Mihai Pătraşcu
Curriculum Vitæ

32 Vassar St., Room G596 — `mip@mit.edu`
Cambridge, MA 02139, USA — `http://web.mit.edu/~mip/www/`
Romanian citizen, F-1 visa — Born: July 17, 1982

Education

2007–2008(?) Massachusetts Institute of Technology — PhD (in progress)
Thesis topic: lower bounds for data structures. — Adviser: Erik Demaine

2006–2007 Massachusetts Institute of Technology — Master of Science
Thesis: "*Computational Geometry through the Information Lens*" — Adviser: Erik Demaine
Supported by Akamai Presidential Fellowship.

2002–2006 Massachusetts Institute of Technology — Bachelor of Science in Mathematics with Comp. Sci.
GPA: 5.0/5.0. Phi Beta Kappa Honorary Society.

2001–2002 Univ. of Craiova, Romania — Computer Engineering (freshman)
GPA: 10/10. University exceptional fellowship (first time ever awarded to a freshman).

1997–2001 C.N. Carol I, Craiova, Romania — (high school)
GPA: 9.6/10; Baccalaureate: 9.55. Merit scholarship for outstanding achievement, 1998–2001.

Positions Held

Jun–Aug'07 Research intern at IBM Almaden. Mentor: T.S. Jayram. Manager: Ron Fagin.

Jun–Aug'06 Research intern at AT&T Labs. Mentor: Mikkel Thorup. Manager: David Johnson.

Feb'03–May'06 Undergraduate researcher at MIT CSAIL. Adviser: Erik Demaine

Sep–Dec'02 Undergraduate researcher at MIT LCS, Program Compilation and Verification.
Advisers: Viktor Kuncak and Martin Rinard. — (theorem provers, logic)

Sep'01–Mar'02 Research engineer at Softwin Romania. — (biometrics)

Jul–Aug'01 Research intern at SyncRo Soft, Romania. — (voice recognition)

May–Jun'99 Intern at Idaco Systems, Romania. — (real-time control)

Committees

- Program Committee, 11th Scandinavian Workshop on Algorithm Theory (*SWAT'08*)
- Scientific Committee, 11th Balkan Olympiad in Informatics (2003). Author of 3 contest problems.
- Scientific Committee, Romanian National Olympiad in Informatics, 2002 – 2004; 9 problems.

Grants

2007–2008 Google Research Awards, *Data Structures*, Erik Demaine (PI), Mihai Pătraşcu (research personnel).

Teaching

Aug'06 DIKU (U. Copenhagen) *Lower Bound Techniques for Data Structures*
Two-day summer school, co-taught with Mikkel Thorup.

Spring'05 MIT, EECS *6.897 Advanced Data Structures* (graduate level)
Teaching Assistant for Prof. Erik Demaine.
Gave 4 lectures. Created and graded problem sets. Advised student final projects.
Rated 6.0/7.0 by students. Comments in course evaluations included "motivated things well" and "made difficult material easier to understand."

Jun'03 Training camp for Romania's IOI team *Advanced Data Structures* (day course)

Awards

Excluding awards given by own school (listed above).

- Outstanding Undergraduate Award from the Computing Research Association (*CRA*), 2005.
  Award for best undergraduate research in the US and Canada, received as sophomore.
- Best Student Paper in the 32nd International Colloquium on Automata, Languages and Programming, Track A, for a joint paper with Corina Tarniţă (*ICALP'05*)
- President of Romania's "Award for Excellence" 2000, 2001
- first prize for age group, Romanian National Olympiad in Informatics 1993—2001
- gold medal, 13th International Olympiad in Informatics, Tampere, Finland (*IOI 2001*)
- gold medal, 12th International Olympiad in Informatics, Bei Jing, China (*IOI 2000*)
- silver medal, 7th Central European Olympiad in Informatics Cluj, Romania (*CEOI 2000*)
- silver medal, 8th Balkan Olympiad in Informatics, Ohrid, Macedonia (*BOI 2000*)
- silver medal, 11th International Olympiad in Informatics, Antalya, Turkey (*IOI 1999*)
- gold medal, 6th Central European Olympiad in Informatics, Brno, Czech Rep. (*CEOI 1999*)
- first prize in informatics, Tuymaada Olympiad, Yakutsk, Russia, 1998
- first prize, individual and team, Applied Math Competition, Chişinău, Rep. Moldova, 1996
- Romanian National Olympiad in Physics, first (1996) and second (1997) prize
- various prizes, regional Romanian competitions in Computer Science and Physics

