A Few Lessons I've Learned

Erik D. Demaine
Massachusetts Institute of Technology
edemaine@mit.edu

Abstract

This article summarizes four key lessons I've learned doing research as a theoretical computer scientist. The lessons represent my style and approach to research, and are somewhat nonconventional, centered around fun. This article is based on a talk I gave upon receiving the Presburger Award this year, which can be viewed online.[1]

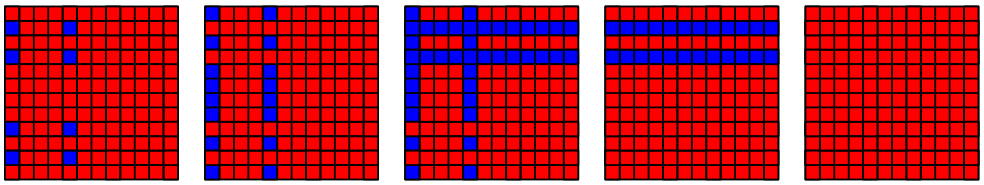
1 Have Fun

We all study theoretical computer science because we enjoy it—the beauty of simple models and algorithms, the certainty of proved theorems, the thrill of solving problems. By working on what we are passionate about, and enjoying what we do, we naturally work harder and are more productive.

I like to embrace "having fun" even further, by studying what I call *recreational computer science* [5] (by analogy to recreational mathematics, an area championed by Martin Gardner). This area is all about studying fun topics, from the perspective of serious theoretical computer science. In this way, we get to play with our work.

For example, a recent paper [7] with my father Martin Demaine, my students Sarah Eisenstat and Andrew Winslow, and my former PhD advisor Anna Lubiw, studied the Rubik's Cube from a theoretical computer science perspective. The Rubik's Cube is probably the most famous puzzle in the world, and its mathematical connections to group theory are well known, but it had not been considered from an algorithmic perspective. The classic $3 \times 3 \times 3$ cube had just been conquered, proving that the worst-case optimal algorithm solves any configuration in 20 moves [13]. But from an algorithmic perspective, the natural question is of scalability: how does the worst-case number of moves to solve an $n \times n \times n$ Cube grow with n? Standard strategies for solving Rubik's Cubes solve $O(1)$ cubies

[1] http://www.youtube.com/watch?v=ROYIVVZ5gvE



(a) Start (b) Cols. flipped (c) Rows flipped (d) Cols. flipped (e) Rows flipped

Figure 1: Simultaneously solving a grid of identically configured cubies in an $n \times n \times 1$ Rubik's Cube. (Figure from [7].) In this example, each cubie in the grid could be solved by flipping its row, then its column, then its row, then its column; but given the shared rows and columns, it is more efficient to solve them all together. The 3D Cube is similar.

in $O(1)$ moves, which suggests an upper bound of $O(n^2)$, the surface area of an $n \times n \times n$ Cube. But the right answer turns out to be $\Theta(n^2/\log n)$—using tricks from data structures, we show you can always kill $\Omega(\log n)$ birds with $O(1)$ stones.

I also love to study the computational complexity of games and puzzles. While solving actual puzzles and playing actual games is fun, the meta-puzzle of figuring out how to solve them algorithmically is even more fun—at least for theoretical computer scientists. In fact, most puzzles are NP-complete or PSPACE-complete, while most 2-player games are PSPACE-complete, EXPTIME-complete, or worse. Proving these results involves constructing portions of puzzles and games ("gadgets") with simple properties, which itself feels like solving a puzzle. The difference is that no one has solved the puzzle before, so it may not have a solution (requiring taking a different approach), but if solved, the result is publishable!

This subject was the thesis topic of my PhD student Bob Hearn, which resulted in a book [12]; see also our survey [9]. Some of my favorite results in the area are that Tetris is NP-complete [2], sliding-block puzzles are PSPACE-complete [11], and Super Mario Bros., Legend of Zelda, and other Nintendo games are all NP-hard [1].

Why study games and puzzles? These results give us a mathematical understanding of what makes games and puzzles challenging, which is probably part of what makes them fun. The results are also accessible to the general public, which helps explain our field and what powerful results it can show about the limitations of computation, and in turn helps attract young students to the field. These problems are also accessible to new researchers: everyone has a favorite game or puzzle, and with relatively little background in the field, can combine their own expertise in how to play with a clear path of searching for gadgets that establish computational complexity. But to me the primary motivation to study these problems is that they are *fun*.

