The use of computer algebra systems in computer science education

Article · January 2003

CITATIONS
2

READS
59

2 authors:

Andreas Koeller
Google Inc.
31 PUBLICATIONS **373** CITATIONS
SEE PROFILE

Mark Hubey
Montclair State University
17 PUBLICATIONS **61** CITATIONS
SEE PROFILE

THE USE OF COMPUTER ALGEBRA SYSTEMS IN COMPUTER SCIENCE EDUCATION*

Andreas Koeller, H. Mark Hubey
Department of Computer Science
Montclair State University
Upper Montclair, NJ 07043
Phone: 973-655-7975
Email: {koellera|hubeym}@mail.montclair.edu

ABSTRACT

Every student of Computer Science will encounter certain very important and fundamental data structures and algorithms. Many of those structures are relatively simple to understand conceptually, but exceedingly difficult to analyze, especially when average behavior is taken into account. Therefore, if students are to choose the best algorithm for a problem or optimize the parameters of an algorithm, they are faced with mathematical challenges beyond their means. We suggest that Computer Algebra tools can make a dramatic difference in the ability of students to analyze very common algorithms and data structures and make meaningful design decisions when implementing such structures. Initial studies in the classroom support our suggestion.

1 INTRODUCTION

Students of Computer Science are usually expected to have a working knowledge of software development by the time they graduate from their undergraduate program. It is important to remember that such knowledge does not only include mastering the principles of software engineering and the syntax of one or several programming languages, but also a basic understanding of the behavior of algorithms. Many fundamental algorithms of Computer Science show dramatically fluctuating runtime behavior when some of their parameters are changed. In

---

order to write good algorithms, students must develop during their studies both an intuition of what a good algorithm is and the ability to analyze an algorithm with regard to its runtime. The latter task in particular has traditionally required a deep understanding of concepts of algebra, statistics, and calculus.

2 BACKGROUND

Students in undergraduate Computer Science programs often struggle with advanced mathematical concepts. While simple algebraic topics, like the worst-case analysis of lists and trees, are often taught successfully, students usually do not have the tools available to perform more complex tasks, like average-case analyses of data structures and algorithms. A relatively simple algorithm like hashing leads to fairly complex analytical results when parameters like the average number of overflows are to be computed.

However, the analysis of such algorithms and data structures is quite beneficial to the student's understanding of the applicability and relative advantages of different algorithmic solutions to a problem. Thus, a discrepancy exists between the level of mathematical skills that is feasible to teach to students in an undergraduate curriculum, and the level of skills necessary for them to perform meaningful analyses of the data structures the will deal with on a regular basis in their careers.

3 MATHEMATICAL REASONING USING COMPUTER ALGEBRA SOFTWARE

With the advent of computer algebra software in the early 1990s it became possible to solve increasingly complex algebraic problems in an interactive fashion. Recent version of the most popular software products contain a wealth of features and capabilities for symbolic algebra, making such software a potential tool for the computer programmer who faces the task of analyzing an algorithm.

We argue in this paper that a certain part of the mathematical, and in particular algebraic, education in the undergraduate Computer Science curriculum should consist of teaching the appropriate use of computer algebra software. We claim that this approach will enable students to use a level of algebraic reasoning that will actually be useful to them in the daily work, by going much beyond the "toy examples" they are usually taught in non-major Mathematics courses.

At our department, we have performed a case study in which students were exposed to the computer algebra system *Maple™* and were encouraged to use *Maple™* in order to derive meaningful and easy to interpret results for the analysis of fairly complex problems.

4 A CASE STUDY IN THE USE OF *MAPLE™*

In a junior-level course on File Processing which included topics like File Structures and File Operations, students were introduced to the topic of *hashing* in all its varieties. The basic hashing algorithm, which is a fairly simple and elegant idea, was presented and explained by way

of examples. Subsequently, students were introduced to the analysis of the algorithm. While the analysis of the length of the average search is often still accessible using traditional mathematical tools, the analysis of collisions clearly is not.

4.1 ANALYSIS OF THE NUMBER OF OVERFLOW RECORDS IN HASHING

The analysis of the number of overflow records is helpful to estimate the degradation of performance of a hash table under insertions. What we need to compute is the likelihood that k records will hash to the same address in the hash table. An understanding of such hash table overflow will enable a programmer (student) to judge the size of the hash table necessary for a particular application as well as the applicability of the hash table concept in general. That is, it is quite important for students to get an intuitive understanding of such issues.

