CS125 Final Lecture

Areas of TCS

(Theoretical Computer Science)

Areas of TCS

- Algorithms
- Data structures
- Complexity Theory
- Computer Science + Economics
- Cryptography and Privacy
- Computational Learning Theory
- Coding Theory
- Quantum Computing

Areas of TCS

- **Algorithms**
  - Word RAM
  - Graph algorithms
  - Algorithmic spectral graph theory
  - Algorithmic linear algebra
  - Distributed algorithms
  - Parallel algorithms
  - Property testing (like the BLR linearity test)
  - Streaming algorithms
  - Online algorithms
  - Approximation algorithms
  - External-memory / cache-oblivious
  - Computational geometry
  - Number-theoretic problems (e.g. primality testing)

Areas of TCS

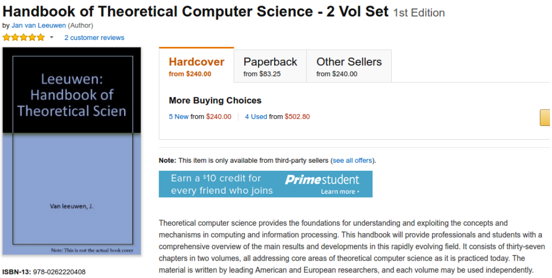
- **Data structures**
  - Many qualifiers: amortized/worst case, static/dynamic, persistent/ephemeral, randomized/deterministic
  - Tradeoffs (time vs. space, or update vs. query time), upper and lower bounds
- **Complexity Theory**
  - Concrete complexity (communication complexity, branching programs, circuits, formulae, ...)
  - Pseudorandomness
  - Algebraic complexity
  - Proof complexity
  - Interactive proof systems
- **Computer Science + Economics**
  - Algorithmic mechanism design
  - Algorithmic game theory
- **Cryptography and Privacy**

Areas of TCS

- **Computational Learning Theory**
- **Coding Theory**
- **Quantum Computing**
  - Quantum error-correction
  - Quantum communication complexity
  - Quantum complexity ("quantum Turing machines")
  - Quantum algorithms

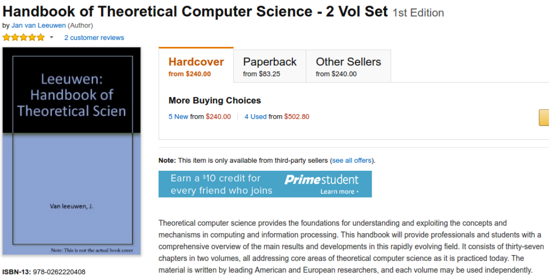
Also “Theory B”

Theory B



Volume A *covers models of computation, complexity theory, data structures, and efficient computation in many recognized subdisciplines of theoretical computer science.* **Volume B** *presents a choice of material on the theory of automata and rewriting systems, the foundations of modem programming languages, logics for program specification and verification, and several chapters on the theoretic modeling of advanced information processing ...*

Theory B



Volume A *covers models of computation, complexity theory, data structures, and efficient computation in many recognized subdisciplines of theoretical computer science.* **Volume B** *presents a choice of material on the theory of automata and rewriting systems, the foundations of modem programming languages, logics for program specification and verification, and several chapters on the theoretic modeling of advanced information processing . . .*

Take for example: CS152 (Programming Languages), CS252r

Back to Theory A

Areas of TCS

- **Algorithms**
  - Word RAM
  - Graph algorithms
  - Algorithmic spectral graph theory
  - Algorithmic linear algebra
  - Distributed algorithms
  - Parallel algorithms
  - Property testing (like the BLR linearity test)
  - Streaming algorithms
  - Online algorithms
  - Approximation algorithms
  - External-memory / cache-oblivious
  - Computational geometry
  - Number-theoretic problems (e.g. primality testing)

Word RAM

- Seen in this course: hashing, counting sort, etc.
- Mentioned throughout semester:
    - Dynamic predecessor in $O(\lg w)$ [van Emde Boas'75]
    - Least significant bit in $O(1)$ time [Fredman, Willard'90]
    - sorting in $O(n\sqrt{\lg \lg n})$ time randomized [Han, Thorup'02]
    - sorting in $O(n \lg \lg n)$ time deterministic [Han'02]
    - Min spanning tree in $O(m + n)$ deterministic [Fredman, Willard'94]
    - Undirected single source shortest paths in $O(m + n)$ [Thorup'99]
    - Directed SSSP in $O(m + n \lg \lg n)$ [Thorup'04]
    - ...

Word RAM

- Seen in this course: hashing, counting sort, etc.
- Mentioned throughout semester:
  - Dynamic predecessor in $O(\lg w)$ [van Emde Boas'75]
  - Least significant bit in $O(1)$ time [Fredman, Willard'90]
  - sorting in $O(n\sqrt{\lg \lg n})$ time randomized [Han, Thorup'02]
  - sorting in $O(n \lg \lg n)$ time deterministic [Han'02]
  - Min spanning tree in $O(m+n)$ deterministic [Fredman, Willard'94]
  - Undirected single source shortest paths in $O(m+n)$ [Thorup'99]
  - Directed SSSP in $O(m + n \lg \lg n)$ [Thorup'04]
  - . . .

Can see some more instances of "the power of word RAM" in CS224 and 6.851 (MIT).

Areas of TCS

- **Algorithms**
  - Word RAM
  - Graph algorithms
  - Algorithmic spectral graph theory
  - Algorithmic linear algebra
  - Distributed algorithms
  - Parallel algorithms
  - Property testing (like the BLR linearity test)
  - Streaming algorithms
  - Online algorithms
  - Approximation algorithms
  - External-memory / cache-oblivious
  - Computational geometry
  - Number-theoretic problems (e.g. primality testing)
  - ...

Algorithmic spectral graph theory / linear algebra

- **Algorithmic spectral graph theory**
  - Can say a lot about graphs by looking at eigenvectors and eigenvalues of matrices.
  - A adj. matrix, $D = diag$(degrees), $L = D - A$ is "Laplacian"
  - Laplacian has eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$

Algorithmic spectral graph theory / linear algebra

- **Algorithmic spectral graph theory**
  - Can say a lot about graphs by looking at eigenvectors and eigenvalues of matrices.
  - A adj. matrix, $D = diag(\text{degrees})$, $L = D - A$ is "Laplacian"
  - Laplacian has eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$
  - **Thm:** Undirected graph G has k connected components iff eigenvalue 0 has multiplicity k.
    (in particular, $\lambda_2 = 0$ iff G disconnected)

Algorithmic spectral graph theory / linear algebra

- **Algorithmic spectral graph theory**
  - Can say a lot about graphs by looking at eigenvectors and eigenvalues of matrices.
  - A adj. matrix, $D = diag(\text{degrees})$, $L = D - A$ is "Laplacian"
  - Laplacian has eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$
  - **Thm:** Undirected graph G has k connected components iff eigenvalue 0 has multiplicity k.
    (in particular, $\lambda_2 = 0$ iff G disconnected)
  - **More robust (Cheeger):** λ_2 small iff G has a sparse cut.
    (And if λ_2 is small, a sparse cut can be found efficiently.)

