Presenting Data from Experiments in Algorithmics

Peter Sanders[1]*

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

Abstract. Algorithmic experiments yield large amounts of data that depends on many parameters. This paper collects a number of rules for presenting this data in concise, meaningful, understandable graphs that have sufficiently high quality to be printed in scientific journals. The focus is on common sense rules that are frequently useful and can be easily implemented using tools such as gnuplot[1].

1 Introduction

A paper in experimental algorithmics will often start by describing the problem and the experimental setup. Then a substantial part will be devoted to presenting the results together with their interpretation. Consequently, compiling the measured data into graphs is a central part of writing such a paper. This problem is often rather difficult because several competing factors are involved. First, the measurements can depend on many parameters: problem size and other quantities describing the problem instance; variables like number of processors, available memory describing the machine configuration used; and the algorithm variant together with tuning parameters such as the cooling rate in a simulated annealing algorithm.

Furthermore, many quantities can be measured such as solution quality, execution time, memory consumption and other more abstract complexity measures such as the number of comparisons performed by a sorting algorithm. Mathematically speaking, we sample function values of a mapping $f : A \to B$ where the domain A can be high-dimensional. We hope to uncover properties of f from the measurements, e.g., an estimate of the time complexity of an algorithm as a function of the input size. Measurement errors may additionally complicate this task.

As a consequence of the the multitude of parameters, a meaningful experimental setup will often produce large amounts of data and still cover only a tiny fraction of the possible measurements. This data has to be presented in

* This work was partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

[1] www.gnuplot.org. The source codes of the examples in this paper can be found under http://www.mpi-sb.mpg.de/~sanders/gnuplot/

a way that clearly demonstrates the observed properties. The most important presentation usually takes place in conference proceedings or scientific journals where limited space and format restriction further complicate the task.

This paper collects rules that have proven to be useful in designing good graphs. Although the examples are drawn from the work of the author, this paper owes a lot to discussions with colleagues and detailed feedback from several referees. Sections 3–7 explains the rules. The stress is on Section 4 where two-dimensional figures are discussed in detail.

Instead of an abstract conclusion, Section 8 collects all the rules in a check list that can possibly be used when looking for teaching and as a source of ideas for improving graphs.

Related Work

A number of papers on the methodology of experimental algorithmics have come out recently [10, 8, 9, 6]. In particular, [6] explains some of the rules presented here.

There are also entire books on presenting data graphically [5, 4, 17]. The role of the present paper is to formulate domain specific rules, to adapt and specialize more abstract rules and to summarize less important rules. For example, the main emphasis of the above books is on approaches to visualize a limited set of data items in ways which discern structure. Tufte even reports that 75 % of the graphics found in newspapers and magazines are time series — a species of graphs rather rare in algorithmics, where we often face a different situation. We have instance generators which provide us with an unlimited supply of examples and we have control over many parameters. Furthermore, we can repeat experiments as often as we want and hence can often reduce measurement errors to quite small values. The difficulty is now to select the right measurements and display a large amount of data in a compact way.

Another active area of research is the visualization of large amounts of data using three-dimensional, colored animations. Here we limit ourselves to simple graphs suited for black-and-white printing that can be produced with off-the-shelf tools like gnuplot. This paper should not be regarded as a research paper but as a collection of "folklore" rules.

2 The Process

In a simplified model of experimental algorithmics a paper might be written using a "waterfall model". The experimental design is followed by a description of the measurement which is in turn followed by an interpretation. In reality, there are numerous feedbacks involved and some might even remain visible in a presentation. After an algorithm has been implemented, one typically builds a simple yet flexible tool that allows many kinds of measurements. After some explorative measurements the researcher gets a basic idea of interesting parameter settings.

Hypotheses are formed which are tested using more extensive measurements using particular parameter ranges. This phase is the scientifically most productive phase and often leads to new insights which lead to algorithmic changes which influence the entire setup.

It should be noted that most algorithmic problems are so complex that one cannot expect to arrive at an ultimate set of measurements that answers all conceivable questions. Rather, one is constantly facing a list of interesting open questions that require new measurements. The process of selecting the measurements that are actually performed is driven by risk and opportunity: The researcher will usually have a set of hypotheses that have some support from measurements but more measurements might be important to confirm them. For example, the hypothesis might be "my algorithm is better than all the others" then a big risk might be that a promising other algorithm or important classes of problem instances have not been tried yet. A small risk might be that a tuning parameter has so far been set in an ad hoc fashion where it is clear that it can only improve a precomputation phase that takes 20 % of the execution time.

An opportunity might be a new idea of the authors' that an algorithm might be useful for a new application where it was not originally designed for. In that case, one might consider to include problem instances from the new application into the measurements.

At some point, a group of researchers decides to cast the current state of results into a paper. The explorative phase is then stopped for a while. To make the presentation concise and convincing, alternative ways to display the data are designed that are compact enough to meet space restrictions and make the conclusions evident. This might also require additional measurements giving additional support to the hypotheses studied.