However, the formal analysis is quite complex. We assume a hash table with *N* addresses, and try to analyze the storage of *r* records, each of which is randomly hashed to an address. A *Bernoulli* process analysis [1] can be performed, with $p = \frac{1}{N}$ and $q = 1 - \frac{1}{N}$, yielding the following equation for the probability that *x* records hash to the same address:

$$P_{r,x} = \frac{r!}{x!(r-x)!} p^x q^{r-x} = \frac{r!}{x!(r-x)!} \left(\frac{1}{N}\right)^x \left(1 - \frac{1}{N}\right)^{r-x} \quad (1)$$

Since we observe that the probability of a success in this process at $p = \frac{1}{N}$ is very small, we can approximate this value by a Poisson approximation [3], as follows:

$$P_{r,x} \approx f_{r,N}(x) = \frac{\left(\frac{r}{N}\right)^x}{x!} e^{-\frac{r}{N}} \quad (2)$$

We call this quantity $f_{r,N(x)}$. The ratio *r/N* is the *load factor* of the hash table.

In order to compute the expected overflow, we observe that we obtain an overflow of *x* if *x*+1 values hash to the same address. Note that the analysis is correct only for chained hashing, since in open-address hashing the overflowing records increase the probability of collisions. Now, the expected overflow when inserting *r* records into a hash table of N addresses can be computed as follows:

$$E(r,N) = N\sum_{x=2}^{\infty}(x-1)f_{r,N}(x) = N\sum_{x=2}^{\infty}(x-1)\frac{\left(\frac{r}{N}\right)^{x}}{x!}e^{-\frac{r}{N}} \quad (3)$$

While these observations are not necessarily difficult to convey to students, the interpretation of the results gets quite complicated. An expression like Eqn. 3 is unlikely to be comprehensible to a student.

4.2 USING *Maple™* TO ANALYZE ALGORITHMS

The use of the software *Maple™* in its GUI version is not very difficult for students to understand. In order to accomplish the task of helping students to analyze algorithms, only very few syntax elements of the software need to be taught. In particular, there are three capabilities of algebra software that students need to use:

- Students must be able to enter simple mathematical expressions into a *Maple™* worksheet,
- students must be able to execute a small number of operations: they need to compute sums and possibly integrals, derivatives, find the zero(es) of expressions, and simplify them, and
- students must be able to visualize their results in a meaningful manner.

While there are text-mode versions of *Maple™*, such software is hard to use for students who are not well experienced in the use of UNIX-based command-line tools. Therefore, GUI-based interfaces (e.g., `xMaple`) should be used as the basis for such teaching.

4.2.1 SIMPLE MATHEMATICAL EXPRESSIONS

The writing of simple mathematical expressions should not pose a problem to students who are experienced in at least one major procedural programming language. Therefore, it can be assumed that students are familiar with the syntax of such expressions.

4.2.2 OPERATIONS

Sums in *Maple™* are computed using the syntax:

```
sum(f(x),x=a..b)
```

where `f(x)` is a mathematical expression, and a and b, the limits of the sum, are themselves expressions. In particular, *Maple™* allows to use the special values `infinity` and `-infinity` in the place of `a` and `b`, such that values of infinite sums are easy to compute.

Integrals are computed in much the same way, for example $\int_0^{\infty} \frac{1}{e^x} dx$ is entered in *Maple™* as

```
int(1/exp(x),x=0..infinity);
```

and correctly yields `1` as result.

Derivatives like $\frac{\partial}{\partial x}\sin(x)$ are entered as `diff(sin(x),x)`, while finding the zero(es) for an expression *f(x)* is entered as `solve(f(x),x)` or simply `f(x)` if there is only one variable.

While *Maple™* usually simplifies expressions quite nicely, it sometimes prints expressions that obviously can simplified further. Due to that reason, it is also necessary for students to know about the possibility of explicitly simplifying expressions using the `simplify()` function. An example is given in Sec. 4.3.

4.2.3 PLOTTING GRAPHS

The single most powerful tool for understanding complex quantitative relationships is their visualization as graphs. Not only is it possible to quickly grasp the behavior of a function over its domain, but especially in an interactive system like *Maple™* it becomes easy to manipulate graphs. That allows a user to literally explore the function to be plotted, and thus obtain an intuitive access to the often complex mathematical concepts arising from algorithmic analysis.

