Asymptotic Complexity from Experiments? A Case Study for Randomized Algorithms

Peter Sanders[1]*, Rudolf Fleischer[2]

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

[2] Department of Computer Science
University of Waterloo
200 University Avenue West
rudolf@uwaterloo.ca

Abstract. In the analysis of algorithms we are usually interested in obtaining closed form expressions for their complexity, or at least asymptotic expressions in $\mathcal{O}(\cdot)$-notation. Unfortunately, there are fundamental reasons why we cannot obtain such expressions from experiments. This paper explains how we can at least come close to this goal using the scientific method. Besides the traditional role of experiments as a source of preliminary ideas for theoretical analysis, experiments can test falsifiable hypotheses obtained by incomplete theoretical analysis. Asymptotic behavior can also be deduced from stronger hypotheses which have been induced from experiments. As long as a complete mathematical analysis is impossible, well tested hypotheses may have to take their place. Several examples for probabilistic problems are given where the average complexity can be well approximated experimentally so that the support for the hypotheses is quite strong. Randomized Shellsort has performance close to $\mathcal{O}(n \log n)$; random polling dynamic load balancing between P processors achieves full load sharing in $\log_2 P + \mathcal{O}(\log \log P)$ steps; randomized writing to D independent disks using a shared buffer size W achieves average efficiency at least $1 - D/(2W)$.

1 Introduction

The complexity analysis of algorithms is one of the core activities of computer scientists in general and in the branch of theoretical computer science known as algorithmics in particular. The ultimate goal would be to find closed form expressions for the runtime or other measures of ressource consumption. Since this is often too complicated, we are usually content with asymptotic expressions for the worst case complexity depending on input parameters like the problem size. Even this task can be very difficult so that it is important to use all available tools.

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

This paper investigates to what extent experiments can help to find these expressions. It is common practice to plot complexity measures derived from experiments to generate conjectures. But people are rightfully wary about over-interpretation of these results so that the experimental "scaffolding" usually disappears from the publication of the results. Here, it is explained why with some care experiments can often play a stronger role. One way to make the meaning of "some care" more precise is to apply the terminology of the scientific method [15]. The scientific method views science as a cycle between theory and practice. Theory can inductively or (partially) deductively 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.

Sect. 2 reviews some of the main problems and explains how to partially solve them. Sect. 3 surveys some related work and Sect. 4 gives several concrete examples for randomized algorithms whose expected ressource consumption only depends on the input size but which are nontrivial to analyze analytically. Finally, Sect. 5 discusses the role of results found using the scientific method.

2 Some Problems with Experiments

Too many inputs: Perhaps the most fundamental problem with experiments in algorithmics is that we can rarely test all possible inputs even for bounded input size because there are usually exponentially many of them. If we do application oriented research this problem may be mitigated by libraries of test instances which are considered "typical" (e.g. [5]). Here we want to concentrate on experiments as a tool for theory however so that we are interested in cases where it is possible to bound the complexity for all inputs of size n in time polynomial in n. 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 [14].

Experiments are more important for randomized algorithms. Often randomization can convert all instances into average case instances. For example, every sorting algorithm which is efficient on the average can be transformed into an efficient algorithm for worst case inputs by permuting the inputs randomly. In this case, a few hundred experiments with random inputs can give a reliable picture of the expected performance of the algorithm for inputs of the given size. On the other hand, closed form analysis of randomized algorithms can be very difficult. For example, the average case performance of Shellsort is open for a long time now [19]. Also refer to Sect. 4.3.

Unbounded input size: Another problem with experiments is that we can only test a finite number of input sizes. For example, assume we find that some sorting algorithm needs an average of $C(n) \leq 3n \log n$ comparisons for $n < 10^6$ elements. We still cannot claim that $C(n) \leq 3n \log n$ is 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 $C(n) > 3n \log n$. Note that not every sound hypothesis is a good hypothesis. For example, it we would cowardly change the above hypothesis to $C(n) \leq 100000n \log n$ it would be difficult to falsify it even if it later turns out that the true bound is $C(n) = n \log n + 0.1n \log^2 n$. But qualitative issues like accuracy, simplicity, and generality of hypotheses are also an issue in the natural sciences and this does not hinder people to use the scientific method there.

$\mathcal{O}(\cdot)$-s are not falsifiable: The next problem is that asymptotic expressions 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 $C(n) \leq \mathcal{O}(n \log n)$ comparisons it cannot even be falsified by a set of inputs which clearly shows 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 Sect. 4.3.