3 Tables

Tables are easier to produce than graphs and perhaps this advantage causes that they are often overused. Tables are more difficult to interpret and too large for large data sets. A more detailed explanation why tables are often a bad idea has been given by McGeoch and Moret [9]. Nevertheless, tables have their place. Tufte [17] gives the rule of thumb that "tables usually outperform a graph for small data sets of 20 numbers or less". Tables give very accurate values which make it easier to check whether some experiments can be reproduced. Furthermore, one sometimes wants to present some quantities, e.g., solution quality, as a function of problem instances which cannot be meaningfully arranged on the axis of a graph. In that case, a graph or bar chart may look nicer but does not add utility compared to a more accurate and compact table. Often a paper will contain small tables with particularly important results and graphs giving results in an abstract yet less accurate way. Furthermore, there may be an appendix or a link to a web page containing larger tables for more detailed documentation of the results.

4 Two-dimensional Figures

As our standard example we will use the case that execution time should be displayed as a function of input size. The same rules will usually apply for many other types of variables. Sometimes we mention special examples which should be displayed differently.

4.1 The x-Axis

The first question one can ask oneself is what unit one chooses for the x-axis. For example, assume we want to display the time it takes to broadcast a message of length k in some network where transmitting k' bytes of data from one processor to another takes time $t_0 + k'$. Then it makes sense to plot the execution time as a function of k/t_0 because for many implementations, the shape of the curve will then become independent of t_0. More generally, by choosing an appropriate unit, we can sometimes get rid of one degree of freedom. Figure 1 gives an example.

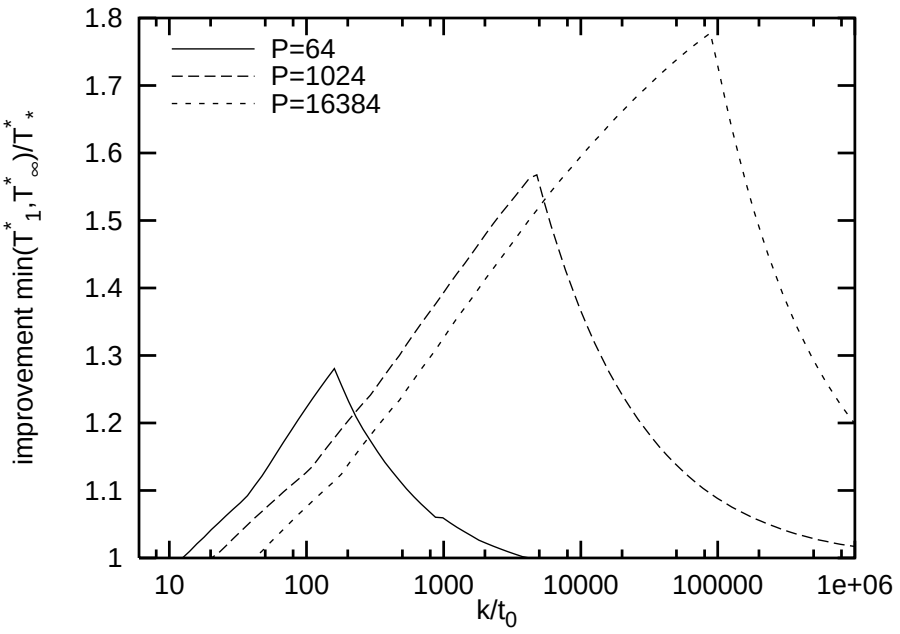

Fig. 1. Improvement of the fractional tree broadcasting algorithm [15] over the best of pipelined binary tree and sequential pipeline algorithm as a function of message transmission time k over startup overhead t_0. P is the number of processors. (See also Section 4.3 and 4.5)

The variable defining the x-axis can often vary over many orders of magnitude. Therefore one should always consider whether a logarithmic scale is appropriate for the x-axis. This is an accepted way to give a general idea of a function

over a wide range of values. One will then choose measurement values such that they are about evenly spaced on the x-axis, e.g., powers of two or powers of $\sqrt{2}$. Figures 3, 5, and 6 all use powers of two. In this case, one should also choose tic marks which are powers of two and not powers of ten. Figures 1 and 4 use the "default" base ten because there is no choice of input sizes involved here.

Sometimes it is appropriate to give more measurements for small x-values because they are easily obtained and particularly important. Conversely, it is not a good idea to measures using constant offsets ($x \in \{x_0 + i\Delta : 0 \leq i < i_{\max}\}$) as if one had a linear scale and then to display the values on a logarithmic scale. This looks awkward because points are crowded for large values. Often there will be too few values for small x and one nevertheless wastes a lot of measurement time for large inputs.