Journal Publications

1. Timothy Chan and Mihai Pătraşcu: **Point Location in Sublogarithmic Time and Other Transdichotomous Results in Computational Geometry**
   Invited to *SIAM Journal on Computing* (special issue with selected papers from FOCS'06).
   Represents a merging of two independent conference publications by each author.
   - Mihai Pătraşcu: **Planar Point Location in Sublogarithmic Time**
     Proc. 47th IEEE Symposium on Foundations of Computer Science (*FOCS'06*), 325–332.
2. Mihai Pătraşcu and Mikkel Thorup: **Higher Lower Bounds for Near-Neighbor and Further Rich Problems**
   Invited to *SIAM Journal on Computing* (special issue with selected papers from FOCS'06).
   Also in Proc. 47th IEEE Symposium on Foundations of Computer Science (*FOCS'06*), 646–654.

3. Ilya Baran, Erik Demaine and Mihai Pătraşcu: **Subquadratic Algorithms for 3SUM**
   Algorithmica, to appear. Special issue with selected papers from WADS'05.
   Also in Proc. 9th Workshop on Algorithms and Data Structures (*WADS'05*), pp. 409–421.
4. Mihai Pătraşcu and Corina Tarniţă: **On Dynamic Bit-Probe Complexity**
   Theoretical Computer Science 380, pp. 127–142 (2007). Special issue for ICALP'05.
   Also in Proc. 32nd International Colloquium on Automata, Languages and Programming (*ICALP'05*), pp. 969–981. Received *Best Student Paper* Award.
5. Erik Demaine, Dion Harmon, John Iacono, and M. Pătraşcu: **Dynamic Optimality—Almost**
   SIAM Journal on Computing, 37(1), pp. 240–251 (2007). Special issue for FOCS'04.
   Also in Proc. 45th IEEE Symposium on Foundations of Computer Science (*FOCS'04*), 484–490.
6. Mihai Pătraşcu and Erik Demaine: **Logarithmic Lower Bounds in the Cell-Probe Model**
   SIAM Journal on Computing 35(4), pp. 932–963 (2006). Special issue with selected papers from STOC'04. Preliminary versions appeared as:
   - **Lower Bounds for Dynamic Connectivity**
     Proc. 36th ACM Symposium on Theory of Computing (*STOC'04*), pp. 546–553.
   - **Tight Bounds for the Partial-Sums Problem**
     Proc. 15th ACM–SIAM Symposium on Discrete Algorithms (*SODA'04*), pp. 20–29.
     Invited to special issue of ACM Transactions on Algorithms; declined.
7. Mihai Pătraşcu: **On Two Problems from the National Olympiad in Informatics 2002, New Solutions and Generalizations** (in Romanian)
   Gazeta Informatică, February 2003, pp. 13–14.

Conference Publications

Papers already published in journals are *only* listed above.

8. Amit Chakrabarti, T. S. Jayram, and Mihai Pătraşcu:
   Tight Lower Bounds for Selection in Randomly Ordered Streams
   Proc. 19th ACM/SIAM Symposium on Discrete Algorithms (*SODA'08*), to appear.
9. Mihai Pătraşcu and Mikkel Thorup: **Planning for Fast Connectivity Updates**
   Proc. 48th IEEE Symposium on Foundations of Computer Science (*FOCS'07*), pp. 263–271.
10. Gianni Franceschini, S. Muthukrishnan, and M. Pătraşcu: **Radix Sorting With No Extra Space**
    Proc. 15th European Symposium on Algorithms (*ESA'07*), 194–205. Full version `arXiv:0706.4107`.
11. Mihai Pătraşcu: **Lower Bounds for 2-Dimensional Range Counting**
    Proc. 39th ACM Symposium on Theory of Computing (*STOC'07*), pp. 40–46.
12. Timothy Chan and Mihai Pătraşcu: **Voronoi Diagrams in $n \cdot 2^{O(\sqrt{\lg\lg n})}$ Time**
    Proc. 39th ACM Symposium on Theory of Computing (*STOC'07*), pp. 31–39.
13. Erik Demaine and M. Pătraşcu: **Tight Bounds for Dynamic Convex Hull Queries (Again)**
    Proc. 23rd ACM Symposium on Computational Geometry (*SoCG'07*), pp. 354–363.
14. Nicholas Harvey, Mihai Pătraşcu, Yonggang Wen, Sergey Yekhanin, and Vincent Chan:
    Non-Adaptive Fault Diagnosis for All-Optical Networks via Combinatorial Group Testing on Graphs,
    Proc. 26th IEEE Conference on Computer Communications (*INFOCOM'07*), pp. 697–705.