Plotting in *Maple™* is quite simple and can be accomplished for functions with one or two real parameters. Especially the latter possibility ("3D-plots") yields very informative graphs that help students to understand algorithms.

Plotting is initiated by a "plot setup", which depends on the operating system that the software is running on and for UNIX-based *Maple™* -versions looks like

```
plotsetup(x11)
```

The plot setup is not absolutely necessary when using `xMaple` but opens a separate window for each function which is helpful in the interactive manipulation of graphs.

The only information necessary for *Maple™* to plot a function is the function definition and the range of its free variables.

For example, the function *f(x)* = sin(*x*) is drawn simply by

```
plot(sin(x),x=-Pi..Pi)
```

while the function $f(x,y) = \sqrt{x^2 + y^2}$ is drawn using

```
plot3d(sqrt(x^2+y^2),x=-1..1,y=-1..1)
```

A useful feature of the `plot3d` function is that the appearing graph can be reformatted and even rotated using the mouse. Also, recoloring and changes of the perspective of the graphs is possible. We have found it most useful that the *Maple™* graph plotter supports *Z-coloring* which colors the graph according to its value in the z-(output)-dimension. Also, plotting the graph with *boxed* axes is helpful in assessing the value of the function at a given point.

4.3 USING *Maple™* TO IMPROVE THE UNDERSTANDING OF ALGORITHMIC ANALYSES

With all those tools in place, it is now quite straightforward to enable students to access a result like Eqn. 3. Entering the formula[1] immediately gives the simplified result

$$E(r,N) = N \cdot e^{-\frac{r}{N}}\left(1 - e^{\frac{r}{N}}\left(1 - \frac{r}{N}\right)\right) \tag{4}$$

which, however, can be further simplified (using the `simplify` function) to

$$E(r,N) = N \cdot e^{-\frac{r}{N}} - N + r \tag{5}$$

Clearly, the last equation is much more accessible to students than the original result. At this point, a simple plot can be computed, which will display the relationship between the size of the hash table, the number of inserted records, and the number of overflow records.

It might be noted here to students that experimenting with the software might lead to simpler and simpler results. For example, substituting $m = \frac{r}{N}$ in the original formula will enable *Maple™* to produce the result

$$E(m,N) = N \cdot (e^{-m} + m - 1) \tag{6}$$

which, for a given *N*, can be easily plotted as a two-dimensional graph (Fig. 1), visualizing nicely the relationship between load factor and the average number of overflow records per address in the hash table, which approaches $N \cdot 1/e$ for *m*=1 (as one might derive earlier in the course using simple means of probability).

4.4 USING *Maple™* TO DERIVE MORE COMPLEX ANALYTICAL RESULTS

While the results above are still quite accessible to the mathematically trained undergraduate computer science student, more complex analyses can also be successfully performed, and kept plausible to undergraduate students. For example, one might want to present to students the concept of *hash buckets* which allow *b* elements to hash to the same