Algorithmic spectral graph theory / linear algebra

- **Algorithmic spectral graph theory**
  - Can say a lot about graphs by looking at eigenvectors and eigenvalues of matrices.
  - A adj. matrix, $D = diag(\text{degrees})$, $L = D - A$ is "Laplacian"
  - Laplacian has eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$
  - **Thm:** Undirected graph G has k connected components iff eigenvalue 0 has multiplicity k.
    (in particular, $\lambda_2 = 0$ iff G disconnected)
  - **More robust (Cheeger):** λ_2 small iff G has a sparse cut.
    (And if λ_2 is small, a sparse cut can be found efficiently.)
  - **Recently:** "higher-order Cheeger". λ_k small iff G can be partitioned into k clusters without many edges crossing the clusters. [Lee, OveisGharan, Trevisan'12], [Kwok, Lau, Lee, OveisGharan, Trevisan'13], . . .

Algorithmic spectral graph theory / linear algebra

- **Algorithmic linear algebra**
  - $Lx = b$ in nearly-linear time [Spielman, Teng'04], [Koutis, Miller, Peng'10], [Koutis, Miller, Peng'11], [Kelner, Orecchia, Sidford, Zhu'13], [Cohen, Kyng, Miller, Pachocki'14], [Peng, Spielman'14], [Kyng, Lee, Peng, Sachdeva, Spielman'16], [Kyng, Sachdeva'16]
    useful subroutine in other problems, like max-flow

Algorithmic spectral graph theory / linear algebra

- **Algorithmic linear algebra**
  - $Lx = b$ in nearly-linear time [Spielman, Teng'04], [Koutis, Miller, Peng'10], [Koutis, Miller, Peng'11], [Kelner, Orecchia, Sidford, Zhu'13], [Cohen, Kyng, Miller, Pachocki'14], [Peng, Spielman'14], [Kyng, Lee, Peng, Sachdeva, Spielman'16], [Kyng, Sachdeva'16]
    useful subroutine in other problems, like max-flow
  - $L = \sum_{e \in E} b_e b_e^T$ (for $e = (u, v)$, $b_e = \mathbf{1}_u - \mathbf{1}_v$ for $u < v$)
  - For $x \in \{-1, 1\}^n$, $\frac{1}{4} x^T L x$ is size of cut
  - **Thm:** Can sample only $O(n \lg n)$ edges s.t. new $\tilde{L}$ satisfies $x^T \tilde{L} x \approx x^T L x$ for all $x \in \{-1, 1\}^n$ [Benczúr, Karger'96]

Algorithmic spectral graph theory / linear algebra

- **Algorithmic linear algebra**
  - $Lx = b$ in nearly-linear time [Spielman, Teng'04], [Koutis, Miller, Peng'10], [Koutis, Miller, Peng'11], [Kelner, Orecchia, Sidford, Zhu'13], [Cohen, Kyng, Miller, Pachocki'14], [Peng, Spielman'14], [Kyng, Lee, Peng, Sachdeva, Spielman'16], [Kyng, Sachdeva'16]
    useful subroutine in other problems, like max-flow
  - $L = \sum_{e \in E} b_e b_e^T$ (for $e = (u, v)$, $b_e = \mathbf{1}_u - \mathbf{1}_v$ for $u < v$)
  - For $x \in \{-1, 1\}^n$, $\frac{1}{4} x^T L x$ is size of cut
  - **Thm:** Can sample only $O(n \lg n)$ edges s.t. new $\tilde{L}$ satisfies $x^T \tilde{L} x \approx x^T L x$ for all $x \in \{-1, 1\}^n$ [Benczúr, Karger'96]
  - In fact can get to hold for all $x \in \mathbb{R}^n$ [Spielman, Srivastava'08], and with $O(n)$ edges [Batson, Spielman, Srivastava'09]

Algorithmic spectral graph theory / linear algebra

- **Algorithmic linear algebra**
  - $Lx = b$ in nearly-linear time [Spielman, Teng'04], [Koutis, Miller, Peng'10], [Koutis, Miller, Peng'11], [Kelner, Orecchia, Sidford, Zhu'13], [Cohen, Kyng, Miller, Pachocki'14], [Peng, Spielman'14], [Kyng, Lee, Peng, Sachdeva, Spielman'16], [Kyng, Sachdeva'16]
    useful subroutine in other problems, like max-flow
  - $L = \sum_{e \in E} b_e b_e^T$ (for $e = (u, v)$, $b_e = \mathbf{1}_u - \mathbf{1}_v$ for $u < v$)
  - For $x \in \{-1, 1\}^n$, $\frac{1}{4} x^T L x$ is size of cut
  - **Thm:** Can sample only $O(n \lg n)$ edges s.t. new $\tilde{L}$ satisfies $x^T \tilde{L} x \approx x^T L x$ for all $x \in \{-1, 1\}^n$ [Benczúr, Karger'96]
  - In fact can get to hold for all $x \in \mathbb{R}^n$ [Spielman, Srivastava'08], and with $O(n)$ edges [Batson, Spielman, Srivastava'09]
  - Ideas eventually led to solution of 57-year old "Kadison-Singer" problem in functional analysis [Marcus, Spielman, Srivastava'15]

Algorithmic spectral graph theory / linear algebra

- **Algorithmic linear algebra**
  - Also, randomized techniques have been developed to quickly solve linear algebra problems in statistics.

Algorithmic spectral graph theory / linear algebra

- **Algorithmic linear algebra**
  - Also, randomized techniques have been developed to quickly solve linear algebra problems in statistics.
  - **Least squares regression:** Given $X \in \mathbb{R}^{n \times d}, y \in \mathbb{R}^n$, compute $\beta^{LS} = argmin \|X\beta - y\|_2^2$
  - **Low-rank approximation:** Given $A \in \mathbb{R}^{n \times d}, k \geq 1$, compute $A_k = argmin_{rank(B) \leq k} \|A - B\|$
  - Classical algorithms solve above in time $O(nd^2)$

Algorithmic spectral graph theory / linear algebra

- **Algorithmic linear algebra**
  - Also, randomized techniques have been developed to quickly solve linear algebra problems in statistics.
  - **Least squares regression:** Given $X \in \mathbb{R}^{n \times d}, y \in \mathbb{R}^n$, compute $\beta^{LS} = argmin \|X\beta - y\|_2^2$
  - **Low-rank approximation:** Given $A \in \mathbb{R}^{n \times d}, k \geq 1$, compute $A_k = argmin_{rank(B) \leq k} \|A - B\|$
  - Classical algorithms solve above in time $O(nd^2)$
  - Modern techniques can get $O(nd \lg n)$, or even $nnz(A) + poly(d)$; "Sketching as a Tool for Numerical Linear Algebra" by Woodruff.

Algorithmic spectral graph theory / linear algebra

- **Algorithmic linear algebra**
  - Also, randomized techniques have been developed to quickly solve linear algebra problems in statistics.
  - **Least squares regression:** Given $X \in \mathbb{R}^{n \times d}, y \in \mathbb{R}^n$, compute $\beta^{LS} = argmin \|X\beta - y\|_2^2$
  - **Low-rank approximation:** Given $A \in \mathbb{R}^{n \times d}, k \geq 1$, compute $A_k = argmin_{rank(B) \leq k} \|A - B\|$
  - Classical algorithms solve above in time $O(nd^2)$
  - Modern techniques can get $O(nd \lg n)$, or even $nnz(A) + poly(d)$; "Sketching as a Tool for Numerical Linear Algebra" by Woodruff.