2 Do More Than One Thing

I work primarily in four areas of algorithms: computational geometry (in particular folding [10]), data structures (such as cache-oblivious [4]), recreational algorithms (just described), and graph algorithms and minors (see [8]). I have worked in the first three fields since early on as a graduate student. The recreational algorithms were for fun; the other two areas were inspired by my PhD advisors Anna Lubiw (primarily in computational geometry) and Ian Munro (primarily in data structures). But instead of picking one of the areas to focus on, I chose to focus on all of them.

Why do more than one thing? The primary reason is to avoid getting stuck, which can easily happen on any one problem, for unpredictable lengths. Hedging your bets against many problems helps you smooth out this behavior, and consistently be able to solve something at all times. Beyond just solving problems, each paper or project evolves from problem to solution to write-up to submission to revision and so on. Having multiple projects on the go means that you usually have one at each stage, which lets you adapt what you work on according to your mood. When you're feeling creative, you can brainstorm problems and/or solutions—but when you're not, you can be just as productive by finishing a write-up or revising a paper. And whenever you learn a new technique from reading a paper or watching a talk, you can check whether it sheds new light on any of your problems.

I think everyone has their own capacity to how many problems and fields they can reasonably keep active. But I believe everyone should learn to make it greater than 1 (and perhaps much greater), given the efficiency gain. Plus, it's more of a good thing, so more fun.

A pleasant surprise is when one project you're working on inspires progress on another. This effect is especially cool across different areas. For example, I find geometry offers a useful perspective on traditionally nongeometric problems, like data structures; and in the reverse direction, I find many techniques from data structures are useful in many other areas such as computational geometry (and the Rubik's Cube mentioned above). By knowing both areas, I can solve problems that most others can't.

3 Collaborate

I find collaboration an extremely productive way to do research. I've written papers with 326 people (still a far cry from the famous mathematician Paul Erdős's 511 co-authors). The only papers I've written by myself are surveys, a few papers before I entered theoretical computer science (and discovered collaboration), and this article you're reading.

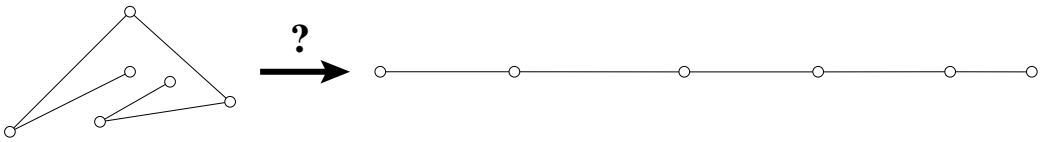


Figure 2: The Carpenter's Rule Problem: can you unfold any polygonal chain while preserving edge lengths and avoiding collisions?

I learned the highly collaborative research model from a series of computational geometry workshops held at the Bellairs Research Institute of McGill University in Holetown, Barbados. These workshops generally have no talks—just lists of open problems to work on, and progress reports on partial results. In the span of one week, a group of 10–30 researchers can solve an impressive number of problems, which lead to several papers. The style of research is constant brainstorming: suggesting ideas, finding counterexamples, drawing pictures, etc. The results that come out typically combine key ideas from many people, and avoid pitfalls observed by many more people, leading to (well-deserved) many-author papers.

My theory of collaboration is that problems decompose into several subproblems, each of which is probably easy for *someone* to solve, but also at least one of which is probably hard for each individual. If you try to solve the whole problem yourself, you'll inevitably get stuck on the subproblem that's hard for you. But if you combine the right mix of collaborators, with a mix of expertise, every subproblem can be easy. In this way, n people can solve a problem much faster than a single person, even working for n times as long.

A personal success for me was the Carpenter's Rule Theorem [3]: any 2D polygonal chain can be continuously unfolded to a straight line while keeping the edge lengths constant and avoiding collisions. Whether this was possible had been open for decades, and it quickly became my favorite open problem, so I carried it wherever I went. Eventually I found the right ideas from the right collaborators. Günter Rote had the counterintuitive idea that the unfolding could simultaneously expand all pairwise vertex distances. Bob Connelly then realized that this put the problem within the domain of his area of expertise, rigidity and tensegrity, so he could quickly identify powerful tools to help answer it. Then together we found a geometric argument that solved the problem, within just a few hours. So ultimately the problem was "easy" with the right collaborators and tools—finding them was the hard part. Without collaboration, the problem might never be solved.

Collaboration also allows us to grow our own toolset by learning each others' tools. What better way to learn a new tool or area than by solving an open problem that needs it, together with an expert who knows it? This is usually a necessary ingredient for interdisciplinary work, which is the next topic.

