The Interview Column

by

Chen Avin and Stefan Schmid

Ben Gurion University, Israel and TU Berlin, Germany
{chenavin,schmiste}@gmail.com

KNOW THE PERSON BEHIND THE PAPERS

Today: Don Knuth

Bio: *Don Knuth is Professor Emeritus at Stanford University and the author of the landmark series* The Art of Computer Programming. *His contributions to algorithms, analysis, formal languages, TeX, literate programming, and combinatorics have shaped modern computing. He is a Turing Award laureate and continues to be an active researcher, writer, and musician.*

Figure 1: Photo credits: Rajan P. Parrikar

Stefan Schmid: Is there a photo you like to share with us, for the interview column?

Don Knuth: You can just pick something from my homepage, there's actually a whole collection of "downloadable graphics" with more or less unusual photos of me. Maybe you can take a picture that shows me in a natural setting, perhaps near an organ or a piano, which are important parts of my life.

Stefan Schmid: Can you please tell us something about you that most of the readers of your papers probably don't know?

Don Knuth: One thing is my lifelong connection to music. I started playing the organ when I was 12 years old — my father had a piano at home because he was teaching students. That led to a deep love of organ music. I eventually wrote a large organ composition, which was performed on my 80th birthday. It mixes about 18 different musical styles; I knew it would probably be the only big piece I ever wrote, so I decided not to hold back. These days I'm working my way, on the piano, through Broadway scores in alphabetical order. Recently I played through *Carousel*, then *Damn Yankees*, and most recently *Fiddler on the Roof*. I also helped pay for the organ at the church I attend, using money from the Kyoto Prize, and I have an organ at home. Music is still a big part of my daily life.

Stefan Schmid: Can you tell us the story behind one of your papers? For example one that took very long or one which evolved in an unexpected way?

Don Knuth: A nice example goes back to my first paper in the *Journal of the ACM*. It was about how to place instructions on a rotating drum so as to minimize latency. Depending on where instructions were placed, you could lose on the order of 50 milliseconds if they were in the "wrong" place. I set up a small example as an integer programming problem, and I think I was perhaps the first person to implement Ralph Gomory's algorithm for integer programming — he told me so, at least. Unfortunately, our computer at the time was extremely small and slow, so the problem was hard to solve. It was finally cracked only decades later, and I recently saw a paper about "Knuth's old latency problem," now solved very quickly by modern IP solvers. So that's a paper whose story stretches from drum memories in the early 1960s all the way to 21st-century optimization software.

Another kind of "paper that got out of hand" is not a paper but the sections in *The Art of Computer Programming* on BDDs, SAT solvers, and exact cover with colors (XCC). None of these topics existed when I drafted the table of contents in the 1960s, but they turned out to be so important that, for example, the SAT-solver section alone ended up with almost 500 exercises.

Stefan Schmid: When (or where) is your most productive working time (or place)?

Don Knuth: Historically, my most productive period was probably around 1966, when I was at Caltech and working both on algorithms and on Volume 2 of *The Art of Computer Programming*. I would wake up in the morning saying to myself, "Here's what I'm going to finish today," and then work until it was finished, often late at night. I had two young children at the time, and I did a lot of work at home with the television on in the background. It was highly productive in terms of "algorithms per hour," but it was also completely unsustainable: I ended up in the

hospital. You can actually locate exactly where I was in Volume 2 at that moment — there is a sneaky index entry under "brute force" pointing to a Latin phrase on the page I was working on.

Nowadays I still work mostly at home. I usually swim in the morning and, while swimming, think about what I'm going to write later. Then I come back and spend the rest of the day writing, programming, or debugging. Wednesdays are partly reserved for gardening and laundry.

Stefan Schmid: How do you choose what to work on? And what kind of impact are you hoping your work will have? Did this change over the course of your career?

Don Knuth: In 1962 I wrote down a tentative table of contents for what became *The Art of Computer Programming*. That list, in a sense, has defined my research agenda ever since. I simply move through it steadily: last week I might have been on one page, this week on the next. Of course, the contents have evolved — new topics like BDDs, SAT solvers, and parameterized complexity have reshaped some chapters — but the overall roadmap is still there.

As for impact, when I write about a topic I try to ask myself, "Will this still be interesting 50 years from now?" I'm not very good at predicting the future, but I do my best to choose subjects that will remain valuable in the long run. My goal has always been to provide a deep, reliable foundation that future generations can build on.

Stefan Schmid: What do you do when you get stuck with a research problem? How do you deal with failures?

Don Knuth: Sometimes I write to colleagues or talk to people at lunch at Stanford, explaining exactly where I'm stuck. In other cases I simply put the problem aside and hope that, months or years later, I'll return with a fresh perspective. Only rarely — maybe two or three times in my whole life — has a solution come to me in the middle of the night. More often, I write a program to get more facts about the problem, and the very process of programming helps clarify my thoughts in some magical way. Of course failures greatly outnumber successes.

A concrete example: I recently spent four hours trying to understand an early survey paper on what we now call treewidth. The quantifiers ("for some," "for every") were used in such an ambiguous way that I simply couldn't tell what was really meant. In the end I pencilled a note on the paper saying, "I can't make head nor tail of this," dated it, and put it away — with a few clues I had extracted in case I ever return. Then I went back to working on Hamiltonian cycles, where I could actually make progress.

Stefan Schmid: Do you have any advice for young researchers? In what should they invest time, what should they avoid?