A plain linear scale is adequate if the interesting range of x-values is relatively small, for example if the x-axis is the number of processors used and one measures on a small machine with only 8 processors. A linear scale is also good if one wants to point out periodic behavior, for example if one wants to demonstrate that slow-down due to cache conflicts get very large whenever the input size is a multiple of the cache size. However, one should resist the temptation to use a linear scale when x-values over many orders of magnitude are important but the own results look particularly good for large inputs.

Sometimes, transformations of the x-axis other than linear or logarithmic make sense. For example, in queuing systems one is often interested in the delay of requests as the system load approaches the maximum performance of the system. Figure 2 gives an example. Assume we have a disk server with 64 disks. Data is placed randomly on these disks using a hash function. Assume that retrieving a block from a disk takes one time unit and that there is a periodic stream of requests — one every $(1+\epsilon)/64$ time units. Using queuing theory one can show that the delay of a request is approximately proportional to $1/\epsilon$ if only one copy of every block is available. Therefore, it makes sense to use $1/\epsilon$ as the x-value. First, this transformation makes it easy to check whether the system measured also shows this behavior linear in $1/\epsilon$. Second, one gets high resolution for arrival rates near the saturation point of the system. Such high arrival rates are often more interesting than low arrival rates because they correspond to very efficient uses of the system.

4.2 The y-Axis

Given that the x-axis often has a logarithmic scale, we often seem to be forced to use a logarithmic scale also for the y-axis. For example, if the execution time is approximately some power of the problem size, such a double-logarithmic plot will yield a straight line.

However, plots of the execution time can be quite boring. Often, we already know the general shape of the curve. For example, a theoretical analysis may tell us that the execution time is between $T(n) = \Omega(n)$ and $T(n) = \mathcal{O}(n\text{Polylog}(n))$. A double-logarithmic plot will show something very close to a diagonal and discerns very little about the Polylog term we are really interested in. In such a

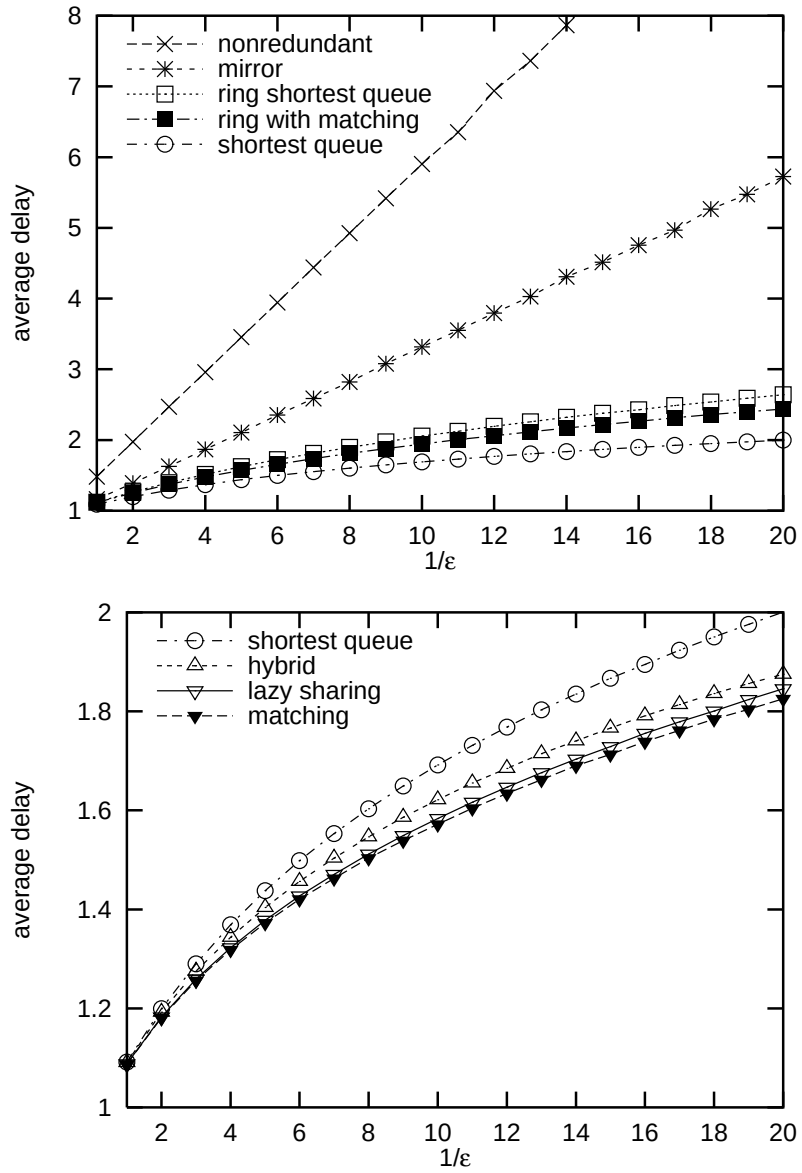

Fig. 2. Comparison of eight algorithms for scheduling accesses to parallel disks using the model described in the text (note that "shortest queue" appears in both figures). Only the two algorithms "nonredundant" and "mirror" exhibit a linear behavior of the access delay predicted by queuing theory. The four best algorithms are based on *random duplicate allocation* — every block is available on two randomly chosen disks and a scheduling algorithm [13] decides which copy to retrieve. (See also Section 4.3)

