rule which appears to be very robust with respect to large second terms.

The hybrid rule incorporates an iterative diagnosis and repair technique that combines the existing heuristics to produce improved guess function modes. The technique is designed to find upper bounds for functions of the form $ax^b + cx^d$ with rational exponents $b > d \geq 0$ and real coefficients $a \ll c$. This method represents a departure from our approach up to now: The earlier methods were intended to be general, but this one is specific to functions with relatively large coefficients on low order terms. This suggests a new role for the methods we have discussed so far: Instead of using them to guess at the order of a function, they can provide diagnostic information about the function (e.g., whether $a \ll c$), and then more specific, purpose-built methods, designed for particular kinds of functions, can estimate parameters.

To illustrate this new approach, we developed a three-step hybrid method for functions of the form $\bar{f}(x) = ax^b + cx^d$;

1. Apply a discrete derivative (the Difference rule) to the datasets, in order to find the integer interval of the exponent b.
2. Refine the guess for the exponent using the Guess Ratio rule. We start with the known upper and lower bound for the exponent, u and l. At each step we consider the model $x^{(u+l)/2}$ by plotting x against $y/x^{(u+l)/2}$. If the plotted points appear to be decreasing, then $(u+l)/2$ is overestimating the exponent, and we replace u by $(u+l)/2$. If the points are increasing, then l will be replaced. The estimates are refined until u and l get within a desired distance ϵ of each other. At this point, if the dataset $y/x^{(u+l)/2}$ has a DownUp feature, then we know that function $\bar{f}$ must have a relatively high coefficient c on a low order term. This diagnosis invokes the next step.
3. If, as we suspect, the current result is tracking a low-order term with a high coefficient, then this term will dominate $\bar{f}$ for small values of x. Thus we can approximate the upper bound for small x's to be cx^d. Let (x_1, y_1) and (x_2, y_2) be two points from the beginning part of the curve. If we consider that $y_1 \approx cx_1^d$ and $y_2 \approx cx_2^d$, then d can be approximated

by $\frac{\log y_1 - \log y_2}{\log x_1 - \log x_2}$, and c is $\frac{y_1}{x_1^d}$. Now we can correct the model using these estimates, in order to make the high-order term appear. For all points (x, y), we transform y into $\frac{y}{x^d} - c$. Now we can apply the same procedure as above to find the a and b parameters, assuming that $y \approx ax^b$. In this case, though, we use for our estimates two points that have high values of x, as the influence of the high-order term is stronger for these points.

This technique illustrates a way in which models can be improved by generating data and comparing it against the real values to obtain diagnostic information (step 2), which suggests a method specific to the diagnosis—in this case, a method specific to functions with large coefficients on low order terms. (We envision similar diagnostics and methods for functions with negative coefficients, but we haven't designed them, yet.)

The results of this method are found in the columns labeled HY in Figures 1 and 2. The results are tight upper bounds when $\bar{f}$ does in fact contain a low order term with a large coefficient (functions 7, 10, 11, 14, and 17 in Figure 5.5). In fact, these bounds are tighter than those returned by the other methods, and, remarkably, this hybrid method estimates coefficients and low order exponents very well. When the functions do not contain low order terms with large coefficients, the bounds returned by this method remain correct but they are looser than those given by other methods. Interestingly, this situation is often indicated by very low estimated coefficients on the high order terms; for example, in funtion 1 (Fig. 1), the coefficient of the first term is 0.03. The only cases when the technique fails are those in which negative coefficients appear in the low-order terms. The failure is probably due to the sensitivity of the Guess Ratio heuristic to such circumstances. This new method was also tested on noisy datasets but the noise had negligible effects. The new method used different oracles and different implementations of oracles from the previous methods, which might account for the relatively robust performance. Or, the small effects of noise might be due to a different method for sampling data from the given functions. Clearly, the effects of noise on these methods are still poorly understood.

5.6.1 Remark

In our informal and designed experiments with little or no random noise in the data, all the rules generally can get within about a $\sqrt{x}$ factor of the exact bound, except when they become "fooled" by large or negative-valued second-order terms. It is possible to reduce the effect of large second-order terms by taking larger problem sizes, but the rules are slow to respond to this type of change. The hybrid diagnostic method described in Section 5.6 can be used with success on such problems.