Don Knuth: My main advice is: Try to work on things that are genuinely interesting and somewhat unique to you, rather than on topics that are "hot" because the community says so. Everyone has a different background and set of experiences, which means each person is uniquely qualified to solve certain problems. You usually do not know in advance which problems those are, so it's important to get to know your own strengths and weaknesses.

I also wish there were more emphasis on algorithms that actually compute things in practice, not just in asymptotic theory. There are many papers shaving off a factor like the cube root of $\log\log n$ when $\log\log n$ is never more than five. Such results can be interesting because they expose bottlenecks, but too often we stop after proving polynomial vs exponential time and forget that constants matter enormously.

So: Invest time in problems where you can really understand the phenomena deeply, where you can implement and test algorithms, and where your work might remain meaningful decades from now. Prefer a problem that "has your name on it" because of your personal background. Avoid chasing fashions just because they're fashionable.

Stefan Schmid: What research topics or approaches do you wish the TCS community in general would study more?

Don Knuth: I'd like to see more work on what can be *really* computed, not just in principle. Our field has many beautiful complexity results, but it's easy to stop at labels like "polynomial time" or "exponential time" and forget that the constant factors and actual implementability are crucial.

Of course I don't restrict attention entirely to questions of efficient real-world computation. Sometime a purely theoretical question that has a negligible impact on practical programs is of great interest, often because of its historical value or the way it helps us to focus on general techniques that are widely useful also in other contexts. For example, the study of optimum addition chains has always fascinated me: It is one of the earliest fundamental computational problems to have been investigated, and its theory leads to numerous insights — even though I'll never actually want to compute $x^{5784689}$ with the fewest possible multiplications.

I've recently spent a lot of time on exact cover, SAT, and now exact cover with colors (XCC), which have turned out to be tremendously powerful tools for solving concrete combinatorial problems such as graph embedding.

At the same time, I admire developments in parameterized complexity and kernelization, which have enriched our understanding of hard problems like vertex cover. My wish is not to replace one area with another, but to keep an eye on the interplay between elegant theory and robust, implementable algorithms.

Stefan Schmid: How do you approach evaluating research papers? What about

grant proposals?

Don Knuth: I'm usually interested mostly in learning the techniques that led to a solution to a specific problem, much more than in the actual solution itself; those techniques will help me with future problems. Before I turn each page, I try to guess what will be on that page. I reformulate the problem in other notations, and I try to solve it myself before reading on to see what the author has actually done. Of course I usually fail; but this exercise has prepared me to learn from my failures, and to appreciate what the author has achieved.

I am especially skeptical of proofs that consist of hundreds or thousands of unverified cases in an appendix or technical report. In one area of graph classes, for example, I found typographical errors in the cases I checked and no code to back up the claims, which made the results unconvincing to me.

So I value papers where the authors either provide implementations or at least make it reasonably clear how their algorithms would be programmed and tested. The same general criteria would apply if I were evaluating grant proposals.

Stefan Schmid: Do you see a main challenge or opportunity for theoretical computer scientists for the near future?

Don Knuth: One continuing challenge is to maintain high standards of rigor while engaging seriously with real-world computational problems. We have powerful tools — SAT solvers, IP solvers, XCC solvers, parameterized algorithms, and many others — and there are huge opportunities in understanding when each tool is appropriate and how they complement one another.

But in general, I'm not good at predicting the future; in fact, there are many developments in computer science that surprised me, and hardly any I foresaw. My own focus remains on topics where I feel I can contribute durable insights.

Stefan Schmid: Is there a blog, podcast, book, or movie you like particularly?

Don Knuth: I haven't had a television set for about 40 years, and I don't follow blogs or podcasts; I honestly don't know where people find the time. Reading the newspaper already takes more time than it should and tends to depress me.

However, my homepages include a page in the FAQ section where I list about forty books that I've especially enjoyed — mostly not about mathematics. One mathematical book I particularly like is Peter Winkler's *Mathematical Puzzles*. A new edition came out recently, and it's brilliant.

Stefan Schmid: Is there a nice anecdote from your career you would like to share with our readers?

Don Knuth: One favorite anecdote involves knight's tours on the chessboard. When I first learned about computers in the 1950s, one of the earliest problems I tried to tackle was: "How many knight's tours are there on an 8×8 board?" I

wrote a little program and quickly realized the search space was enormous, so I gave up.

Then, in the 1990s, Ingo Wegener and a coauthor used clever methods and modern computers to compute what they believed was the exact number, around 33 billion. When I saw their paper, I immediately noticed that their number was not divisible by 4, and it's not hard to prove that the count must be a multiple of 4. So I knew something was wrong.

I contacted my colleagues, and they eventually computed the correct number and found the error in the earlier calculation. The method itself was basically sound; there had just been a slip. For me this story nicely illustrates several themes: the power of computers and algorithms, the importance of simple invariants (like divisibility by 4), and the way mathematics, computation, and careful checking all interact.

Please complete the following sentences?

- *My first research discovery*
  was an "imaginary" number system based on the square root of -2, which I developed in high school and later published after submitting it to the Westinghouse Science Talent Search.

- *The key to being a happy academic*
  is to pretend that politics doesn't exist.

Stefan Schmid: Don, many thanks for this wonderful conversation. Any final words before we let you go?

Don Knuth: Thank you! I really enjoyed the questions you asked. And now I think I'll go fold the laundry — it's Wednesday, after all.