situation, we transform the y-axis so that a priori information is factored out. In our example above we could better display $T(n)/n$ and then use a linear scale for the y-axis. A disadvantage of such transformations is that they may be difficult to explain. However, often this problem can be solved by finding a good term describing the quantity displayed. For example, "time per element" when one divides by the input size, "competitive ratio" when one divides by a lower bound, or "efficiency" when one displays the ratio between an upper performance bound and the measured performance. Figure 3 gives an example for using such a ratio.

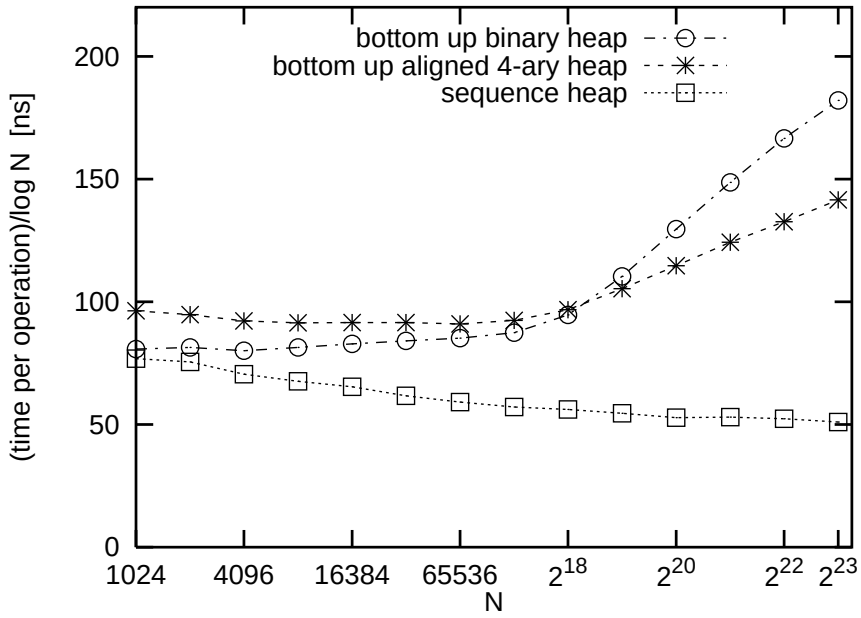

Fig. 3. Comparison of three different priority queue algorithms [16] on a MIPS R10000 processor. N is the size of the queue. All algorithms use $\Theta(\log N)$ key comparisons per operation. The y-axis shows the total execution time for some particular operation sequence divided by the number of deletion/insertion pairs and $\log N$. Hence the plotted value is proportional to the execution time per key comparison. This scaling was chosen to expose cache effects which are now the main source of variation in the y-value. (See also Sections 4.1 and 4.3.)

Another consideration is the range of y-values displayed. Assume $y_{\min} > 0$ is the minimal value observed and $y_{\max}$ is the maximal value observed. Then one will usually choose $[y_{\min}, y_{\max}]$ or (better) a somewhat larger interval as the displayed range. In this case, one should be careful however with overinterpreting the resulting picture. A change of the y-value by 1 % will look equal to a change of y-value of 400 %. If one wants to support claims such as "for large x

the improvements due to the new algorithm become very large" using a graph, choosing the range $[0, y_{\max}]$ can be a more sound choice. (At least if $y_{\max}/y_{\min}$ is not too close to one. Some of the space "wasted" this way can often be used for placing curve labels.) In Figure 2, using $y_{\min} = 1$ is appropriate since no request can get an access delay below one in the model used.

The choice of the the maximum y value displayed can also be nontrivial. In particular, it may be appropriate to clip extreme values if they correspond to measurement points which are clearly useless in practice. For example, in Figure 2 it is not very interesting to see the entire curve for the algorithm "nonredundant" since it is clearly outclassed for large $1/\epsilon$ anyway and since we have a good theoretical understanding of this particular curve.

A further degree of freedom is the vertical size of the graph. This parameter can be used to achieve the above goals and the rule of "banking to 45°": The weighted average of the slants of the line segments in the figure should be about 45°.[2] Refer to [5] for a detailed discussion. The weight of a segment is the x-interval bridged. There is good empirical and mathematical evidence that graphs using this rule make changes in slope most easily visible.

If banking to 45° does not yield a clear insight regarding the graph size, a good rule of thumb is to make the graph a bit wider than high [17]. A traditional choice is to use the golden ratio, i.e., a graph that is 1.62 times wider than high.