But let me also mention that collaboration is *fun*. It makes research a social experience. We share ideas over meals, in coffee breaks, and while traveling. These interactions build close personal bonds between researchers that makes the work all the more enjoyable.

4 Cross Disciplines

Interdisciplinary research is quite popular these days, and for good reason. Traditional fields are well-developed, and many of the basic problems have been solved or are known to be difficult. But in between these fields, there are still many new problems to be discovered, with lots of potential for exciting results. I would encourage everyone to explore Theoretical Computer Science + X, for their favorite value of X, especially those that have not yet been explored.

My favorite value of X (at the moment) is art. At first glance, art may seem like a completely different discipline from mathematics. But they are not really so different. Both math and art are all about creativity—about having a clever idea, and then carefully executing that idea within the medium. For mathematics, the medium is proof; for art, it might be sculpture, performance, or purely conceptual. Both math and art have an aesthetic of beauty: mathematicians try not just to prove a theorem, but to find an elegant proof. In this way, mathematics is an art form [6].

I got into art because my father's background is in visual arts, but also because it was helpful for my mathematical research. We built physical models to better understand the geometric structures we were studying, and those models turned out to be nice in their own right. So we decided to pursue the artistic goal of turning them into sculpture.

The most fun for us is when we can pursue the same project from both an artistic and mathematical perspective. This can be a powerful approach, because the two perspectives play off of each other, leading to inspirations on both sides—similar to collaboration or working on more than one thing. Our mathematics inspires new sculptures to build, leading to new art. Our artistic ideas often lead to questions about whether desired structures actually exist, leading to new mathematics. We believe that this makes us more productive, both as artists and as mathematicians, and leads to work on both sides that would not otherwise happen (similar to collaboration). See [6] for some examples.

You can follow these lessons however you see fit, but have fun with it!

Figure 3: Sculpture 0351 by Erik and Martin Demaine, Mi-Teintes watercolor paper, 2013, 8″ × 15″ × 16″ high. For more of our sculpture, see `http://erikdemaine.org/curved/`.

References

[1] Greg Aloupis, Erik D. Demaine, and Alan Guo. Classic Nintendo games are (NP-)hard. arXiv:1203.1895, March 2012.

[2] Ron Breukelaar, Erik D. Demaine, Susan Hohenberger, Hendrik Jan Hoogeboom, Walter A. Kosters, and David Liben-Nowell. Tetris is hard, even to approximate. *International Journal of Computational Geometry and Applications*, 14(1–2):41–68, 2004.

[3] Robert Connelly, Erik D. Demaine, and Günter Rote. Straightening polygonal arcs and convexifying polygonal cycles. *Discrete & Computational Geometry*, 30(2):205–239, September 2003.

[4] Erik D. Demaine. Cache-oblivious algorithms and data structures. In *Lecture Notes from the EEF Summer School on Massive Data Sets*. BRICS, University of Aarhus, Denmark, 2002. `http://erikdemaine.org/papers/BRICS2002/`.

[5] Erik D. Demaine. Recreational computing: Puzzles and tricks from Martin Gardner inspire math and science. *American Scientist*, 98(6):452–456, 2010.

[6] Erik D. Demaine and Martin L. Demaine. Mathematics is art. In *Proceedings of 12th Annual Conference of BRIDGES: Mathematics, Music, Art, Architecture, Culture (BRIDGES 2009)*, pages 1–10, Banff, Canada, July 2009.

[7] Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Anna Lubiw, and Andrew Winslow. Algorithms for solving Rubik's cubes. In *Proceedings of the 19th Annual European Symposium on Algorithms (ESA 2011)*, pages 689–700, September 2011. arXiv:1106.5736.

[8] Erik D. Demaine and MohammadTaghi Hajiaghayi. The bidimensionality theory and its algorithmic applications. *The Computer Journal*, 51(3):292–302, 2008.

[9] Erik D. Demaine and Robert A. Hearn. Playing games with algorithms: Algorithmic combinatorial game theory. In Michael H. Albert and Richard J. Nowakowski, editors, *Games of No Chance 3*, volume 56 of *Mathematical Sciences Research Institute Publications*, pages 3–56. Cambridge University Press, 2009.

[10] Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, July 2007.

[11] Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1–2):72–96, October 2005.

[12] Robert A. Hearn and Erik D. Demaine. *Games, Puzzles, and Computation*. A K Peters, July 2009.

[13] Tomas Rokicki, Herbert Kociemba, Morley Davidson, and John Dethridge. God's number is 20, 2010. `http://cube20.org`.