See CS229r (Algorithms for Big Data), and 6.S978 and 18.409 (An Algorithmist's Toolkit) at MIT.

Areas of TCS

- **Algorithms**
  - Word RAM
  - Graph algorithms
  - Algorithmic spectral graph theory
  - Algorithmic linear algebra
  - Distributed algorithms
  - Parallel algorithms
  - Property testing (like the BLR linearity test)
  - Streaming algorithms
  - Online algorithms
  - Approximation algorithms
  - External-memory / cache-oblivious
  - Computational geometry
  - Number-theoretic problems (e.g. primality testing)
  - ...

Streaming algorithms

A (fake) search engine query log from Nov 7th:

18:58:02	`wikileaks`
18:59:12	`mlb playoffs`
19:07:40	`gmail login`
19:07:42	`gmail`
19:07:58	`p vs np`
19:09:37	`aa flight status 1597`
19:10:14	`halloween costumes`
19:10:18	`gmail`
19:11:28	`gmail`

Streaming algorithms

A (fake) search engine query log from Nov 7th:

18:58:02	`wikileaks`
18:59:12	`mlb playoffs`
19:07:40	**`gmail`** `login`
19:07:42	**`gmail`**
19:07:58	`p vs np`
19:09:37	`aa flight status 1597`
19:10:14	`halloween costumes`
19:10:18	**`gmail`**
19:11:28	**`gmail`**

Finding frequent items

Problem: Given stream of items (e.g. words) coming from some universe $\mathcal{U}$ (e.g. English dictionary), report a small list $L \subset \mathcal{U}$ containing all "frequent" items

- "frequent" depends on some input parameter ε
- stream with m items, "frequent" means appearing $> \varepsilon m$ times

Finding frequent items

Problem: Given stream of items (e.g. words) coming from some universe $\mathcal{U}$ (e.g. English dictionary), report a small list $L \subset \mathcal{U}$ containing all "frequent" items

- "frequent" depends on some input parameter ε
- stream with m items, "frequent" means appearing $> \varepsilon m$ times
- trivial solution: use $n = |\mathcal{U}|$ words of memory

Finding frequent items

Problem: Given stream of items (e.g. words) coming from some universe $\mathcal{U}$ (e.g. English dictionary), report a small list $L \subset \mathcal{U}$ containing all "frequent" items

- "frequent" depends on some input parameter ε
- stream with m items, "frequent" means appearing $> \varepsilon m$ times
- trivial solution: use $n = |\mathcal{U}|$ words of memory
- **Goal:** using $\ll n$ memory, output small such L (e.g. $|L| \leq \frac{10}{\varepsilon}$)

Harder problem: change detection

A (fake) search engine query log from Nov 7th:

18:58:02	`wikileaks`
18:59:12	`mlb playoffs`
19:07:40	`gmail login`
19:07:42	`gmail`
19:07:58	`p vs np`
19:09:37	`aa flight status 1597`
19:10:14	`halloween costumes`
19:10:18	`gmail`
19:11:28	`gmail`

Harder problem: change detection

A (fake) search engine query log from Nov 8th:

18:58:02	`wikileaks`
18:59:12	`mlb playoffs`
19:07:40	`how to move to canada`
19:07:42	`gmail`
19:07:58	`canada immigration`
19:09:37	`aa flight status 1597`
19:10:14	`halloween costumes`
19:10:18	`gmail`
19:11:28	`gmail`
19:13:42	`work visas canada`

Harder problem: change detection

A (fake) search engine query log from Nov 8th:

18:58:02	`wikileaks`
18:59:12	`mlb playoffs`
19:07:40	`how to move to` **`canada`**
19:07:42	`gmail`
19:07:58	**`canada`** `immigration`
19:09:37	`aa flight status 1597`
19:10:14	`halloween costumes`
19:10:18	`gmail`
19:11:28	`gmail`
19:13:42	`work visas` **`canada`**

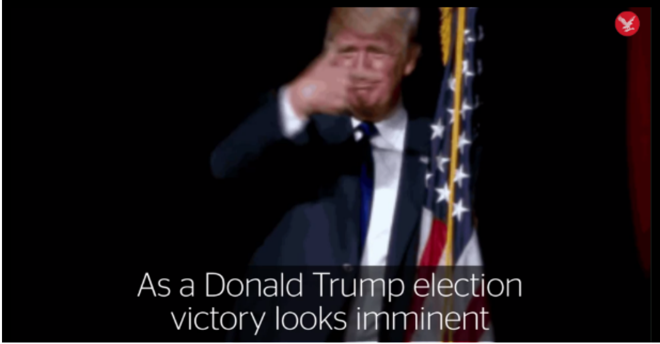
News › World › Americas

How to move to Canada: Immigration website crashes as Donald Trump becomes 45th US President

Moving to live in Canada is not easy but not impossible either

Andrew Griffin | @_andrew_griffin | Tuesday 8 November 2016 | 395 comments

46K shares



Change detection

‘gmail’ popular both days, but big *change* in popularity for ‘canada’

Change detection

‘gmail’ popular both days, but big *change* in popularity for ‘canada’

Goal: use low memory to find small list of items that had large frequency changes.

In general

Streaming algorithms: make one pass over a massive dataset while answering queries, using memory *sublinear* in the data size.

In general

Streaming algorithms: make one pass over a massive dataset while answering queries, using memory *sublinear* in the data size.

Other queries: distinct items, moments, graph streaming (e.g. conncted components in $o(m)$ memory with edge deletions), etc.

In general

Streaming algorithms: make one pass over a massive dataset while answering queries, using memory *sublinear* in the data size.

Other queries: distinct items, moments, graph streaming (e.g. conncted components in $o(m)$ memory with edge deletions), etc.

Take CS229r (Algorithms for Big Data), CS222 (Algorithms at the End of the Wire), or "Sublinear Algorithms" at MIT.

Areas of TCS

- **Algorithms**
  - Word RAM
  - Graph algorithms
  - Algorithmic spectral graph theory
  - Algorithmic linear algebra
  - Distributed algorithms
  - Parallel algorithms
  - Property testing (like the BLR linearity test)
  - Streaming algorithms
  - Online algorithms
  - Approximation algorithms
  - External-memory / cache-oblivious
  - Computational geometry
  - Number-theoretic problems (e.g. primality testing)
  - ...

Online algorithms

- Sequence of irreversible decisions must be made.
- Want to make decisions that aren't too regrettable in hindsight.

Online algorithms

- Sequence of irreversible decisions must be made.
- Want to make decisions that aren't too regrettable in hindsight.
- **Example:** (Ski rental problem). You+friends are on a ski trip. Every morning you vote on whether to keep skiing, or to go home. You can't predict future votes.
- Renting skis costs \$1 and buying costs \$B.

Online algorithms

- Sequence of irreversible decisions must be made.
- Want to make decisions that aren't too regrettable in hindsight.
- **Example:** (Ski rental problem). You+friends are on a ski trip. Every morning you vote on whether to keep skiing, or to go home. You can't predict future votes.
- Renting skis costs \$1 and buying costs \$B.
- If knew #days n: if $n > B$ buy, else rent every day.