On data from algorithmic research problems, the rules can return results within a factor of x and sometimes less (of the correct answer when it is known, and of one another when it is not known). The rules are not reliable in distinguishing low-order and logarithmic factors (this holds even when logarithms are added to the NextOrder oracle). Thus while the simple rules applied here provide fairly reliable conjectures to guide future analytical research when the known bounds are separated by at least a linear factor, more sophisticated approaches (or perhaps better data sets) are necessary if finer distinctions are needed.

It is sometimes possible to improve the data sets to obtain more reliable results. Although the rules do not much respond when the largest problem size is doubled, they do seem to be very responsive to reductions in data variance. This is good news for algorithm analyzers, since variance can be reduced by taking more random trials, and trials are easier to get when Y grows slowly: the situations where small variance is most needed are those situations where small variance is easiest to obtain.

Can Humans Do Better? We have preliminary results concerning interactive uses of the rules. In one experiment, the fourth co-author was given the 25 data sets presented here, without any information about their provenance, and was allowed to use any data analysis approach available in the powerful CLASP library. The human was more frequently incorrect than any of the implemented rules, and the human/machine interactions took much more time to accomplish.

A second experiment involved strict application of the heuristic rules, but with a human oracle (the first co-author) who was familiar with the eight algorithmic data sets. Here also, interactive trials required much more time to perform than did the offline versions (on the order of a few hours rather than a few seconds). Very preliminary results indicate that: the GR produces worse (less close) bounds with a human Trend oracle; the human Concavity oracle tends to agree with the implemented one when used by the Power rules (no change in performance); a human-interactive version of the GD rule is more successful at finding initial DownUp curves (leading to more frequent success), but is not able to find tighter bounds for this rule in general; and an interactive BoxCox can be used to provide upper/lower bounds that bracket the estimate, thus avoiding the “close” and errorneous bounds returned by the implemented version.

Removing Constant Terms. In many applications it may be possible to remove a constant from Y before analysis, either by testing with $x = 0$ or by subtracting an estimated constant. Our preliminary results suggest that subtraction of a known constant uniformly improves all the rules, but subtracting an estimated constant gives mixed results.

Rule Variations. It is a problem for future research to implement and evaluate the many variations on the oracles and the iterative rules GR, GD, and BC. The Guess Ratio rule would probably be improved by a Trend oracle that is robust with respect to negated second terms. Indeed, it is likely that much more sophisticated oracle functions than our simple ones can be developed.

The Guess Difference rule appears to be very sensitive to the initial function and to the granularity of the step functions in the NextOrder and NextCoefficient oracles. So far we cannot find any pattern for this sensitivity. It does seem clear that when an initial guess is too close to the answer, GD fails to find an initial DownUp curve. This rule might be greatly improved by addition of a heuristic search mechanism. Also, we might give the iterative rules fewer options to choose from. The BoxCox rule sometimes improves with coarser step size (because the best transformation gives an exponent somewhere the first and second terms). When the fit is close, however, the BC rule can make erronous bound claims. Thus the rule's goal of finding the best fit works at odds with the goal of finding a reliable bound. The bounds returned by GR and GD nearly always improve when step size decreases. The PWD might be improved by taking differences more than once; one promising idea is to take differences until the data appears concave downwards.

5.7 Discussion

We have seen different aspect of the problem how to identify asymptotic behaviour from experiments. Sections 5.4–5.6 provide us with a few rather general semi-automatic tools for this purpose but also with plenty of examples where these rule do not work.

More successful is the more specific approach based on the scientific method discussed in Section 5.3. But in what sense are these examples "successful"? Assume that using the scientific method we have found an experimentally well supported hypothesis about the running time of an important, difficult to analyze algorithm. How should this result be interpreted? It may be viewed as a conjecture for guiding further theoretical research for a mathematical proof. If this proof is not found, a well tested hypothesis may also serve as a surrogate. For example, in algorithmics the hypotheses "a good implementation of the simplex method runs in polynomial time" or "NP-complete problems are hard to solve in the worst case" play an important role. The success of the scientific method in the natural sciences — even where deductive results would be possible in principle — is a further hint that such hypotheses may play an increasingly important role in algorithmics. For example, Cohen-Tannoudji et al. [5.13] (after 1095 pages of deductive results) state that "in all fields of physics, there are very few problems which can be treated completely analytically." Even a simple two-body system like the hydrogen atom cannot be handled analytically without making simplifying assumptions (like handling the proton classically). For the same reason, experiments are of utmost importance in chemistry although there is little doubt that well known laws like the Schrödinger equation in principle could explain most of chemistry.