15. M. Pătraşcu and Mikkel Thorup: **Randomization Does Not Help Searching Predecessors** Proc. 18th ACM–SIAM Symposium on Discrete Algorithms (*SODA'07*), pp. 555–564.

16. Alexandr Andoni, Piotr Indyk, and Mihai Pătraşcu: **On the Optimality of the Dimensionality Reduction Method**
Proc. 47th IEEE Symposium on Foundations of Computer Science (*FOCS'06*), pp. 449–458.

17. Mette Berger, Esben Rune Hansen, Rasmus Pagh, M. Pătraşcu, Milan Ružić, and Peter Tiedemann: **Deterministic Load Balancing and Dictionaries in the Parallel Disk Model**
Proc. 18th ACM Symposium on Parallelism in Algorithms and Architectures (*SPAA'06*), 299–307.

18. Mihai Pătraşcu and Mikkel Thorup: **Time-Space Trade-Offs for Predecessor Search**
Proc. 38th ACM Symposium on Theory of Computing (*STOC'06*), pp. 232–240.

19. Erik Demaine, Friedhelm Meyer auf der Heide, Rasmus Pagh, and Mihai Pătraşcu: **De Dictionariis Dynamicis Pauco Spatio Utentibus** (On Dynamic Dictionaries Using Little Space)
Proc. 7th Latin American Theoretical Informatics (*LATIN'06*), pp. 349–361. Full version available as `arXiv:cs.DS/0512081`.

20. Micah Adler, Erik Demaine, Nicholas Harvey, and Mihai Pătraşcu: **Lower Bounds for Asymmetric Communication Channels and Distributed Source Coding**
Proc. 17th ACM–SIAM Symposium on Discrete Algorithms (*SODA'06*), pp. 251–260.

21. Christian Worm Mortensen, Rasmus Pagh, and Mihai Pătraşcu: **On Dynamic Range Reporting in One Dimension**, in Proc. 37th ACM Symposium on Theory of Computing (*STOC'05*), pp. 104–111. Full version available as `arXiv:cs.DS/0502032`.

22. Corina Tarniţă and Mihai Pătraşcu: **Computing Order Statistics in the Farey Sequence**
Proc. 6th Algorithmic Number Theory Symposium (*ANTS'04*), pp. 358–366.

23. Stelian Ciurea, Erik Demaine, Corina Tarniţă, and Mihai Pătraşcu:
Finding a Divisible Pair and a Good Wooden Fence
Proc. 3rd International Conference on Fun with Algorithms (*FUN'04*), pp. 206–219.
A poster on the divisible pair problem was displayed at the 6th Algorithmic Number Theory Symposium (ANTS'04). An invited extended abstract of the divisible-pair material appeared in the *ACM SIGSAM Bulletin*, volume 38:3, September 2004, pp. 98–100.

24. Erik Demaine, Thouis Jones, and M. Pătraşcu: **Interpolation Search for Non-Independent Data**
Proc. 15th ACM–SIAM Symposium on Discrete Algorithms (*SODA'04*), pp. 522–523.

Other Publications

25. Timothy Chan, Mihai Pătraşcu, and Liam Roditty: **Dynamic Connectivity: Connecting to Networks and Geometry.** Submitted to STOC'08.

26. Jakub Pawlewicz and Mihai Pătraşcu: **Order Statistics in the Farey Sequences in Sublinear Time**. Submitted to *Algorithmica.* Merging of a paper by Pawlewicz in ESA'07, and my subsequent technical report `arXiv:0706.4107`.

27. Mihai Pătraşcu: **Searching the Integers**
Invited mini-survey in *Encyclopedia of Algorithms* (Springer Reference Works).

28. Mihai Pătraşcu: **Lower Bounds for Dynamic Connectivity**
Invited mini-survey in *Encyclopedia of Algorithms* (Springer Reference Works).

29. Alexandr Andoni, Dorian Croitoru, and Mihai Pătraşcu: **Hardness of Nearest-Neighbor Search under ℓ_∞.** Manuscript.

30. Marek Karpinski, Yakov Nekrich, and Mihai Pătraşcu: **Reporting from 4 Dimensions: The Butterfly Effect.** Manuscript.
31. Alexandr Andoni and Mihai Pătraşcu: **Hardness of Sketching for the Edit Distance** In preparation.