Online algorithms

- Sequence of irreversible decisions must be made.
- Want to make decisions that aren't too regrettable in hindsight.
- **Example:** (Ski rental problem). You+friends are on a ski trip. Every morning you vote on whether to keep skiing, or to go home. You can't predict future votes.
- Renting skis costs \$1 and buying costs \$B.
- If knew #days n: if $n > B$ buy, else rent every day.
- But don't know n, so: rent first $B - 1$ days, buy on Bth day. "competitive ratio" $(2B - 1)/B < 2$.

Online algorithms

- Sequence of irreversible decisions must be made.
- Want to make decisions that aren't too regrettable in hindsight.
- **Example:** (Ski rental problem). You+friends are on a ski trip. Every morning you vote on whether to keep skiing, or to go home. You can't predict future votes.
- Renting skis costs \$1 and buying costs \$B.
- If knew #days n: if $n > B$ buy, else rent every day.
- But don't know n, so: rent first $B - 1$ days, buy on Bth day. "competitive ratio" $(2B - 1)/B < 2$.
- $e/(e - 1) < 1.582$ achievable with randomized algorithm

Online algorithms

- Sequence of irreversible decisions must be made.
- Want to make decisions that aren't too regrettable in hindsight.
- **Example:** (Ski rental problem). You+friends are on a ski trip. Every morning you vote on whether to keep skiing, or to go home. You can't predict future votes.
- Renting skis costs \$1 and buying costs \$B.
- If knew #days n: if $n > B$ buy, else rent every day.
- But don't know n, so: rent first $B-1$ days, buy on Bth day.
  "competitive ratio" $(2B-1)/B < 2$.
- $e/(e-1) < 1.582$ achievable with randomized algorithm
- Other problems: e.g. caching (which page to evict?)

Online algorithms

- Sequence of irreversible decisions must be made.
- Want to make decisions that aren't too regrettable in hindsight.
- **Example:** (Ski rental problem). You+friends are on a ski trip. Every morning you vote on whether to keep skiing, or to go home. You can't predict future votes.
- Renting skis costs \$1 and buying costs \$B.
- If knew #days n: if $n > B$ buy, else rent every day.
- But don't know n, so: rent first $B - 1$ days, buy on Bth day. "competitive ratio" $(2B - 1)/B < 2$.
- $e/(e - 1) < 1.582$ achievable with randomized algorithm
- Other problems: e.g. caching (which page to evict?)

Take CS224 (Adv. Algorithms), or 6.854 (Adv. Algorithms) at MIT

Areas of TCS

- **Algorithms**
  - Word RAM
  - Graph algorithms
  - Algorithmic spectral graph theory
  - Algorithmic linear algebra
  - Distributed algorithms
  - Parallel algorithms
  - Property testing (like the BLR linearity test)
  - Streaming algorithms
  - Online algorithms
  - Approximation algorithms
  - External-memory / cache-oblivious
  - Computational geometry
  - Number-theoretic problems (e.g. primality testing)
  - ...

External-memory / cache-oblivious

- **Fact:** Touching data in memory is $\approx 10^6$-times faster than seeking to a random location on disk.

External-memory / cache-oblivious

- **Fact:** Touching data in memory is $\approx 10^6$-times faster than seeking to a random location on disk.
- **Model:** Touching memory is free, but touching disk costs 1. Minimize cost. Also, sequential read on disk much faster than seeks, so allow reading B data items per data transfer.
- Memory is bounded size. Disk is infinite.

External-memory / cache-oblivious

- **Fact:** Touching data in memory is $\approx 10^6$-times faster than seeking to a random location on disk.
- **Model:** Touching memory is free, but touching disk costs 1. Minimize cost. Also, sequential read on disk much faster than seeks, so allow reading B data items per data transfer.
- Memory is bounded size. Disk is infinite.
- **Example:** for predecessor, rather than use balanced BST, better to use tree with branching factor $\Theta(B)$; "B-tree".

External-memory / cache-oblivious

- **Fact:** Touching data in memory is $\approx 10^6$-times faster than seeking to a random location on disk.
- **Model:** Touching memory is free, but touching disk costs 1. Minimize cost. Also, sequential read on disk much faster than seeks, so allow reading B data items per data transfer.
- Memory is bounded size. Disk is infinite.
- **Example:** for predecessor, rather than use balanced BST, better to use tree with branching factor $\Theta(B)$; "B-tree".
- **Another example:** M/B-way mergesort (split into M/B arrays and recursively sort then merge). Can prove optimal.

External-memory / cache-oblivious

- **Fact:** Touching data in memory is $\approx 10^6$-times faster than seeking to a random location on disk.
- **Model:** Touching memory is free, but touching disk costs 1. Minimize cost. Also, sequential read on disk much faster than seeks, so allow reading B data items per data transfer.
- Memory is bounded size. Disk is infinite.
- **Example:** for predecessor, rather than use balanced BST, better to use tree with branching factor $\Theta(B)$; "B-tree".
- **Another example:** M/B-way mergesort (split into M/B arrays and recursively sort then merge). Can prove optimal.
- Cache-obliviousness [Frigo, Leiserson, Prokop, Ramachandran'99]: get good performance *but algorithm not allowed to know* B, M

External-memory / cache-oblivious

- **Fact:** Touching data in memory is $\approx 10^6$-times faster than seeking to a random location on disk.
- **Model:** Touching memory is free, but touching disk costs 1. Minimize cost. Also, sequential read on disk much faster than seeks, so allow reading B data items per data transfer.
- Memory is bounded size. Disk is infinite.
- **Example:** for predecessor, rather than use balanced BST, better to use tree with branching factor $\Theta(B)$; "B-tree".
- **Another example:** M/B-way mergesort (split into M/B arrays and recursively sort then merge). Can prove optimal.
- Cache-obliviousness [Frigo, Leiserson, Prokop, Ramachandran'99]: get good performance *but algorithm not allowed to know* B, M

See book by Vitter "Algorithms and Data Structures for External Memory". CS229r (Algorithms for Big Data), and 6.851 at MIT.

Areas of TCS

- **Data structures**
  - Many qualifiers: amortized/worst case, static/dynamic, persistent/ephemeral, randomized/deterministic
  - Tradeoffs (time vs. space, or update vs. query time), upper and lower bounds
- **Complexity Theory**
  - Concrete complexity (communication complexity, branching programs, circuits, formulae, ...)
  - Pseudorandomness
  - Algebraic complexity
  - Proof complexity
  - Interactive proof systems
- **Computer Science + Economics**
  - Algorithmic mechanism design
  - Algorithmic game theory
- **Cryptography and Privacy**

Areas of TCS

- **Data structures**
  - Many qualifiers: amortized/worst case, static/dynamic, persistent/ephemeral, randomized/deterministic
  - Tradeoffs (time vs. space, or update vs. query time), upper and lower bounds
- **Complexity Theory**
  - Concrete complexity (communication complexity, branching programs, circuits, formulae, ...)
  - Pseudorandomness
  - Algebraic complexity
  - Proof complexity
  - Interactive proof systems
- **Computer Science + Economics**
  - Algorithmic mechanism design
  - Algorithmic game theory
- **Cryptography and Privacy**

Communication complexity

- **Simplest setup:** Two parties, Alice and Bob.
- Alice gets $x \in \mathcal{X}$, Bob gets $y \in \mathcal{Y}$
  both know $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ and want to compute $f(x,y)$

Communication complexity

- **Simplest setup:** Two parties, Alice and Bob.
- Alice gets $x \in \mathcal{X}$, Bob gets $y \in \mathcal{Y}$

  both know $f : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$ and want to compute $f(x, y)$
- Allowed to send messages back and forth, but want to minimize total bits communicated, or number of rounds.