Of course, no tool is perfect, and the hazards of extrapolating from experimental data to find reliable asymptotic bounds can not be ignored. Our study of five simple heuristic strategies (with variations) suggests that any of the approaches can produce a correct asymptotic bound within an order of magnitude when the data set is well-behaved: that is, when there is very little random noise in the y-values, and when the largest problem size is large enough to overcome "noise" due to large constant factors in low order terms.

However, when the research problem requires inferences about bounds that are more finely-tuned than one order of magnitude (for example, whether a function grows as $O(n)$ vs $O(n \log n)$, or whether a root-n factor is present), the five rules become unreliable, especially when the quality of data deteriorates. The rules are quite sensitive to random variation in the y-values, and somewhat less sensitive to changes in the largest problem size.

In these types of experimental situations, then, the extrapolation techniques described here must be used with caution, and/or steps must be taken to improve the quality of the data obtained from the experiment. Fortunately, in many algorithmic research problems it is easy to reduce variance in the experimental data by taking more experiments or by applying variance reduction techniques. It does appear to be an important component of good experimental practice to set problem sizes as large as possible, so as to overcome any possible interference from low order terms.

It is an interesting open research problem to develop better and more sophisticated strategies for obtaining reliable asymptotic inferences from algorithmic experiments.

References

5.1 A. C. Atkinson. *Plots, Transformations and Regression: an Introduction to Graphical Methods of Diagnostic Regression Analysis.* Oxford University Press, U.K., 1987.

5.2 Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. In *Proceedings of the 26th ACM Symposium on the Theory of Computation (STOC'94)*, pages 593–602, 1994.

5.3 D. Baldwin and J. A. G. M. Koomen. Using scientific experiments in early computer science laboratories. *ACM SIGCSE Bulletin*, 24(1):102–106, 1992.

5.4 R. S. Barr, R. V. Helgaon, and J. L. Kennington. Minimal spanning trees: An empirical investigation of parallel algorithms. *Parallel Computing*, 12:45–52, 1989.

5.5 R. A. Becker, J. A. Chambers, and A. R. Wilks. *The New S Language: A Programming Enviornment for Data Analysis and Graphics.* Wadsworth & Brooks/Cole, 1988.

5.6 J. L. Bentley, D. S. Johnson, F. T. Leighton, and C. C. McGeoch. An experimental study of bin packing. In *Proceedings of the 21th Annual Allerton Conference on Communication, Control, and Computing*, pages 51–60, 1983.

5.7 J. L. Bentley, D. S. Johnson, C. C. McGeoch, and L. A. McGeoch. Some unexpected expected behavior results for bin packing. In *Proceedings of the 16th ACM Symposium on Theory of Computation (STOC'84)*, pages 279–298, 1984.

5.8 P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking. Balanced allocations: the heavily loaded case. In *Proceedings of the 32nd ACM Symposium on the Theory of Computation (STOC'00)*, 2000.

5.9 R. D. Blumofe and C. E. Leiserson. Scheduling multithreaded computations by work stealing. In *Proceedings of the 35th Symposium on Foundations of Computer Science (FOCS'94)*, pages 356–368, 1994.

5.10 G. P. Box, W. G. Hunter, and J. S. Hunter. *Statistics for Experimenters.* John Wiley & Sons, Inc., Chichester, 1978.

5.11 J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey. *Graphical Methods for Data Analysis.* Duxbury Pres, Boston, 1983.

5.12 P. R. Cohen. *Empirical Methods for Artificial Intelligence.* The MIT Press, Cambridge, MA, and London, England, 1995.

5.13 C. Cohen-Tannoudji, B. Diu, and F. Laloë. *Quantum Mechanics*, volume 2. John Wiley & Sons, Inc., Chichester, 1977.

5.14 P. J. Denning. What is experimental computer science? *Communications of the ACM*, 23(10):543–544, 1980.

5.15 P. J. Denning. Performance analysis: Experimental computer science at its best. *Communications of the ACM*, 24(11):725–727, 1981.

5.16 N. Fenton, S. L. Pfleger, and R. L. Glass. Science and substance: A challenge to software engineers. *IEEE Software*, 11(4):86–95, 1994.