4.3 Arranging Multiple Curves

An important feature of two-dimensional graphs is that we can place several curves in a single graph as in Figures 1, 2, and 3. In this way we can obtain a high information density without the disadvantages of three-dimensional plots. However, one can easily overdo it resulting in a chaos of undecipherable points and lines. How many curves fit into one pictures depends on the information density. When curves are very smooth, and have few points where they cross each other, as in Figure 2, up to seven curves may fit in one figure. If curves are very complicated, even three curves may be too much. Often one will start with a straight-forward graph that turns out to be too ugly for publication. Then one can use a number of techniques to improve it:

- Remove unnecessary curves. For example, Figure 2 from [13] compares only eight algorithms out of eleven studied in this paper. The remaining three are clearly outclassed or equivalent to other algorithms for the measurement considered.
- If several curves are too close together in an important range of x-values, consider using another y range or scale. If the small differences persist and are important, consider to use a separate graph with a magnification. For example, in Figure 2 the four fastest algorithms were put into a separate plot to show the differences between them.

[2] This is one of the few things described here which are are not easy to do with gnuplot. But even keeping the principle of banking to 45° in mind is helpful.

- Check whether several curves can be combined into one curve. For example, assume we want to compare a new improved algorithm with several inferior old algorithms for input sizes on the x-axis. Then it might be sufficient to plot the speedup of the new algorithm over the best of the old algorithms; perhaps labeling the sections of the speedup curve so that the best of the old algorithms can be identified for all x-values. Figure 1 gives an example where the speeup of one algorithm over two other algorithms is shown.
- Decrease noise in the data as described in Section 4.6.
- Once noise is small, replace error bars with specifications of the accuracy in the caption as in Figure 6.
- Connect points belonging to the same curves using straight lines.
- Choose different point styles and line styles for different curves.
- Arrange labels explaining point and line styles in the "same order"[3] as they appear in the graph. Sometimes one can also place the labels directly at the curves. But even then the labels should not obscure the curves. Unfortunately, gnuplot does not have this feature so that we could not use it in this paper.
- Choose the x-range and the density of x-values appropriately.

Sometimes we need so many curves that they cannot fit into one figure. For example, when the cross-product of several parameter ranges defines the set of curves needed. Then we may finally decide to use several figures. In this case, the same y-ranges should usually be chosen so that the results remain comparable. Also one should choose the same point styles and line styles for related curves in different figures, e.g., for curves belonging to the same algorithm as for the "shortest queue" algorithm in Figure 2. Note that tools such as gnuplot cannot do that automatically.

The explanations of point and line styles should avoid cryptic abbreviations whenever possible and at the same time avoid overlapping the curves. Both requirements can be reconciled by placing the explanations appropriately. For example, in computer science, curves often go from the lower left corner to the upper right corner. In that case, the best place for the definition of line and point styles is the upper left corner.

4.4 Arranging Instances

If measurements like execution time for a small set of problem instances are to be displayed, a bar chart is an appropriate tool. If other parameters such as the algorithm used, or the time consumed by different parts of the algorithm should be differentiated, the bars can be augmented to encode this. For example, several bars can be stacked in depth using three-dimensional effects or different pieces of a bar can get different shadings.[4]

[3] For example, one could use the order of the y-values at the larges x-value as in Figure 3.

[4] Sophisticated fill styles give us additional opportunities for diversification but Tufte notes that they are often too distracting [17].

If there are so many instances that bar charts consume too much space, a *scatter plot* can be useful. The x-axis stands for a parameter like problem size and we plot one point for every problem instance. Figure 4 gives a simple example. Point styles and colors can be used to differentiate different types of instances or variations of other parameters such as the algorithm used. Sometimes these points are falsely connected by lines. This should be avoided. It not only looks confusing but also wrongly suggests a relation between the data points that does not exist.

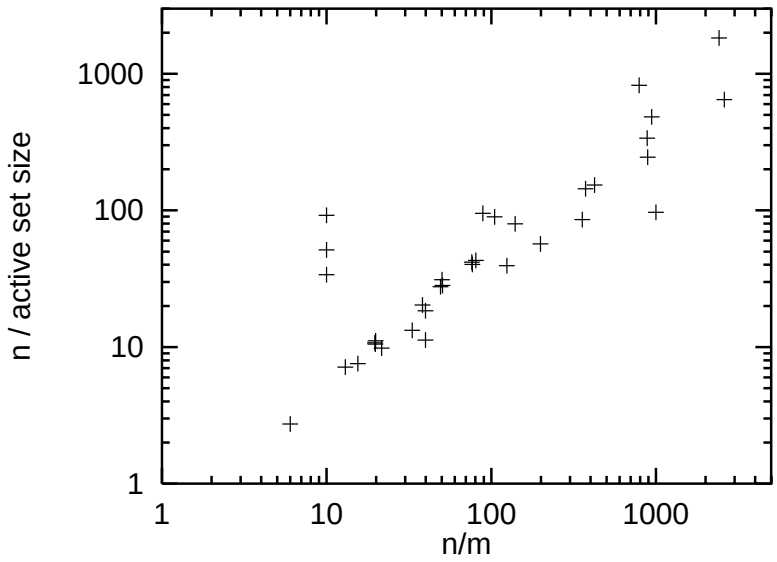

Fig. 4. Each point gives the ratio between total problem size and "core" problem size in a fast algorithm for solving set covering problems from air line crew scheduling [1]. The larger this ratio, the larger the possible speedup for a new algorithm. The x-axis is the ratio between the number of variables and number of constraints. This scale was chosen to show that there is a correlation between these two ratios that is helpful in understanding when the new algorithm is particularly useful. The deviating points at $n/m = 10$ are artificial problems rather different from typical crew scheduling problems. (See also Section 4.1.)