Complexity of the Machine Model: Although the actual execution time of an algorithm is perhaps the most interesting subject of analysis, this measure of ressource consumption is often difficult to model by closed form expressions. Caches, virtual memory, memory management, compilers and interference with other processes all influence execution time in a difficult to predict way.[1] At some loss of accuracy, this problem can be solved by counting the number of times a certain set of source code operations is executed which cover all the inner loops of the program. This count suffices to grasp 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 an appropriate formula for a worst case upper bound given only measurements; even if the investigated ressource consumption only depends on the input size. The measured function may be nonmonotonic while we are only interested in a monotonic upper bound. There are often considerable contributions of lower order terms for small inputs. Experience shows that curve fitting often won't

[1] Remember that the above complexity 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.

work in particular if we are interested in fine distinctions like logarithmic factors [12]. Again, the scientific method helps to mitigate this problem. Often, we are able to handle a related or simplified version of the system analytically or we can 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. Sect. 4 gives several examples of this powerful approach which so far seems to be underrepresented in algorithmics.

3 Related Work

The importance of experiments in algorithm design has recently gained much attention. New workshops (ALENEX, WAE) and journals (ACM J. of Experimental Algorithmics) have been installed and established conferences (e.g., SODA, ESA) explicitly call for experimental work. Using the scientific method as a basis for algorithmics was proposed by Hooker [6]. McGeoch, Precup and Cohen [12] give heuristic algorithms for finding upper bounds on measured function values which are found to be reliable within a factor $\sqrt{n}$. They stress that finding more accurate bounds would be futile in general. This is no contradiction to the examples given in Sect. 4 where even $\log\log n$ terms are discussed because Sect. 4 uses additional problem specific information via the scientific method.

4 Examples

Our first example in Sect. 4.1 can be viewed as the traditional role of experiments as a method to generate conjectures on the behavior of algorithms but it has an additional interpretation where the experiment plus theory on a less attractive algorithm yields a useful hypothesis. Sect. 4.2 gives an example where an experiment is used to validate a simplification made in the middle of a derivation. Sections 4.3 and 4.4 touch the difficult question of how to use experiments to learn something about the asymptotic complexity of an algorithm. In addition, Sect. 4.4 is a good example how experiments can suggest that an analysis can be sharpened.

This paper only scratches the surface of a related important methodological topic; namely how to perform experiments accurately and efficiently and how to evaluate the confidence in our findings statistically. In Sect. 4.2 we apply such a statistical test and find a very high level of confidence. In Sect. 4.4 we give an example how the number of repetitions can be coupled to the measured standard error. We also shortly discuss the choice of random number generator.

4.1 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

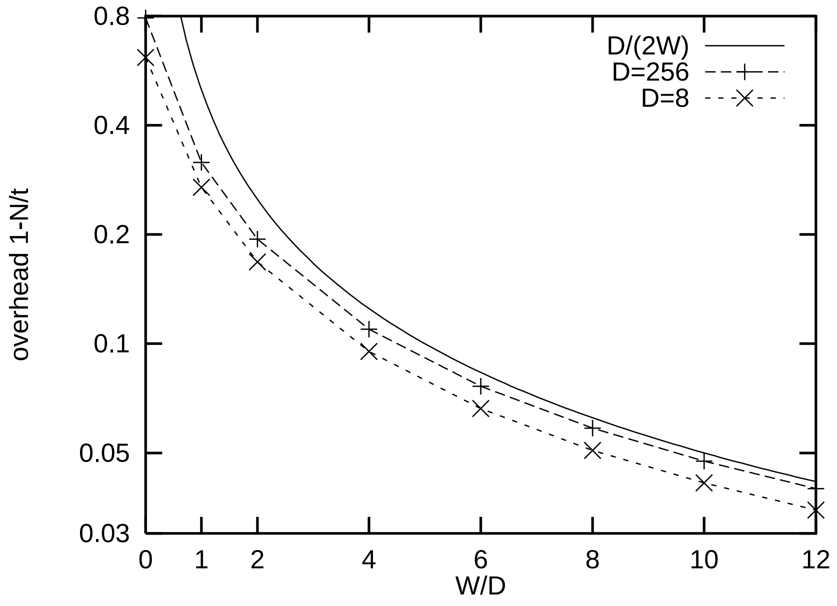

Fig. 1. Overhead (i.e., 1−efficiency) of EAGER. $N = 10^6 \cdot D$ blocks were written.

general technique for scheduling parallel disks [18]. We maintain one queue Q_i for each disk. The queues share a buffer space of size $W = \mathcal{O}(D)$. We first put all the blocks into the queues and then write one block from each nonempty queue. If after that the sum of the queue lengths exceeds W, additional write steps are invested. We have no idea how to analyze this algorithm. Therefore, in [18] 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 $D/(2\epsilon)$. 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.[2]

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. 1. It looks like $1 - D/(2W)$ 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 observation that algorithm EAGER with ϵ slightly larger than $D/(2W)$ would yield a similar efficiency.