[1] `As N*sum((x-1)*(r/N)^x/x!*exp(-r/N),x=2..infinity`

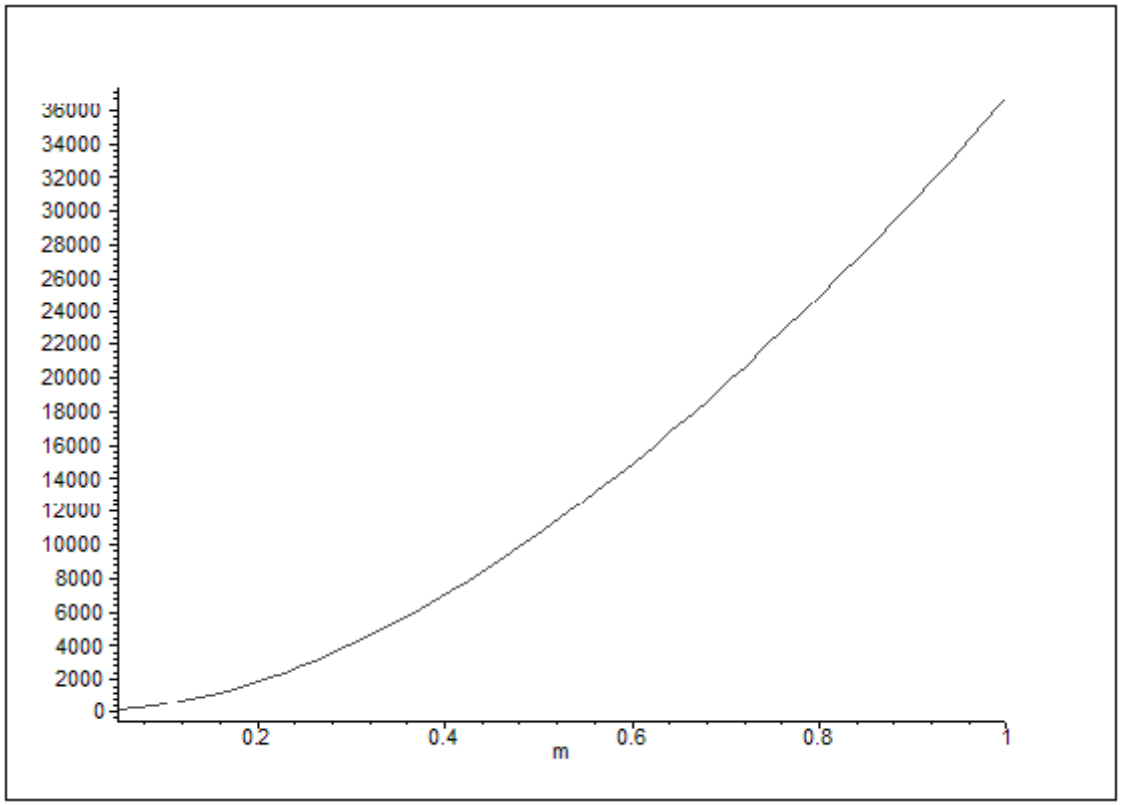

Figure 1: Relationship between Load Factor and Number of Overflow Records for N=100,000

address, for a fixed *b*. The bucket hashing algorithm is more difficult to analyze, since an additional parameter has to be taken into consideration.

Starting from the same original Poisson approximation , Eqn. 2, we observe that the expected overflow is now reduced by the bucket size, that is

$$E(r,N,b) = N \cdot \sum_{x=b+1}^{\infty} (x-b) f_{r,N}(x)$$

$$= N \cdot \sum_{x=b+1}^{\infty} (x-b) \frac{\left(\frac{r}{N}\right)^x}{x!} e^{-\frac{r}{N}} \qquad (7)$$

Solving this expression using *Maple*™ yields the solution (after simplification)

$$E(r,N,b) = \frac{\left(\frac{r}{N}\right)^b r e^{-\frac{r}{N}} M_{2,b+2}\left(\frac{r}{N}\right)}{(b+1)!} \qquad (8)$$

where $M_{m,v}(z)$ is the *Kummer M-function*, which is equivalent to Barnes' extended hypergeometric function $F([m],[v],z)$ [4].

At this point, it is important to note that it is not necessary for students to understand the meaning of such advanced functions of probability theory, since their goal is mainly to derive some insight into the behavior of algorithms at certain input values. While this function will not