4.5 How to Connect Measurements

Tools such as gnuplot allow us to associate a measured value with a symbol like a cross or a star that clearly specifies that point and encodes some additional information about the measurement. For example, one will usually choose one point symbol for each displayed curve. Additionally, points belonging to the same curve can be connected by a straight line. Such lines should usually not be viewed as a claim that they present a good interpolation of the curve but just

as a visual aid to find points that belong together. In this case, it is important that the points are large enough to stand out against the connecting lines. An alternative is to plot measurements points plus curves stemming from an analytic model as in Figure 5.

The situation is different if only lines and no points are plotted as in Figure 1. In this case, it is often impossible to tell which points have been measured. Hence such a lines-only plot implies the very strong claim that the points where we measured are irrelevant and the plotted curve is an accurate representation of the true behavior for the entire x-range. This only makes sense if very dense measurements have been performed and they indeed form a smooth line. Sometimes one sees smooth lines that are weighted averages over a neighborhood in the x-coordinates. Then one often uses very small points for the actual measurements that form a cloud around this curve.

A related approach is connecting measured points with interpolated curves such as splines which are more smooth than lines. Such curves should only be used if we actually conjecture that the interpolation used is close to the truth.

4.6 Measurement Errors

Tools allow us to generalize measured points to ranges which are usually a point plus an error bar specifying positive and negative deviations from the y-value.[5] The main question from the point of view of designing graphs is what kind of deviations should be displayed or how one can avoid the necessity for error bars entirely.

Let us start with the well behaved case that we are simulating a randomized algorithm or work with randomly generated problem instances. In this case, the results from repeated runs are independent identically distributed random variables. In this case, powerful methods from statistics can be invoked. For example, the point itself may be the average of the measured values and the error bar could be the standard deviation or the standard error [11]. Figure 5 gives an example. Note that the latter less well known quantity is a better estimate for the difference between the average and the actual mean. By monitoring the standard error during the simulation, we can even repeat the measurement sufficiently often so that this error measure is below some prespecified value. In this case, no error bars are needed and it suffices to state the bound on the error in the caption of the graph. Figure 6 gives an example.

The situation is more complicated for measurements of actual running times of deterministic algorithms, since this involves errors which are not of a statistical nature. Rather, the errors can stem from hidden variables such as operating system interrupts, which we cannot fully control. In this case, points and error bars based on order statistics might be more robust. For example, the y value could be the median of the measured values and the error bar could define the minimum and the maximum value measured or values exceeded in less than 5

[5] Uncertainties in both x and y-values can also be specified but this case seems to be rare in Algorithmics.

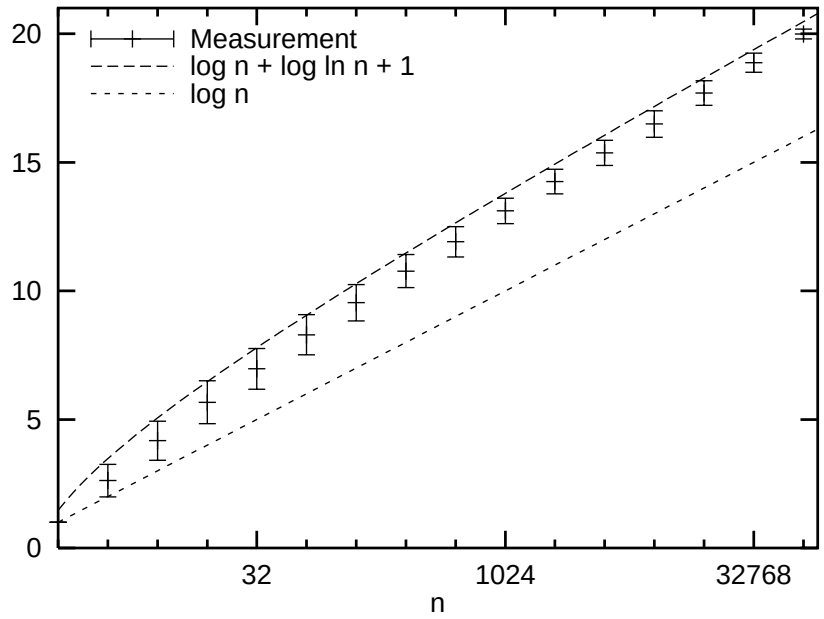

Fig. 5. Number of iterations that the dynamic load balancing algorithm *random polling* spends in its warmup phase until all processors are busy. Hypothesized upper bound, lower bound and measured averages with standard deviation [12, 14]. (See also Sections 4.1 and 4.5.)