More generally speaking, the algorithms we are most interested in might be too difficult to understand analytically. Then it makes sense to analyze a related

[2] 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.

and possibly inferior algorithm and to use the scientific method to come to theoretical insights about the original algorithm. In theoretical computer science, the latter step is sometimes omitted leading to friction between theory and practice.

4.2 "Heuristic" Deduction: Random Polling

Let us consider the following simplified model for the startup phase of *random polling dynamic load balancing* [9,3,17] 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 at most double in each step. An analysis for a more general model yields an $\mathbf{E}[T] = \mathcal{O}(\log n)$ upper bound [17]. We will now argue that there is a much tighter upper bound of $\mathbf{E}[T] \leq \log n + \log\ln n + 1$.

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 $\mathbf{P}[X_{i0}=1] = 1-1/n$. Let $U_k = \sum_{i<n} X_{ik}$. We have

$$\mathbf{E}U_k = \mathbf{E}\sum_{i<n} X_{ik} = \sum_{i<n} \mathbf{P}[X_{ik}=1] = n\mathbf{P}[X_{ik}=1] \quad .$$

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

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

and, by induction,

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

Therefore, $\mathbf{E}[U_k] \geq n(1-e^{-2^k/n})$ and for $k = \log n + \log\ln n$, $\mathbf{E}[U_k] \geq n-1$. 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. 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.

The probability that the measured values are only accidentally below the conjectured bound can be estimated using the Student-t test. Following [13] we get a probability

$$1 - A\left(\sqrt{1000}\frac{\bar{T} - (\log n + \log\ln n + 1)}{\sigma}|999\right)$$

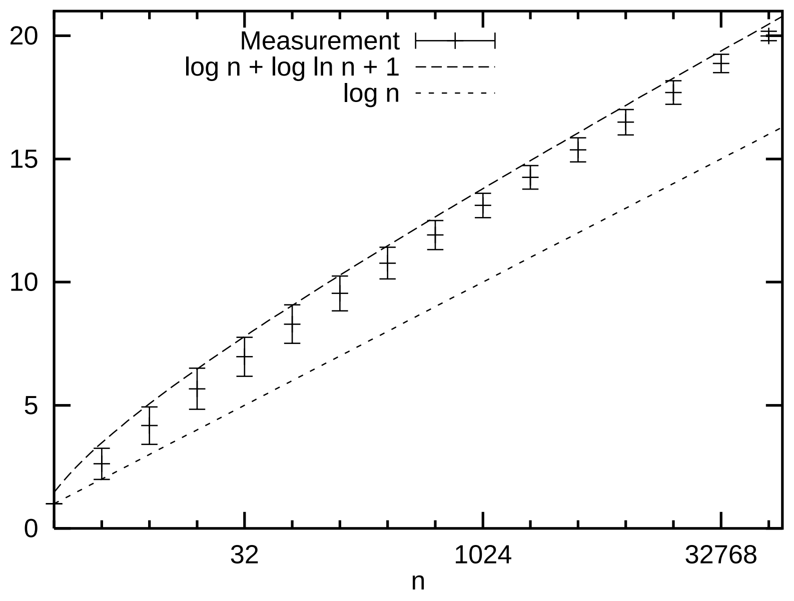

Fig. 2. Number of random polling steps to get all PEs busy: Hypothesized upper bound, lower bound and measured averages with standard deviation.

where $\bar{T}$ is the measured average, σ is the measured standard deviation and $A(t|\nu)$ is the cumulative distribution function of the Student t-distribution with ν degrees of freedom. Within the computational precision of Maple, this probability is zero.