  (trivial: $\lg|\mathcal{X}|$ or $\lg|\mathcal{Y}|$ comm.)

Communication complexity

- **Simplest setup:** Two parties, Alice and Bob.
- Alice gets $x \in \mathcal{X}$, Bob gets $y \in \mathcal{Y}$
  both know $f : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$ and want to compute $f(x, y)$
- Allowed to send messages back and forth, but want to minimize total bits communicated, or number of rounds.
  (trivial: $\lg |\mathcal{X}|$ or $\lg |\mathcal{Y}|$ comm.)
- **Applications:** Circuit depth lower bounds ("Karchmer-Wigderson games"), streaming space lower bounds, data structure lower bounds (cell probe model, pset4), ...

Communication complexity

- **Simplest setup:** Two parties, Alice and Bob.
- Alice gets $x \in \mathcal{X}$, Bob gets $y \in \mathcal{Y}$

  both know $f : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$ and want to compute $f(x, y)$
- Allowed to send messages back and forth, but want to minimize total bits communicated, or number of rounds.

  (trivial: $\lg |\mathcal{X}|$ or $\lg |\mathcal{Y}|$ comm.)
- **Applications:** Circuit depth lower bounds ("Karchmer-Wigderson games"), streaming space lower bounds, data structure lower bounds (cell probe model, pset4), ...

See "Communication complexity" book by Kushilevitz and Nisan. Also CS229r (Information Theory in Computer Science).

Areas of TCS

- **Data structures**
  - Many qualifiers: amortized/worst case, static/dynamic, persistent/ephemeral, randomized/deterministic
  - Tradeoffs (time vs. space, or update vs. query time), upper and lower bounds
- **Complexity Theory**
  - Concrete complexity (communication complexity, branching programs, circuits, formulae, ...)
  - Pseudorandomness
  - Algebraic complexity
  - Proof complexity
  - Interactive proof systems
- **Computer Science + Economics**
  - Algorithmic mechanism design
  - Algorithmic game theory
- **Cryptography and Privacy**

Pseudorandomness

- Understand how to generate few many "good enough" random bits from very few truly random bits.
- **Example:** universal hash family mapping $[u]$ to $[m]$. Truly random hash function needs $O(u \lg m)$ bits, but random function from universal family only needs $O(\lg(um))$.

Pseudorandomness

- Understand how to generate few many "good enough" random bits from very few truly random bits.
- **Example:** universal hash family mapping $[u]$ to $[m]$. Truly random hash function needs $O(u \lg m)$ bits, but random function from universal family only needs $O(\lg(um))$.
- Is even the small bit of randomness necessary? **RP** = **P**? **BPP** = **P**?

Pseudorandomness

- Understand how to generate few many "good enough" random bits from very few truly random bits.
- **Example:** universal hash family mapping $[u]$ to $[m]$. Truly random hash function needs $O(u \lg m)$ bits, but random function from universal family only needs $O(\lg(um))$.
- Is even the small bit of randomness necessary? $\mathbf{RP} = \mathbf{P}$? $\mathbf{BPP} = \mathbf{P}$?
- Define $\mathbf{L}$ to be class of languages solvable by log-space Turing machines (space-$s(n)$ means 2 tapes: one is input read-only, and the second tape is work tape; should use at most first $s(n)$ cells on work tape).
- $\mathbf{RL} = \mathbf{L}$? $\mathbf{BPL} = \mathbf{L}$? Best known: $\mathbf{BPL} \subseteq \mathbf{L}^{3/2}$ [Saks, Zhou'95]

Pseudorandomness

- Understand how to generate few many "good enough" random bits from very few truly random bits.
- **Example:** universal hash family mapping $[u]$ to $[m]$. Truly random hash function needs $O(u \lg m)$ bits, but random function from universal family only needs $O(\lg(um))$.
- Is even the small bit of randomness necessary? $\mathbf{RP} = \mathbf{P}$? $\mathbf{BPP} = \mathbf{P}$?
- Define $\mathbf{L}$ to be class of languages solvable by log-space Turing machines (space-$s(n)$ means 2 tapes: one is input read-only, and the second tape is work tape; should use at most first $s(n)$ cells on work tape).
- $\mathbf{RL} = \mathbf{L}$? $\mathbf{BPL} = \mathbf{L}$? Best known: $\mathbf{BPL} \subseteq \mathbf{L}^{3/2}$ [Saks, Zhou'95]

Take CS225.

Areas of TCS

- **Data structures**
  - Many qualifiers: amortized/worst case, static/dynamic, persistent/ephemeral, randomized/deterministic
  - Tradeoffs (time vs. space, or update vs. query time), upper and lower bounds
- **Complexity Theory**
  - Concrete complexity (communication complexity, branching programs, circuits, formulae, ...)
  - Pseudorandomness
  - Algebraic complexity
  - Proof complexity
  - Interactive proof systems
- **Computer Science + Economics**
  - Algorithmic mechanism design
  - Algorithmic game theory
- **Cryptography and Privacy**

Algebraic circuit complexity

- Circuits, but gates are $+$ and $\times$ instead of AND, OR, NOT.
- Want to compute some function $\mathbf{F}^n \to \mathbf{F}$ ($\mathbf{F}$ a field, like $\mathbb{R}$).

Algebraic circuit complexity

- Circuits, but gates are $+$ and $\times$ instead of AND, OR, NOT.
- Want to compute some function $\mathbf{F}^n \to \mathbf{F}$ ($\mathbf{F}$ a field, like $\mathbb{R}$).
- [Valiant'79]: **VP** are functions f of $poly(n)$ degree with poly-size circuits. **VNP** are those s.t. for any monomial, there is a poly-size circuit to compute the coefficient of that monomial.

Algebraic circuit complexity

- Circuits, but gates are $+$ and $\times$ instead of AND, OR, NOT.
- Want to compute some function $\mathbf{F}^n \to \mathbf{F}$ ($\mathbf{F}$ a field, like $\mathbb{R}$).
- [Valiant'79]: **VP** are functions f of $poly(n)$ degree with poly-size circuits. **VNP** are those s.t. for any monomial, there is a poly-size circuit to compute the coefficient of that monomial.
- **Thm.** Permanent of a matrix (viewed as a function on n^2 numbers) is **VNP**-complete.
- still unknown if permanent has poly-size circuits

Algebraic circuit complexity

- Circuits, but gates are $+$ and $\times$ instead of AND, OR, NOT.
- Want to compute some function $\mathbf{F}^n \to \mathbf{F}$ ($\mathbf{F}$ a field, like $\mathbb{R}$).
- [Valiant'79]: **VP** are functions f of $poly(n)$ degree with poly-size circuits. **VNP** are those s.t. for any monomial, there is a poly-size circuit to compute the coefficient of that monomial.
- **Thm.** Permanent of a matrix (viewed as a function on n^2 numbers) is **VNP**-complete.
- still unknown if permanent has poly-size circuits
- Determinant(A): $\sum_{\sigma \in S_n} sgn(\sigma) \cdot \prod_{i=1}^{n} A_{i,\sigma(i)}$.
- Permanent(A): $\sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$.