Coauthors

Micah Adler (*Prof., U. Mass*), Alexandr Andoni (*student, MIT*), Ilya Baran (*student, MIT Graphics*), Mette Berger (*OctoShape Denmark*), Amit Chakrabarti (*Prof., Dartmouth*), Timothy Chan (*Prof., U. Waterloo*), Vincent Chan (*Prof., MIT LIDS*), Stelian Ciurea (*Prof., U.L.B. Sibiu*), Dorian Croitoru (*student, MIT Math*), Erik Demaine (*Prof., MIT*), Gianni Franceschini (*postdoc, U. Pisa*), Esben Rune Hansen (*student, IT U. Copenhagen*), Dion Harmon (*student, MIT*), Nicholas Harvey (*student, MIT*), John Iacono (*Prof., Brooklyn Poly.*), Piotr Indyk (*Prof., MIT*), T. S. Jayram (*IBM*), Thouis Jones (*student, MIT Graphics*), Marek Karpinski (*Prof., U. Bonn*), Friedhelm Meyer auf der Heide (*Prof., U. Paderborn*), Christian Mortensen (*student, IT U. Copenhagen*), S. Muthukrishnan (*Google; Prof. Rutgers*), Yakov Nekrich (*postdoc, U. Bonn*), Rasmus Pagh (*Prof., IT U. Copenhagen*), Jakub Pawlewicz (*student, Warsaw Univ.*), Liam Roditty (*postdoc, Weizmann*), Milan Ružić (*student, IT U. Copenhagen*), Corina Tarniţă (*student, Harvard Math*), Peter Tiedemann (*student, IT U. Copenhagen*), Mikkel Thorup (*AT&T*), Yonggang Wen (*student, MIT LIDS*), Sergey Yekhanin (*Microsoft Research*).

Research Visits and Talks

Excluding conference talks.

Feb'08 Dagstuhl meeting on *Data Structures* / *TBD*

Nov'07 MIT / *A Perspective on Slepian-Wolf Coding*

Oct'07 U.L. Bruxelles / Stefan Langerman / *Farey Sequences & Counting Primitive Lattice Points*

Oct'07 U. Bonn / Yakov Nekrich / *Dynamic Graph Algorithms invade Geometry*

Sep'07 Tsinghua U. / China Theory Week / *Round Elimination: A Proof, A Concept, A Direction*

Jul'07 IBM Almaden / *Dynamic Optimality—Almost*

May'07 U. Washington / Paul Beame / *Lower Bounds for 2-Dimensional Range Counting*

May'07 Microsoft Research, Redmond / Asaf Shapira / *Planning for Fast Connectivity Updates*

Apr'07 MIT / Piotr Indyk / *Geometric Searching with Bounded Precision*
Guest lecture in 6.850 Geometric Computation (graduate level).

Apr'07 MIT / Erik Demaine / *Tight Lower Bounds for Predecessor Search*
Guest lecture in 6.851 Advanced Data Structures (graduate level).

Mar'07 Brown Univ. / Crystal Kahn / *"Dynamic Connectivity": Questions and Some Answers*

Feb'07 UPenn / Sanjeev Khanna / *On the Optimality of the Dimensionality Reduction Method*

Feb'07 MIT / Crypto & Complexity / *Information Complexity and High-dimensional Geometry*

Dec'06 Tel Aviv U. / Uri Zwick / *C.G. Through the Information Lens: Dynamic Convex Hull*

Dec'06 Weizmann Inst. / Liam Roditty / *C.G. Through the Information Lens: Voronoi Diagrams*

Dec'06 The Technion, Haifa / Yuval Ishai / *C.G. Through the Information Lens: Point Location*

Aug'06 Bell Labs / Lisa Zhang / *Planar Geometry on the Grid*

Aug'06 AT&T Labs / *Communication Complexity and Data-Structure Lower Bounds*

Jul'06 NEC Labs, New Jersey / Pranab Sen

Jun'06 IBM Almaden / T.S. Jayram / *Data-Structure Lower Bounds*

Apr'06 Stanford / Theory Lunch / *Searching in an Integer Universe*

Apr'06 MIT / Algorithms & Complexity / *Hardness Results for Near-Neighbor Problems*

Mar'06 U. Washington / Paul Beame / *Cell-Probe Complexity and Predecessor Search*

Oct'05 MIT / ToC Student Seminar / *Cell-Probe versus Communication Complexity*

Jun'05 Max Planck Institut für Informatik, Saarbrücken / Seth Pettie / *The Saga of Dynamic Lower Bounds around the Logarithmic Barrier*

Jun'05 Oberwolfach meeting on *Complexity Theory*

Sep'04 IT U. Copenhagen / Rasmus Pagh / *Logarithmic Lower Bounds in the Cell-Probe Model*