5.17 J. N. Hooker. Needed: An empirical science of algorithms. *Operations Research*, 42(2):201–212, 1994.

5.18 T. Jiang, M. Li, and P. Vitányi. Average-case complexity of Shellsort. In *Proceedings of the 26th International Colloquium on Automata, Languages and Programming (ICALP'99).* Springer Lecture Notes in Computer Science 1644, pages 453–462, 1999.

5.19 D. S. Johnson. A theoretician's guide to the experimental analysis of algorithms, 1996. Manuscript.

5.20 D. E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching.* Addison-Wesley, Reading, MA, 2nd edition, 1998.

5.21 V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing. Design and Analysis of Algorithms.* Benjamin/Cummings, 1994.

5.22 T. Leighton. *Introduction to Parallel Algorithms and Architectures.* Morgan Kaufmann, 1992.

5.23 P. Lukowicz, E. A. Heinz, L. Prechelt, and W. F. Tichy. Experimental evaluation in computer science: A quantitative case study. *Journal of Systems and Software*, 28(1):9–18, 1995.

5.24 M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8:3–30, 1998. `http://www.math.keio.ac.jp/~matumoto/emt.html`.

5.25 C. C. McGeoch. Analyzing algorithms by simulation: Variance reduction techniques and simulation speedups. *ACM Computing Surveys*, 245(2):195–212, 1992.

5.26 C. C. McGeoch. All pairs shortest paths and the essential subgraph. *Algorithmica*, 13:426–441, 1995.

5.27 C. C. McGeoch. Toward an experimental method for algorithm simulation, 1996.

5.28 C. C. McGeoch and B. Moret. How to present a paper on experimental work with algorithms. *SIGACT News*, 30(4):85–90, 1999.

5.29 B. M. E. Moret. Towards a discipline of experimental algorithmics. In *5th DIMACS Challenge*, DIMACS Monograph Series, 2000. to appear.

5.30 R. Niedermeier, K. Reinhard, and P. Sanders. Towards optimal locality in mesh-indexings. In *Proceedings of the 11th International Conference on Fundamentals of Computation Theory (FCT'97).* Springer Lecture Notes in Computer Science 1279, pages 364–375, 1997.

5.31 K. R. Popper. *Logik der Forschung.* Springer-Verlag, Heidelberg, 1934. English Translation: *The Logic of Scientific Discovery*, Hutchinson, 1959.

5.32 W. H. Press, S. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C.* Cambridge University Press, Cambridge, England, 2. edition, 1992.

5.33 J. O. Rawlings. *Applied Regression Analysis: A Research Tool.* Wadsworth & Brooks/Cole, 1988.

5.34 P. Sanders. *Lastverteilungsalgorithmen für parallele Tiefensuche.* Number 463 in Fortschrittsberichte, Reihe 10. VDI Verlag, 1997.

5.35 P. Sanders, S. Egner, and J. Korst. Fast concurrent access to parallel disks. In *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA'00)*, pages 849–858, 2000.

5.36 C. Schaffer. *Domain-Independent Scientific Function Finding.* Ph.D. thesis, Department of Computer Science, Rutgers University, 1990.

5.37 Computer Science and Telecommunications Board. Academic careers for experimental computer scientists and engineers. *Communications of the ACM*, 37(4):87–90, 1994.

5.38 R. Sedgewick. Analysis of shellsort and related algorithms. In *Proceedings of the 4th European Symposium on Algorithms (ESA'96).* Springer Lecture Notes in Computer Science 1136, pages 1–11, 1996.

5.39 D. L. Shell. A high-speed sorting procedure. *Communications of the ACM*, 2(7):30–33, 1958.

5.40 J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis.* Springer-Verlag, Heidelberg, 1993.

5.41 W. F. Tichy. Should computer scientists experiment more? *Computer*, 31(5):32–40, 1998.

5.42 J. W. Tukey. *Exploratory Data Analysis.* Addison-Wesley, Reading, MA, 1977.

5.43 B. Vöcking. How asymmetry helps load balancing. In *Proceedings of the 40th Symposium on Foundations of Computer Science (FOCS'99)*, pages 131–140, 1999.

5.44 L. Weisner. *Introduction to the Theory of Equations.* The MacMillan Press Ltd., London, 1938.

5.45 M. A. Weiss. Empirical study of the expected running time of Shellsort. *The Computer Journal*, 34(1):88–91, 1991.