Areas of TCS

- **Data structures**
  - Many qualifiers: amortized/worst case, static/dynamic, persistent/ephemeral, randomized/deterministic
  - Tradeoffs (time vs. space, or update vs. query time), upper and lower bounds
- **Complexity Theory**
  - Concrete complexity (communication complexity, branching programs, circuits, formulae, . . .)
  - Pseudorandomness
  - Algebraic complexity
  - Proof complexity
  - Interactive proof systems
- **Computer Science + Economics**
  - Algorithmic mechanism design
  - Algorithmic game theory
- **Cryptography and Privacy**

Proof complexity

- Given axiomatic system, want to understand not just its power (what can be proven and what can't), but how difficult it is to prove some specific theorem in that axiomatic system.
- e.g. minimum length of a proof?

Proof complexity

- Given axiomatic system, want to understand not just its power (what can be proven and what can't), but how difficult it is to prove some specific theorem in that axiomatic system.
- e.g. minimum length of a proof?
- Recently highly applicable: sums of squares proofs.
- **Thm.** [Artin'27], [Krivine'64], [Stengle'74]. Let $P_1, \ldots, P_m$ be n-variate polynomials with real coefficients. Then the system of equations $P_1(X) = \cdots = P_m(x) = 0$ has no solution over $\mathbb{R}^n$ iff there exist polynomials $Q_1, \ldots, Q_m$, and some polynomial S expressible as a sum of squares, s.t. $-1 = S + \sum_{i=1}^m Q_i P_i$.
- "Complexity" of proof defined as max degree ℓ of the $Q_i P_i$. Can find low-complexty proofs algorithmically.

Proof complexity

- Given axiomatic system, want to understand not just its power (what can be proven and what can't), but how difficult it is to prove some specific theorem in that axiomatic system.
- e.g. minimum length of a proof?
- Recently highly applicable: sums of squares proofs.
- **Thm.** [Artin'27], [Krivine'64], [Stengle'74]. Let $P_1, \ldots, P_m$ be n-variate polynomials with real coefficients. Then the system of equations $P_1(X) = \cdots = P_m(x) = 0$ has no solution over $\mathbb{R}^n$ iff there exist polynomials $Q_1, \ldots, Q_m$, and some polynomial S expressible as a sum of squares, s.t. $-1 = S + \sum_{i=1}^{m} Q_i P_i$.
- "Complexity" of proof defined as max degree ℓ of the $Q_i P_i$. Can find low-complexty proofs algorithmically.

Take "Proofs, beliefs and algorithms through the lens of Sum of Squares" at Harvard/MIT.

Areas of TCS

- **Data structures**
  - Many qualifiers: amortized/worst case, static/dynamic, persistent/ephemeral, randomized/deterministic
  - Tradeoffs (time vs. space, or update vs. query time), upper and lower bounds
- **Complexity Theory**
  - Concrete complexity (communication complexity, branching programs, circuits, formulae, ...)
  - Pseudorandomness
  - Algebraic complexity
  - Proof complexity
  - Interactive proof systems
- **Computer Science + Economics**
  - Algorithmic mechanism design
  - Algorithmic game theory
- **Cryptography and Privacy**

CS+Econ

- Game theory $\cap$ (Algorithms $\cup$ Complexity)
  - Two-player zero-sum games provably have equillibrium strategies ("existence"). But the complexity of finding one? **CS125:** polynomial time (linear programming)

CS+Econ

- Game theory $\cap$ (Algorithms $\cup$ Complexity)
  - Two-player zero-sum games provably have equillibrium strategies ("existence"). But the complexity of finding one? **CS125:** polynomial time (linear programming)
  - Other games? What about non-zero sum? (if I get $+x$ dollars, you don't necessarily get $-x$)
  - **Nash'50, '51:** Equillibrium exists even if not zero-sum. Proofs via fixed-point theorems in topology.

CS+Econ

- Game theory $\cap$ (Algorithms $\cup$ Complexity)
  - Two-player zero-sum games provably have equillibrium strategies ("existence"). But the complexity of finding one? **CS125:** polynomial time (linear programming)
  - Other games? What about non-zero sum? (if I get $+x$ dollars, you don't necessarily get $-x$)
  - **Nash'50, '51:** Equillibrium exists even if not zero-sum. Proofs via fixed-point theorems in topology.
  - can we find equillibrium efficiently, algorithmically? n players?

    "If your laptop can't find it, then neither can the market." — Kamal Jain

CS+Econ

- Game theory $\cap$ (Algorithms $\cup$ Complexity)
  - Two-player zero-sum games provably have equillibrium strategies ("existence"). But the complexity of finding one? **CS125:** polynomial time (linear programming)
  - Other games? What about non-zero sum? (if I get $+x$ dollars, you don't necessarily get $-x$)
  - **Nash'50, '51:** Equillibrium exists even if not zero-sum. Proofs via fixed-point theorems in topology.
  - can we find equillibrium efficiently, algorithmically? n players?

    "If your laptop can't find it, then neither can the market." — Kamal Jain
  - **Thm.** Finding equilibrium even in 2-player games is **PPAD**-complete [Goldberg, Daskalakis, Papadimitriou'06], [Cheng, Deng'06]

CS+Econ

- Game theory $\cap$ (Algorithms $\cup$ Complexity)
  - Two-player zero-sum games provably have equillibrium strategies ("existence"). But the complexity of finding one? **CS125:** polynomial time (linear programming)
  - Other games? What about non-zero sum? (if I get $+x$ dollars, you don't necessarily get $-x$)
  - **Nash'50, '51:** Equillibrium exists even if not zero-sum. Proofs via fixed-point theorems in topology.
  - can we find equillibrium efficiently, algorithmically? n players?
    "If your laptop can't find it, then neither can the market." — Kamal Jain
  - **Thm.** Finding equilibrium even in 2-player games is **PPAD**-complete [Goldberg, Daskalakis, Papadimitriou'06], [Cheng, Deng'06]
  - **PPAD:** The class for which the following problem is complete: given digraph G implicitly s.t. each vertex has at most one outgoing and at most one incoming edge. Given $s \in V(G)$ and a *description* of a poly-time computable function $f(v)$ to compute successors/predecessors, find a sink. [Papadimitriou'94]

CS+Econ

- Networks
  - Given the Facebook graph how do you choose who to show ads to in order to make your product go viral?
  - Given interaction graph, how do you predict disease spread?
- Algorithmic mechanism design
  - Simplest setup: single-item, multiple-bidder auction
  - Bidders submit sealed bids, then run a procedure ("mechanism") to figure out who gets item and at what price

CS+Econ

- Networks
  - Given the Facebook graph how do you choose who to show ads to in order to make your product go viral?
  - Given interaction graph, how do you predict disease spread?
- Algorithmic mechanism design
  - Simplest setup: single-item, multiple-bidder auction
  - Bidders submit sealed bids, then run a procedure ("mechanism") to figure out who gets item and at what price
  - Can get more complicated, e.g. Google AdWords (many items i.e. "keywords", many bidders, and winners are ranked i.e. order ads are displayed).