Scientific Service

Journal Referee: JACM, SICOMP, TAlg, Algorithmica, Information & Computation, IPL, Computers & Graphics

Conference Referee: STOC, FOCS, SODA, SoCG, WADS, ESA, STACS, FSTTCS

Developed the software system used for grading the 11th Balkan Olympiad in Informatics (*BOI'03*), the Romanian National Olympiad, and several regional olympiads

Mihai Pătraşcu
Research Statement

In my research work, I find myself asking time and again: what exactly am I doing, and why do I think it is important? I struggle to maintain an objective (some would say cynical) understanding of my research goals and motivation. Theoretical research has, in my opinion, three main deliverables:

splash factor: This is the most directly measurable outcome — what is the size of the triumphant march when an open problem is solved? This immediate impact is proportional to how compelling, old, and well-established the problem is. Viewed in this way, research is a competitive sport where the object is to solve famous problems.

theoretical impact: After immediate impact is forgotten, future impact in theory is proportional to the "niceness" of a solution. The development of theories is often spearheaded by *gems*: solutions that are easy but fascinating, which can be taught in courses, and enter the collective mind. By contrast, "painful" solutions may have high initial impact, but in the long term remain mere references (and a turn-off).

broad impact: This is the broad understanding generated outside theory — the bottom line of the work, in the eyes of somebody who doesn't necessarily want to read the proof. In some cases, this is an implementable algorithm for a useful problem. In other cases, it is a more abstract type of understanding: for example, as useful as some algorithms are, they cannot compete with the impact of NP-completeness. While this type of outcome is the most elusive and hardest to judge, it is also the one I find most compelling and, consequently, the one to which I allocate the most time.

In the following, I consider these three classes of deliverables and discuss some of my past work and research directions in light of these objectives.

Splash factor. The following are some highlights of our work, judged by immediate impact. (All citations in this document refer to the numbering in my CV.)

[**6**] showed the first $\Omega(\lg n)$ lower bound for *any* dynamic data structure, and in particular solved some well-studied problems such as dynamic trees and partial sums. The previous lower-bound record had been standing since the work of Fredman and Saks in *STOC 1989*, despite significant efforts in the area. In a survey from 2000 on cell-probe complexity, Miltersen listed an $\Omega(\lg n)$ lower bound as one of the three "*major challenges for future research.*" Surprisingly, our solution was extremely simple (less than 3 pages, and no calculation!)

[**4**] reproved the same $\Omega(\lg n)$ dynamic lower bound, with a minor twist to the old epoch argument of Fredman and Saks. This was unexpected, since all previous attempts to obtain an $\Omega(\lg n)$ bound had concentrated on epoch arguments. This new proof allowed us to deduce an $\widetilde{\Omega}(\lg^2 n)$ lower bound in the bit-probe model, solving a second of the three "major challenges" from Miltersen's survey. This paper obtained the *Best Student Paper Award* at ICALP.

[**5**] gave the first nontrivial competitive ratio for an online binary search tree, 21 years after Sleator and Tarjan proposed their famous dynamic optimality conjecture (the splay conjecture).

[**18**] was the first paper to separate *linear* space from *polynomial* space. Previous lower-bound techniques could not differentiate, say, space $O(n \lg n)$ and space $O(n^9)$, and thus were helpless against many central problems. Our initial paper closed the predecessor problem, which was undoubtedly the most well-studied problem in static data structures. Since then, these initial ideas have been developed significantly for understanding other problems [2, 11, 30, 15].

[1] solved planar point location in time $O(\sqrt{\lg u})$ if the query comes from the grid $[u] \times [u]$. This was the first sublogarithmic bound for point location, and allowed us to revisit many fundamental bounds in computational geometry, such as the complexity of constructing Voronoi diagrams, segment intersection, dynamic convex hull etc. Breaking the logarithmic barrier for these problems was a well-known open problem since, at least, [Willard: *SODA 1992*]. In recent years, John Iacono had been actively advocating this as a major challenge.

[11] showed a tight lower bound for range counting in two dimensions (e.g. count the number of employees in a company that earn between 70k and 90k, and were hired between 1995 and 2000). This was the first tight bound for group computations, despite a large body of literature proving bounds in the weaker semigroup model. Proving such a bound in the group model has been regarded as an important challenge at least since [Fredman: *JACM 1982*] and [Chazelle: *FOCS 1986*], and was the main lower-bound question in the survey on range searching by Agarwal and Erickson, from 1999.