4.3 Shellsort

Shellsort [20] is a classical sorting algorithm which is considered a good algorithm for almost sorted inputs in particular, if an in-place routine is desired or small to medium sized inputs are considered. 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
```

Interestingly, 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 wether there is an offset sequence which yields a sorting algorithm with $T(n) = \mathcal{O}(n \log n)$ or even one with $T(n) = o(n \log^2 n)$ [19,7]. It is known that any algorithm with $T(n) = \mathcal{O}(n \log n)$ must use $\Theta(\log n)$ offsets [7]. Previous experiments with many carefully constructed offset sequences led to the conjecture that no sequence yields $T(n)$ close to $\mathcal{O}(n \log n)$ [22].

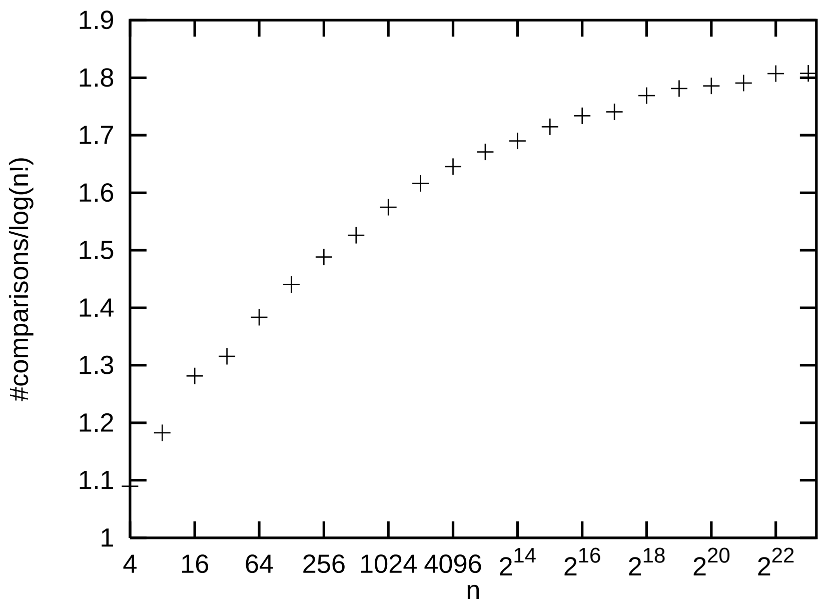

Fig. 3. Competitive ratio of the average number of key comparisons of random offset Shellsort compared to the information theoretic lower bound $\log(n!)$. We used $h_i := \lfloor h_{i-1} \cdot f_i + 1 \rfloor$ where f_i is a random factor from the interval $[0, 4]$. Averages are based on 1000 repetitions for $n \leq 2^{13}$ and 100 repetitions for larger inputs.

Led by the successful use of randomness for sorting networks [10, Sect. 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 have used offsets which are the product of random numbers. The situation now is more difficult than in Sect. 4.2 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. 3. According to the conjecture in [22] 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 *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. 3. The curve gets flatter and flatter and its inclination might even converge to zero.

We might conjecture that $T(n) = \mathcal{O}\left(n \log^{1+o(1)} n\right)$. 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. One thing we could do however is to hypothesize that $2^{T(n)/\log(n!)}$ is a concave function. This hypothesis is falsifiable and together with the measurements it implies[3] $T(n) = \mathcal{O}\left(n \log^{1+\epsilon} n\right)$ for quite small values of ϵ which we can further decrease by doing measurements for larger n.

4.4 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

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

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

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

whp can be proven.

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. [1] 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. Only very recently Berenbrink et al. [2] have shown (using quite nontrivial arguments) that

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

[3] We mean logical implication here, i.e., if the hypothesis is false nothing is said about the truth of the implied assertion.

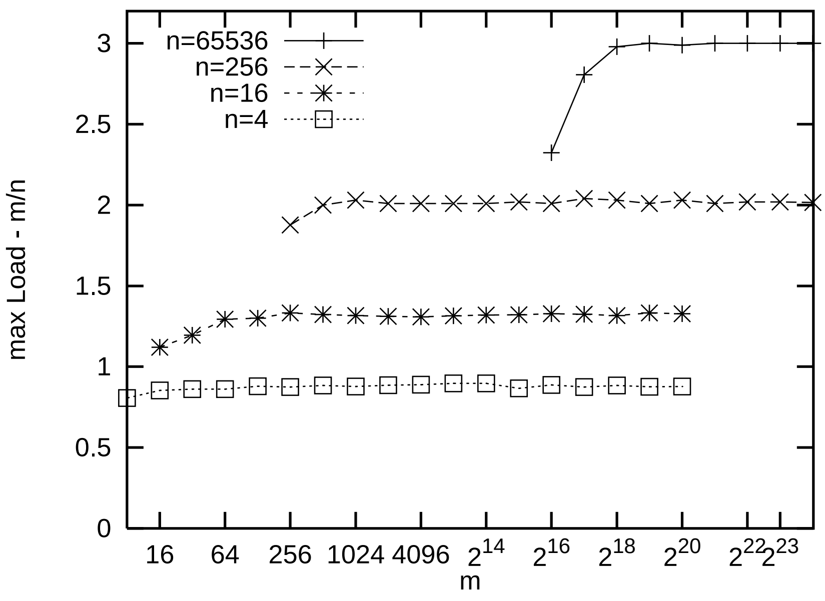

Fig. 4. Excess load for randomized balanced allocation as a function of n for different n. The experiments have been repeated at least sufficiently often to reduce the *standard error* $\sigma/\sqrt{\text{repetitions}}$ [16] below one percent of the average excess load. In order to minimize artifacts of the random number generator, we have used a generator with good reputation and very long period ($2^{19937}-1$) [11]. In addition, we have repeated some experiments with the Unix generator `srand48` leading to almost identical results.