CS+Econ

- Networks
  - Given the Facebook graph how do you choose who to show ads to in order to make your product go viral?
  - Given interaction graph, how do you predict disease spread?
- Algorithmic mechanism design
  - Simplest setup: single-item, multiple-bidder auction
  - Bidders submit sealed bids, then run a procedure ("mechanism") to figure out who gets item and at what price
  - Can get more complicated, e.g. Google AdWords (many items i.e. "keywords", many bidders, and winners are ranked i.e. order ads are displayed).
  - Businesses: Maximize revenue?
  - Government: Maximize "social welfare" (total happiness)?

Take CS134 (Networks), CS284r (Social Data Mining), CS284r (Incentives and Information in Networks). See more at `econcs.seas.harvard.edu`. Also 6.891 (Topics in AGT) and 6.853 (Games, Decision, and Computation) at MIT.

Areas of TCS

- **Data structures**
  - Many qualifiers: amortized/worst case, static/dynamic, persistent/ephemeral, randomized/deterministic
  - Tradeoffs (time vs. space, or update vs. query time), upper and lower bounds
- **Complexity Theory**
  - Concrete complexity (communication complexity, branching programs, circuits, formulae, ...)
  - Pseudorandomness
  - Algebraic complexity
  - Proof complexity
  - Interactive proof systems
- **Computer Science + Economics**
  - Algorithmic mechanism design
  - Algorithmic game theory
- **Cryptography and Privacy**

Cryptography and Privacy

- Cryptography
  - **Private-key**: Alice+Bob share secret, communicate covertly.
  - **Public-key**: Alice has public key visible to everyone. Anyone can send her a message, but then only Alice can decrypt it. (**assumption:** some problems are computationally hard, and hackers are computationally bounded)

Cryptography and Privacy

- Cryptography
  - **Private-key**: Alice+Bob share secret, communicate covertly.
  - **Public-key**: Alice has public key visible to everyone. Anyone can send her a message, but then only Alice can decrypt it. (**assumption:** some problems are computationally hard, and hackers are computationally bounded)
  - **Authentication**

Cryptography and Privacy

- Cryptography
  - **Private-key**: Alice+Bob share secret, communicate covertly.
  - **Public-key**: Alice has public key visible to everyone. Anyone can send her a message, but then only Alice can decrypt it. (**assumption:** some problems are computationally hard, and hackers are computationally bounded)
  - **Authentication**
  - **Zero-knowledge proofs.** Proves a mathematical statement to you so that you learn nothing more than the truth of the statement. e.g. graph non-isomorphism.

Cryptography and Privacy

- Cryptography
  - **Private-key**: Alice+Bob share secret, communicate covertly.
  - **Public-key**: Alice has public key visible to everyone. Anyone can send her a message, but then only Alice can decrypt it. (**assumption:** some problems are computationally hard, and hackers are computationally bounded)
  - **Authentication**
  - **Zero-knowledge proofs.** Proves a mathematical statement to you so that you learn nothing more than the truth of the statement. e.g. graph non-isomorphism.
  - **Fully homomorphic encryption.** Have data but lack computational power (e.g. Freivalds). Want cloud to compute $f(x_1, \ldots, x_n)$ for us, but we don't want to send $x_1, \ldots, x_n$. Can we send encrypted data for cloud to compute on, sending back encrypted answer, never learning the x_i? Yes! [Gentry'09]

Cryptography and Privacy

- Cryptography
  - **Private-key**: Alice+Bob share secret, communicate covertly.
  - **Public-key**: Alice has public key visible to everyone. Anyone can send her a message, but then only Alice can decrypt it. (**assumption:** some problems are computationally hard, and hackers are computationally bounded)
  - **Authentication**
  - **Zero-knowledge proofs.** Proves a mathematical statement to you so that you learn nothing more than the truth of the statement. e.g. graph non-isomorphism.
  - **Fully homomorphic encryption.** Have data but lack computational power (e.g. Freivalds). Want cloud to compute $f(x_1, \ldots, x_n)$ for us, but we don't want to send $x_1, \ldots, x_n$. Can we send encrypted data for cloud to compute on, sending back encrypted answer, never learning the x_i? Yes! [Gentry'09]
  - **Security multiparty computation**, **Functional encryption**, **Private information retrieval** ...

Cryptography and Privacy

- Privacy
  - Sensitive dataset, e.g. hospital records.

Cryptography and Privacy

- Privacy
  - Sensitive dataset, e.g. hospital records.
  - Patients want privacy (and it's the law). Biostatisticians and other researchers want to query dataset to learn facts beneficial to humanity: how many patients with initial complaint X wind up with complication Y? Given record of hospital visits so far, can we predict when patient will next visit hospital? etc.

Cryptography and Privacy

- Privacy
  - Sensitive dataset, e.g. hospital records.
  - Patients want privacy (and it's the law). Biostatisticians and other researchers want to query dataset to learn facts beneficial to humanity: how many patients with initial complaint X wind up with complication Y? Given record of hospital visits so far, can we predict when patient will next visit hospital? etc.
  - **Tension** between privacy and utility when releasing dataset.

Cryptography and Privacy

- Privacy
  - Sensitive dataset, e.g. hospital records.
  - Patients want privacy (and it's the law). Biostatisticians and other researchers want to query dataset to learn facts beneficial to humanity: how many patients with initial complaint X wind up with complication Y? Given record of hospital visits so far, can we predict when patient will next visit hospital? etc.
  - **Tension** between privacy and utility when releasing dataset.
  - **Differential privacy.** Formal statement about when a randomized mechanism provides privacy, and to what extent: "neighboring" datasets should have nearly indistinguishable output distributions [Dwork, McSherry, Nissim, Smith'06]. (Also see recent book by Cynthia Dwork and Aaron Roth.)

Cryptography and Privacy

- Privacy
  - Sensitive dataset, e.g. hospital records.
  - Patients want privacy (and it's the law). Biostatisticians and other researchers want to query dataset to learn facts beneficial to humanity: how many patients with initial complaint X wind up with complication Y? Given record of hospital visits so far, can we predict when patient will next visit hospital? etc.
  - **Tension** between privacy and utility when releasing dataset.
  - **Differential privacy.** Formal statement about when a randomized mechanism provides privacy, and to what extent: "neighboring" datasets should have nearly indistinguishable output distributions [Dwork, McSherry, Nissim, Smith'06]. (Also see recent book by Cynthia Dwork and Aaron Roth.)

Take CS127 (Intro to Crypto), CS227r (Topics in Crypto+Privacy), CS229r (Mathematical Approaches to Data Privacy). Also 6.875 (Crypto) and 6.876 (Topics in Crypto) at MIT.