[8] showed that any algorithm for finding the median using space $\text{poly}(\lg n)$ requires $\Omega(\lg \lg n)$ passes through the data. This resolved the main open question from the seminal paper on streaming [Munro, Paterson: *FOCS 1978*]. In talks during recent years, Sudipto Guha had repeatedly advertised the problem as an important challenge.

[16] showed a tight lower bound for the most basic problem in asymmetric communication: set disjointness. This was posed as an open problem in the seminal paper on asymmetric communication [Miltersen, Nisan, Safra, Wigderson: *STOC 1995*]. Our proof was strikingly simple (about one page). The benefits of tackling this basic problem were immediate: by simple reductions, our result improved and greatly simplified previous results for partial match and high-dimensional nearest neighbor [Borodin, Ostrovsky, Rabani: *STOC 1999*; Barkol, Rabani: *STOC 2000*; Jayram, Khot, Kumar, Rabani: *STOC 2003*].

[25] showed how to maintain an understanding of connectivity in a dynamic graph under node updates (turning network nodes on and off). Our algorithm in combinatorial and runs in time $\widetilde{O}(m^{2/3})$, whereas the previous algorithm of Chan only obtained a theoretical sublinear time of $O(m^{0.94})$, using fast matrix multiplication. Our elementary algorithm was a surprise, since Chan's paper from *STOC 2002* had stated that "we suspect [...] fast-matrix multiplication is essential to solve our problem."

These results are an expression of my determination to attack interesting problems, whether or not they are considered hard by the community. Of course, I fail to solve such problems on a regular basis — for instance, I twice thought I had an attack on superlinear circuit lower bounds, which is a problem famous enough to be labeled a wild goose chase.

In my future research, I will certainly not be more intimidated by famous problems than I was in the past, and hopefully I will continue to solve some of them.

Theoretical impact. Several of my results have been taught in courses at various universities:

- CS573 Advanced Data Structures, by Jeff Erickson (*Univ. Illinois, Urbana and Champaign*), contained three lectures based on our results: $\Omega(\lg n)$ dynamic lower bounds [6], the predecessor lower bound [18], and competitive binary search trees [5].
- Together with Erik Demaine, I developed MIT's own version of Advanced Data Structures, which also teaches these three results [6, 18, 5].

- CS860 Five Open Problems in Algorithm Design and Analysis, by Alejandro Lopez-Ortiz (*Univ. of Waterloo*), contained two lectures based on our results: subquadratic algorithms for 3SUM [3], and competitive binary search trees [5].
- CMSC39600 Online Algorithms, by Adam Kalai (*Georgia Tech*) contained a lecture on competitive binary search trees [5].
- In 6.850 Geometric Computation, by Piotr Indyk (*MIT*), I gave a guest lecture on sublogarithmic point location [1].

Outside regular courses, some of our work was also presented in the following contexts:

- Some results were presented in reading groups. (I have been told of such cases at the Max Plank Institut, and Warsaw University.)
- Together with Mikkel Thorup, I taught a 2-day summer school at DIKU (Univ. of Copenhagen) that covered [6, 4, 18, 2].
- I was invited to contribute two short survey articles (on predecessor search [18, 15] and logarithmic lower bounds [6, 4]) to Springer's *Encyclopedia of Algorithms.*
- Our work on Bloomier filters [21, 19] was described on Wikipedia (though, admitedly, the description did not always make sense).

I have invested significant effort trying to reach a level of understanding of new results where proofs become simple and intuitive. This has been a rewarding process, and I believe several recent results may find their own way into courses in the future. An important goal of my research career is to continue to seek such crisp and clear explanations.

Broad impact. In *Introduction to Algorithms* classes, we often assign homework questions that ask students to "give the best running time you can find." Unfortunately, we cannot hold ourselves to the same standard: for most algorithmic problems that we discuss in the lectures, we do not understand the optimal running time! Indeed, given the current state of lower bounds, we do not even expect matching bounds in any near future.

Some of my work has focused on developing better lower-bound techniques, with some applications as basic as problems from *Introduction to Algorithms.* As such, the bottom line of my work can be communicated effectively to a very broad audience.

For example, consider binary search trees — the central data structure in an introductory class, and probably all of computer science. What algorithmic problems does this data structure actually solve? A binary search tree maintains a set S of "keys" (under insertions, deletions, splits, concatenates) and answers the following types of queries in logarithmic time:

dictionary: Test whether a key x is in S, and potentially retrieve some data associated with x.

predecessor search: Given some $x \notin S$, find the predecessor and successor of x in the set (i.e. the elements of S that flank x).

dynamic indexing: Given an i, search for the i-th element in S. (This is done by augmenting each node with the number of elements in its subtree. In general, augmented trees answer many types of queries, for which we get optimal lower bounds by the same technique.)