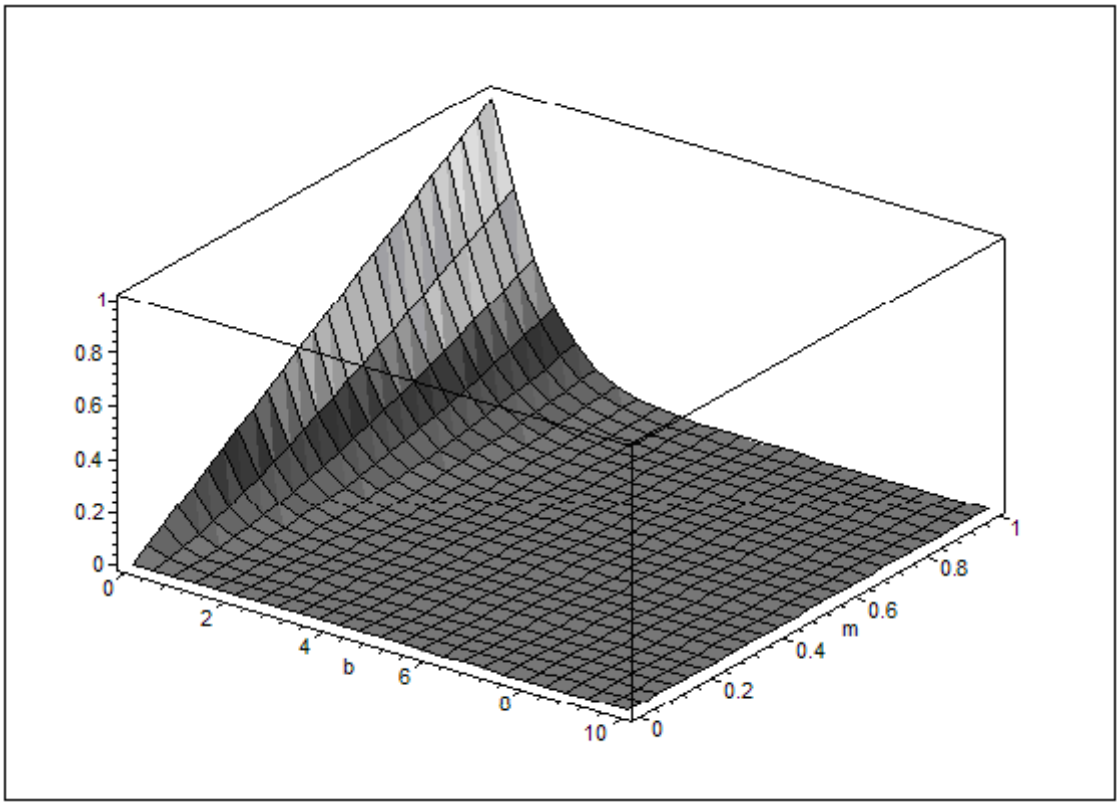

Figure 2: A Maple-drawn Graph for the Overflow Ratio on Bucket Hash Tables

be intuitively comprehensible to students, it is very helpful here to draw a graph. Plotted as a 3D-graph over the dimensions *b* (bucket size) and *r/N* (load factor) for a fixed *N*, one obtains a graph as in Fig. 2

The graph shows very dramatically that having a bucket size of as small as 2 will almost make the problem of overflow disappear. Intuitively understanding such concepts can make a significant difference in students' ability to find appropriate algorithms for the problems they need to solve in their careers.

5 EVALUATION

Maple™ has been used at our department in a number of courses [2]. Initial results of classroom surveys that we have conducted suggest that, while a majority of students had not used tools like *Maple™* before, practically all students surveyed felt that *Maple™* would be a useful tool for the computer scientist. Furthermore, almost 90% of the students answered that

Maple™ or other CAS tools should be used more often in the classroom and that they felt that, given sufficient instruction, they could use such tools successfully in analyzing algorithms.

From our initial studies, we conclude that the following strategy would enhance students' ability to implement good algorithms:

1. Thoroughly train students in the *setup* of algorithmic analysis problems, that is in problem definition, the identification of the parameters affecting an algorithm's runtime and the finding of an algebraic expression to compute the runtime.
2. Enable students to use a Computer Algebra tool, by training them in the use of a small number of concepts of that tool (computing algebraic expressions, graphing such expressions as functions, differentiating expressions and finding zeroes of functions).
3. Use Computer Algebra software in the classroom whenever appropriate, to teach students how to "avoid" difficult mathematical analyses while still solving the problems of algorithmic optimization.

6 CONCLUSION

In this paper, we approach the problem that even simple algorithms and data structures that every student of Computer Science will encounter in their studies are difficult to analyze. If students are to choose the best algorithm for a problem or optimize the parameters of an algorithm, they are faced with mathematical challenges beyond their means. We suggest that Computer Algebra tools like *Maple™* or *Mathematica* can make a dramatic difference in the ability of students to analyze very common algorithms and data structures and make meaningful design decisions when implementing such structures. Initial studies in the classroom support our suggestion.

ACKNOWLEDGMENTS: We would like to thank our anonymous reviewers for valuable comments on the paper.

REFERENCES

[1] Bernard W. Lindgren. *Statistical Theory*. Macmillan Publishing Co., Inc., New York, 3rd edition, 1976.

[2] H. Mark Hubey. *File Processing. Lecture Notes*. Montclair State University, Upper Montclair, NJ, 2000.

[3] John A. Rice. *Mathematical Statistics and Data Analysis*. Duxbury Press, 2nd edition, 1995.

[4] Waterloo Maple Inc. *Maple™ V Release 5 Online Manual*, 1981-1997.