% of the measurements. The caption should explain how many measurements have been performed.

5 Grids and Ticks

Tools for drawing graphs give us a lot of control over how axes are decorated with numbers, tick marks and grid lines. The general rule that is often achieved automatically is to use a few round numbers on each axis and perhaps additional tick marks without numbers. The density of these numbers should not be too high. Not only should they appear well separated but they also should be far from dominating the visual appearance of the graph. When a very large range of values is displayed, we sometimes have to force the system to use exponential notation on a part of the axis before numbers get too long. Figure 6 gives an example for the particularly important case of base two scales. Sometimes we may decide that reading off values is so important in a particular graph that grid lines should be added, i.e., horizontal and vertical lines that span the entire range of the graph. Care must be taken that such grid lines to not dilute the visual impression of the data points. Hence, grid lines should be avoided or at

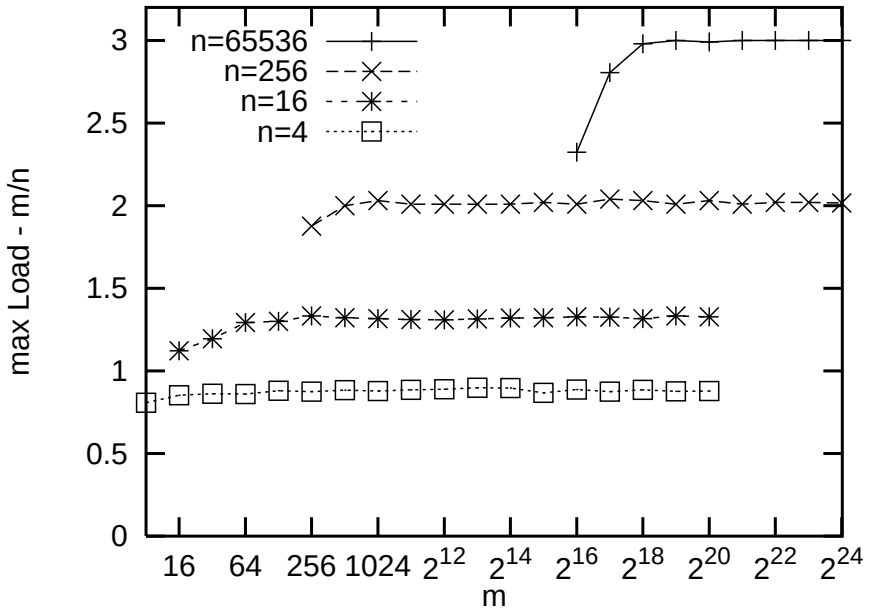

Fig. 6. m Balls are placed into n bins using *balanced random allocation* [2, 3]. The difference between maximal and average load is plotted for different values of m and n. The experiments have been repeated at least sufficiently often to reduce the *standard error* ($\sigma/\sqrt{\text{repetitions}}$ [11]) below one percent. In order to minimize artifacts of the random number generator, we have used a generator with good reputation and very long period ($2^{19937}-1$) [7]. In addition, some experiments were repeated with the Unix generator `srand48` leading to almost identical results. (See also Section 4.3.)

least made thin or, even better, light gray. Sometimes grid lines can be avoided by plotting the values corresponding to some particularly important data points also on the axes.

A principle behind many of the above considerations is called Data-Ink Maximization by Tufte [17]. In particular, one should reduce non-data ink and redundant data ink from the graph. The ratio of data ink to total ink used should be close to one. This principle also explains more obvious sins like pseudo-3D bar charts, complex fill styles, etc.

6 Three-dimensional Figures

On the first glance, three-dimensional figures are attractive because they look sophisticated and promise to present large amounts of data in a compact way. However there are many drawbacks.

- It is almost impossible to read absolute values from the two-dimensional projection of a function.
- In complicated functions interesting parts may be hidden from view.
- If several functions are to be compared, one is tempted to use a corresponding number of three-dimensional figures. But in this case, it is more difficult to interpret differences than in two-dimensional figures with cross-sections of all the functions.

It seems that three-dimensional figures only make sense if we want to present the general shape of a single function. Perhaps three-dimensional figures become more interesting using advanced interactive media where the user is free to choose viewpoints, read off precise values, view subsets of curves, etc.

7 The Caption

Graphs are usually put into "floating figures" which are placed by the text formatter so that page breaks are taken into account. These figures have a caption text at their bottom which makes the figure sufficiently self contained. The captions explains *what* is displayed and *how* the measurements have been obtained. This includes the instances measured, the algorithms and their parameters used, and, if relevant the system configuration (hardware, compiler, ...). One should keep in mind that experiments in a scientific paper should be reproducible, i.e., the information available should suffice to repeat a similar experiment with similar results. Since the caption should not become too long it usually contains explicit or implicit references to surrounding text, literature or web resources.