Are binary search trees the best way to solve these problems, and, if not, what are the optimal running times? For predecessor search and dynamic indexing, the optimal bounds were finally understood in our work [6, 18, 15]. Understanding dictionaries is one of the most appealing open problems that I am currently considering.

For **dictionaries**, we learn a constant-time solution in the very next lecture: hash tables. The downside is, of course, that hash tables require randomization, i.e. the running time is only constant *in expectation.* Despite significant work on derandomizing dictionaries (e.g. perfect hashing, including our work in [19, 21]), it is generally believed that we cannot achieve constant running times deterministically. Thus, for most students, dictionaries are the first example where randomization seems to give a better result!

Proving that deterministic dictionaries must actually take more than constant time is a tantalizing open problem. I believe I have an angle of attack on this problem, and I am actively pursuing it in collaboration with Mikkel Thorup. We speculate the optimal trade-off between the query and update time will be $t_q \cdot t_u \approx \lg n$ (though currently, we do not even know such an upper bound).

Dynamic indexing is a problem arising naturally in programming practice. When learning to program, one is often taught the trade-off between lists and arrays: lists can be manipulated efficiently (by splits, concatenates, etc.), while arrays can be indexed efficiently. Sometimes, however, we need both types of operations. In fact, the combination of indexing and list manipulation is immortalized in the syntactic sugar of some programming languages. Python has a single type for both lists and arrays, which supports both list updates and indexing as primitive operations; see Figure 1 for illustrative examples.

```
>>> a = [0, 1, 2, 3, 4]
>>> a[2:2] = [9, 9, 9]
>>> a
[0, 1, 9, 9, 9, 2, 3, 4]
>>> a[5]
2
```

Figure 1: Python "lists"

In [6], we proved an $\Omega(\lg n)$ lower bound for this problem, which showed that standard binary search trees are in fact the optimal solution! For entertainment value, I note that [6] was written while I was actually taking *Introduction to Algorithms*, as an undergraduate freshman at MIT.

Predecessor search has been dubbed "the most executed algorithmic problem on the planet." Indeed, routers on the Internet run a predecessor search for every packet that they handle: to determine the correct outgoing link, they need to find the routing rule that matches the packet's destination IP.

In theory, a solution that can beat binary search trees is given by the classic data structure of [van Emde Boas: *FOCS 1975*]: if searching among numbers of w bits, the running time is $O(\lg w)$. In practice, [Degermark, Brodnik, Carlsson, Pink: *SIGCOMM 1997*] demonstrated the feasibility of routing in software at Gigabit speeds. Their algorithm is based precisely on the idea of van Emde Boas, with careful engineering of the implementation (for $w = 32$).

In [18], we gave optimal space/time trade-offs for predecessor search, showing that van Emde Boas is indeed optimal for quasilinear space. Thus, any further improvements in practice could only be gained by more engineering of the implementation, not by changing the basic search strategy.

In the future, I will continue to search for elegant solutions to natural and important problems, with the goals of making an impact on our broader understanding and of creating efficient algorithms that can be used in practice. As an example of a direction I am currently enthusiastic about, consider quad trees, a data structure fundamental to practice (computer graphics) and theory alike. We have demonstrated [1] that an alternative data structure can improve quad-tree bounds for the fundamental problem of point location. The new data structure is based on a decomposition in the form of a directed acyclic graph instead of a simple tree. Can such an idea be used to gain an advantage in practice? For example, one can seek a fast implementation of point location, which has good guarantees in the worst case. More broadly, one could hope for faster algorithms for some of the many problems where quad trees have been the canonical solution in practice.

Mihai Pătraşcu

Teaching Statement

My goal in teaching. When I was 8, I attended an open course at a Romanian "youth club" where I learned to program in BASIC on a clone of the Z80 Sinclair. A year later, with my reading skills finally improving, I picked up a book and learned to program in Pascal. By 4th grade, I had gained another useful skill: reading English. So I borrowed a 5-lb tome on the C programming language, and convinced my parents to buy an old 8086 computer.

Like many enthusiastic computer scientists, I started in AI. Much of 5th grade was spent implementing a chess program, from minimax exploration of the game tree, to a user interface on my 320x200-pixels CGA screen. Eventually, my program managed to beat my young sister. But that rewarding moment came too late for my future as an AI researcher: computers had already become a purpose in themselves, so I switched to systems. I spent the next 2 years implementing a compiler for FORTRAN IV, and a Common Lisp system (arguably, the antiquated collection of books that I could borrow made for an eccentric choice). By 8th grade, I was finally ready to embark on my search for the philosopher's stone: I switched to theory.