Areas of TCS

- **Computational Learning Theory**
- **Coding Theory**
- **Quantum Computing**
  - Quantum error-correction
  - Quantum communication complexity
  - Quantum complexity ("quantum Turing machines")
  - Quantum algorithms

Computational Learning Theory

- **PAC learning.**
- Distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, and "concept class" $\mathcal{F}$ which is subset of functions from $\mathcal{X}$ to $\mathcal{Y}$.
- Given many iid samples from $\mathcal{D}$, "learn" the best $f \in \mathcal{F}$ minimizing $\mathbb{P}_{(x,y)\sim\mathcal{D}}(f(x) \neq y)$ [Valiant'84]

Computational Learning Theory

- **PAC learning.**
- Distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, and "concept class" $\mathcal{F}$ which is subset of functions from $\mathcal{X}$ to $\mathcal{Y}$.
- Given many iid samples from $\mathcal{D}$, "learn" the best $f \in \mathcal{F}$ minimizing $\mathbb{P}_{(x,y)\sim\mathcal{D}}(f(x) \neq y)$ [Valiant'84]
- Proper learning: $\mathcal{D}$ always gives $(x, f(x))$ for *some* $f \in \mathcal{F}$
- Improper: not proper, but still want to find an $f \in \mathcal{F}$ that is nearly the best in $\mathcal{F}$

Computational Learning Theory

- **PAC learning.**
- Distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, and "concept class" $\mathcal{F}$ which is subset of functions from $\mathcal{X}$ to $\mathcal{Y}$.
- Given many iid samples from $\mathcal{D}$, "learn" the best $f \in \mathcal{F}$ minimizing $\mathbb{P}_{(x,y)\sim\mathcal{D}}(f(x) \neq y)$ [Valiant'84]
- Proper learning: $\mathcal{D}$ always gives $(x, f(x))$ for *some* $f \in \mathcal{F}$
- Improper: not proper, but still want to find an $f \in \mathcal{F}$ that is nearly the best in $\mathcal{F}$
- **Statisical query model.** Weaker than general PAC learning: learner only allowed to make statistical queries to $\mathcal{D}$ (i.e. can ask oracle some query $\phi : \mathcal{X} \to [-1, 1]$ and get back an estimate of $\mathbb{E}_{x\sim\mathcal{D}}\, \phi(x))$ [Kearns'98].

CS228 (Computational Learning Theory). Also, book by Kearns and Vazirani.

Areas of TCS

- **Computational Learning Theory**
- **Coding Theory**
- **Quantum Computing**
  - Quantum error-correction
  - Quantum communication complexity
  - Quantum complexity ("quantum Turing machines")
  - Quantum algorithms

Quantum Computing

- How do quantum effects affect our study of computation in the physical universe?

Quantum Computing

- How do quantum effects affect our study of computation in the physical universe?
- Cryptography
  - **Quantum money.** Bank notes that are guaranteed to be unforgeable assuming laws of quantum mechanics [Wiesner'83], several later papers by Scott Aaronson and collaborators.
  - **Quantum key distribution.** Alice+Bob share secret keys via public communication; eavesdropping adversary Eve learns nothing [Bennet, Brassard'84], [Ekert'91].

Quantum Computing

- Complexity
  - **Quantum circuits.** Take input that is n "qubits", not bits. n qubits just means a vector $x \in \mathbb{C}^{2^n}$ with $\sum_i |x_i|^2 = 1$ (so probability distribution over $\{0,1\}^n$). Allowed to do unitary operations on x, i.e. $x \mapsto Ux$ for $U^*U = UU^* = I$. Can "measure" and collapse to a basis state given this probability distribution.
  - Basis for classical circuits: AND, OR, NOT. For quantum: gates apply unitary matrix to input; has been shown a finite set of gate types suffices to do quantum computation. (universal set of gates S satisfies $\forall n \geq n_0,$ subgroup generated by S is dense in group of unitary matrices with determinant 1 operating on n qubits)

Quantum Computing

- Complexity
  - **Quantum circuits.** Take input that is n "qubits", not bits. n qubits just means a vector $x \in \mathbb{C}^{2^n}$ with $\sum_i |x_i|^2 = 1$ (so probability distribution over $\{0,1\}^n$). Allowed to do unitary operations on x, i.e. $x \mapsto Ux$ for $U^*U = UU^* = I$. Can "measure" and collapse to a basis state given this probability distribution.
  - Basis for classical circuits: AND, OR, NOT. For quantum: gates apply unitary matrix to input; has been shown a finite set of gate types suffices to do quantum computation. (universal set of gates S satisfies $\forall n \geq n_0,$ subgroup generated by S is dense in group of unitary matrices with determinant 1 operating on n qubits)
  - **BQP**. Languages decided by uniform poly-size quantum circuits. $\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{BQP} \subseteq \mathbf{PP} \subseteq \mathbf{PSPACE}$.

Quantum Computing

- Complexity
  - **Quantum circuits.** Take input that is n "qubits", not bits. n qubits just means a vector $x \in \mathbb{C}^{2^n}$ with $\sum_i |x_i|^2 = 1$ (so probability distribution over $\{0,1\}^n$). Allowed to do unitary operations on x, i.e. $x \mapsto Ux$ for $U^*U = UU^* = I$. Can "measure" and collapse to a basis state given this probability distribution.
  - Basis for classical circuits: AND, OR, NOT. For quantum: gates apply unitary matrix to input; has been shown a finite set of gate types suffices to do quantum computation. (universal set of gates S satisfies $\forall n \geq n_0,$ subgroup generated by S is dense in group of unitary matrices with determinant 1 operating on n qubits)
  - **BQP**. Languages decided by uniform poly-size quantum circuits. $\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{BQP} \subseteq \mathbf{PP} \subseteq \mathbf{PSPACE}$.
  - Some problems known to be in **BQP** but unknown if in **BPP**, e.g. FACTORING [Shor'94].

Quantum Computing

- Complexity
  - **Quantum communication complexity.** How much more efficiently can we solve communication problems when Alice+Bob share quantumly entangled qubits?

Quantum Computing

- Complexity
  - **Quantum communication complexity.** How much more efficiently can we solve communication problems when Alice+Bob share quantumly entangled qubits?
  - **Quantum search.** Clasically need $\Theta(n)$ time to find an item in an unordered database of size n. Quantumly, with the right query model, Grover's algorithm only needs $O(\sqrt{n})$ queries [Grover'96], which is asymptotically optimal [Bennett, Bernstein, Brassad, Vazirani'97].

Quantum Computing

- Complexity
  - **Quantum communication complexity.** How much more efficiently can we solve communication problems when Alice+Bob share quantumly entangled qubits?
  - **Quantum search.** Clasically need $\Theta(n)$ time to find an item in an unordered database of size n. Quantumly, with the right query model, Grover's algorithm only needs $O(\sqrt{n})$ queries [Grover'96], which is asymptotically optimal [Bennett, Bernstein, Brassad, Vazirani'97].
  - **Quantum error-correction.** Need to store data being computed on robustly due to quantum decoherence and faulty gates. Some physical barriers though ("no-cloning theorem" [Wootters, Zurek'82], [Dieks'82])
  - ...

The End

See more TCS courses at:

- http://toc.seas.harvard.edu/toc-courses
- http://econcs.seas.harvard.edu/teaching
- http://toc.csail.mit.edu/

The End

See more TCS courses at:

- `http://toc.seas.harvard.edu/toc-courses`
- `http://econcs.seas.harvard.edu/teaching`
- `http://toc.csail.mit.edu/`
- Also: AM 106/206 (Applied Algebra), AM 107 (Graph Theory & Combinatorics), MATH 155r (Combinatorics), ES 250 (Information Theory), MATH 141 (Intro to Mathematical Logic), Philosophy 144 (Logic & Philosophy)
- Connections to other areas: natural language processing, circuit design, parsing and compiling, programming languages, artificial intelligence ...

And check out our seminars / explore research opportunities!

`http://toc.seas.harvard.edu/events-seminars`