Fig. 4 shows that a simple experiment at least predicts that $l_{\max} - m/n$ cannot depend much on m. Other researchers (e.g. [8]) made some experiments but without trying to induce hypotheses on the asymptotic behavior of balanced allocation.

Our experiments were done before the theoretical solution. Otherwise, we could have picked one of the other open problems in the area of balls into bins games. For example, Vöcking [21] recently proved that an 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 Discussion

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. [4] (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.

References

1. Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. In *26th ACM Symposium on the Theory of Computing*, pages 593–602, 1994.
2. P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking. Balanced allocations: The heavily loaded case. In *32th Annual ACM Symposium on Theory of Computing*, 2000.
3. R. D. Blumofe and C. E. Leiserson. Scheduling multithreaded computations by work stealing. In *Foundations of Computer Science*, pages 356–368, Santa Fe, 1994.
4. C. Cohen-Tannoudji, B. Diu, and F. Laloë. *Quantum Mechanics*, volume 2. John Wiley & Sons, Inc., 1977.
5. A. Goldberg and B. Moret. Combinatorial algorithms test sets (cats). In *10th ACM-SIAM Symposium on Discrete Algorithms*, 1999.
6. J. Hooker. Needed : An empirical science of algorithms. *Operations Res.*, 42(2):201–212, 1994.
7. T. Jiang, M. Li, and P. Vitányi. Average-case complexity of shellsort. In *ICALP*, number 1644 in LNCS, pages 453–462, 1999.
8. J. Korst. Random duplicate assignment: An alternative to striping in video servers. In *ACM Multimedia*, pages 219–226, Seattle, 1997.
9. V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing. Design and Analysis of Algorithms.* Benjamin/Cummings, 1994.
10. T. Leighton. *Introduction to Parallel Algorithms and Architectures.* Morgan Kaufmann, 1992.
11. M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACMTMCS: ACM Transactions on Modeling and Computer Simulation*, 8:3–30, 1998. `http://www.math.keio.ac.jp/~matumoto/emt.html`.
12. C. C. McGeoch, D. Precup, and P. R. Cohen. How to find big-oh in your data set (and how not to). In *Advances in Intelligent Data Analysis*, number 1280 in LNCS, pages 41–52, 1997.
13. P. H. Müller. *Lexikon der Stochastik.* Akademie Verlag, 5th edition, 1991.
14. R. Niedermeier, K. Reinhard, and P. Sanders. Towards optimal locality in mesh-indexings. In B. S. Chlebus and L. Czaja, editors, *Fundamentals of Computation Theory*, number 1279 in LNCS, pages 364–375, Krakow, 1997.

15. K. R. Popper. *Logik der Forschung.* Springer, 1934. English Translation: *The Logic of Scientific Discovery*, Hutchinson, 1959.
16. W. H. Press, S. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C.* Cambridge University Press, 2. edition, 1992.
17. P. Sanders. *Lastverteilungsalgorithmen für parallele Tiefensuche.* Number 463 in Fortschrittsberichte, Reihe 10. VDI Verlag, 1997.
18. P. Sanders, S. Egner, and J. Korst. Fast concurrent access to parallel disks. In *11th ACM-SIAM Symposium on Discrete Algorithms*, pages 849–858, 2000.
19. R. Sedgewick. Analysis of shellsort and related algorithms. *LNCS*, 1136:1–11, 1996.
20. D. L. Shell. A high-speed sorting procedure. *Communications of the ACM*, 2(7):30–33, July 1958.
21. B. Vöcking. How asymmetry helps load balancing. In *40th FOCS*, pages 131–140, 1999.
22. M. A. Weiss. Empirical study of the expected running time of shellsort. *The Computer Journal*, 34(1):88–91, 1991.