My experience has profoundly shaped my personal goal in teaching: I have no intention to ever teach computer science. I want to teach the *love* for computer science, and let the learning happen.

Fancy statements aside, I do not believe I would have been able to appreciate lectures on, say, alpha-beta exploration or context-free grammars, before seeing motivating applications and developing my own *ad hoc* solutions for them. But once I had done some independent thinking about the problem, it was easy and rewarding to understand the "official" solution based on general principles. Thus, I find the most challenging aspect of teaching is to motivate the material, and highlight the wonderful connections across the breadth of computer science. Teaching the actual material to students who *want* to learn it is easy.

As an illustration of the importance I attach to motivation, I would sacrifice problem sets to this altar. Rather than assigning problem sets on a recent lecture, I would assign problems sets on a motivating application of a future lecture. Once they develop the courage to think creatively, students will produce *some* solution, and they will be ready to understand and appreciate a principled solution in a general framework. By contrast, problem sets consisting of applications to recent material tend to generate a kind of "parrot understanding" in students, who know they will forget everything soon after then course is over.

Advanced Data Structures. After the course on BASIC from 2$^{\text{nd}}$ grade, the very next Computer Science class I took was in my freshman year at MIT (2003). This was the first instantiation of Erik Demaine's graduate class, *Advanced Data Structures*. As this became my main area of research, I obtained a better perspective on the material, and started considering how it could be taught better.

To get involved more directly, I became a Teaching Assistant for the next offering of the class (2005), and worked closely with Erik in a semester-long effort to design the course coverage and each lecture individually. The result was rather rewarding, and the breadth of the class was significantly expanded (only a third of the original lectures remained). A significant amount of effort was spent on placing each lecture in context, and on emphasizing connections within and beyond the boundaries of the course. Students repeatedly told us that they enjoy that broad and motivating view that the course offers.

Besides being involved in creating the contents of each lecture, I gave 4 of the lectures myself. In the final course evaluations, students commented that "Mihai motivated things well" and "made difficult material easier to understand." I received an average rating of 6.0 out of 7.

This course is now in a rather stable form, and Erik taught it again in 2007. I would like to teach a similar course in the future.

Undergraduate curriculum. At MIT, and, I am told, most top universities, a weak spot of the undergraduate curriculum is teaching students to write efficient code. After a few core classes like introduction to programming, we have an unfortunate dissociation between writing code and dealing with efficiency: the former is done in software labs and systems classes, which emphasize writing *large* bodies of code, while the latter is done in theoretical algorithms courses, which at best feature a few programming assignments with a patronizing feel. I would like to augment existing classes with notions of efficient programming, and ideally teach a new class entirely on the subject.

To counter a possible misunderstanding, the skill I would like to teach is not how to move `if`'s around to obtain a 2% improvement in efficiency, making a mess of the code. It is how to design algorithms which are implementable in clean code and offer order-of-magnitude improvements over naïve solutions. With our ever-increasing demands for software, a product has many components that could easily move to the critical path if not implemented intelligently (and, with the ever increasing data size, an inefficient algorithm on the critical path is often intolerable). On the other hand, software engineering considerations dictate that all these components should be implemented as cleanly as possible, as long as they are efficient enough to stay off the critical path.

If I may refer to anecdotal evidence from Google, such training seems much needed in today's software landscape. Google finds it particularly easy to hire people from computer olympiads (to the point where walking down their corridors feels like an IOI reunion). Though these are high-school olympiads, they seem to teach a skill for efficient programming that is a remarkable predictor for success even after college.

I believe I am in a fortunate position to teach efficient programming due to my long-term involvement in computer olympiads. I have had many competition years to observe myself and my peers develop the right algorithmic thinking; then, I switched sides to create problems and teach minicourses at training camps. Teaching a college course on efficient programming feels like a natural continuation in which I could capitalize on this knowledge.

Graduate curriculum. Since my work spans several areas of research, I would like to teach several graduate / advanced undergraduate courses such as:

- Concrete Complexity: a view of complexity theory with emphasis on lower bounds in concrete models of computation (communication complexity, circuits, branching programs, algebraic models etc).
- Dealing with Massive Data: a course focusing on efficient algorithms for very large data sets. Ideally, the course would be at intersection of high-dimensional geometry, streaming algorithms, very large databases, networking, and parallel algorithms.
- Computational Geometry: a standard course on low- and high-dimensional geometry, augmented with connections to graphics and vision.