8 A Check List

In the following we summarize the rules discussed above. This list has the additional beneficial effect to serve as a check list one can refer to for preparing graphs and for teaching. The Section numbers containing a more detailed discussion are appended in brackets. The order of the rules has been chosen so that in most cases they can be applied in the order given.

- Should the experimental setup from the exploratory phase be redesigned to increase conciseness or accuracy? (2)
- What parameters should be varied? What variables should be measured? How are parameters chosen that cannot be varied? (2)
- Can tables be converted into curves, bar charts, scatter plots or any other useful graphics? (3, 4.4)
- Should tables be added in an appendix or on a web page? (3)
- Should a 3D-plot be replaced by collections of 2D-curves? (6)
- Can we reduce the number of curves to be displayed? (4.3)
- How many figures are needed? (4.3)
- Scale the x-axis to make y-values independent of some parameters? (4.1)
- Should the x-axis have a logarithmic scale? If so, do the x-values used for measuring have the same basis as the tick marks? (4.1)
- Should the x-axis be transformed to magnify interesting subranges? (4.1)
- Is the range of x-values adequate? (4.1)
- Do we have measurements for the right x-values, i.e., nowhere too dense or too sparse? (4.1)
- Should the y-axis be transformed to make the interesting part of the data more visible? (4.2)
- Should the y-axis have a logarithmic scale? (4.2)
- Is it be misleading to start the y-range at the smallest measured value? (4.2)
- Clip the range of y-values to exclude useless parts of curves? (4.2)
- Can we use banking to 45°? (4.2)
- Are all curves sufficiently well separated? (4.3)
- Can noise be reduced using more accurate measurements? (4.3)
- Are error bars needed? If so, what should they indicate? Remember that measurement errors are usually *not* random variables. (4.6, 4.3)
- Use points to indicate for which x-values actual data is available. (4.5)
- Connect points belonging to the same curve. (4.3,4.5)
- Only use splines for connecting points if interpolation is sensible. (4.3,4.5)
- Do not connect points belonging to unrelated problem instances. (4.5)
- Use different point and line styles for different curves. (4.3)
- Use the same styles for corresponding curves in different graphs. (4.3)
- Place labels defining point and line styles in the right order and without concealing the curves. (4.3)
- Captions should make figures self contained. (7)
- Give enough information to make experiments reproducible. (7)

References

1. P. Alefragis, P. Sanders, T. Takkula, and D. Wedelin. Parallel integer optimization for crew scheduling. *Annals of Operations Research*, 99(1):141–166, 2000.
2. Y. Azar, A. Z. Broder, A. R. Karlin, and Eli Upfal. Balanced allocations. *SIAM Journal on Computing*, 29(1):180–200, February 2000.
3. 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.
4. J. M. Chambers, W. S. Cleveland, b. Kleiner, and P. A. Tukey. *Graphical Methods for Data Analysis*. Duxbury Press, Boston, 1983.
5. W. S. Cleveland. *Elements of Graphing Data*. Wadsworth, Monterey, Ca, 2nd edition, 1994.
6. D. S. Johnson. A theoretician's guide to the experimental analysis of algorithms. In M. Goldwasser, D. S. Johnson, and C. C. McGeoch, editors, *Proceedings of the 5th and 6th DIMACS Implementation Challenges*. American Mathematical Society, 2002.
7. 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`.
8. 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.
9. C.C. McGeoch and B. M. E. Moret. How to present a paper on experimental work with algorithms. *SIGACT News*, 30(4):85–90, 1999.
10. B. M. E. Moret. Towards a discipline of experimental algorithmics. In *5th DIMACS Challenge*, DIMACS Monograph Series, 2000. to appear.
11. W. H. Press, S.A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 2nd edition, 1992.
12. P. Sanders. *Lastverteilungsalgorithmen für parallele Tiefensuche*. Number 463 in Fortschrittsberichte, Reihe 10. VDI Verlag, 1997.
13. P. Sanders. Asynchronous scheduling of redundant disk arrays. In *12th ACM Symposium on Parallel Algorithms and Architectures*, pages 89–98, 2000.
14. P. Sanders and R. Fleischer. Asymptotic complexity from experiments? a case study for randomized algorithms. In *Workshop on Algorithm Engineering*, number 1982 in LNCS, pages 135–146, 2000.
15. P. Sanders and J. Sibeyn. A bandwidth latency tradeoff for broadcast and reduction. In *6th Euro-Par*, number 1900 in LNCS, pages 918–926, 2000.
16. Peter Sanders. Fast priority queues for cached memory. *ACM Journal of Experimental Algorithmics*, 5, 2000.
17. Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, Connecticut, U.S.A., 1983.