CSC373: Algorithm Design, Analysis and Complexity
Winter/Spring 2020

Allan Borodin and Sara Rahmati

Week of March 30-April 3, 2020

Week 12 : Announcements

- We will continue to provide information during this period on the web page, on piazza and on these lecture slides. We will also hold office hours via skype. My skype address is abborodin@gmail.com.
  I am also holding office hours via zoom on Wednesdays at 3PM. This is a slot reserved for CSC303 lectures and tutorials so you need the 303 zoom site password. It is 038954
- Assignment 3 is due Thursday, April 2 at 4:59.
- Assignment 4 is due Thursday, April 16 at 4:59. A fifth and final question is being added to Assignment 4.
- In many cases, students are answering questions correctly on piazza and we appreciate that being done.
- While we are always concerned about plagarism, we are particularly concerned due to the additional pressure we are all feeling.

This weeks agenda

1. The complexity of k-SAT and random walk algorithms for k-SAT.
2. Randomized primality testing
3. Public key cryptography and the RSA method
4. Topics beyond scope of CSC373
    - Fine grained complexity
    - Fixed parameter tractable (FPT) problems
    - Sublinear time algorithms
    - The streaming model

What is the complexity of k-SAT?

- It is not difficult to show that 2-SAT (i.e., determining if a 2CNF formula is satisfiable) is efficiently deterministically solvable in polynomial time. This is in contrast to 3-SAT being NP complete.
- However, it turns out that Max-2-SAT is NP-hard.
- We will see that for 2-SAT there is also a conceptually simple 1-sided error randomized algorithm (based on random walks) running in time $O(n^2)$ that shows that 2-SAT is computationally easy. There is also a deterministic polynomial time algorithm for 2-SAT.
- The same basic random walk approach can be used to derive a randomized 1-sided error algorithm (which in turn has a deterministic variant) for 3-SAT that runs in time $(1.324)^n$. This is, of course, still exponential but significantly better that 2^n. And this can be extended to k-SAT for any fixed k.
- **The exponential time hypothesis (ETH)**: There is no deterministic or randomized algorithm for 3-SAT running in time $2^{o(n)}$. This is an unproven conjecture even assuming $P \neq NP$.

Random walk algorithms for 2-Sat and k-Sat

- First, here is the idea of the deterministic polynomial time algorithm for 2-Sat: We can first eliminate all unit clauses. We then reduce the problem to the directed $s - t$ path problem. We view each clause $(x \vee y)$ in F as two directed edges $(\bar{x}, y)$ and $(\bar{y}, x)$ in a graph G_F whose nodes are all possible literals x and $\bar{x}$. Then the formula is satisfiable iff there does not exist a variable x such that there are paths from x to $\bar{x}$ and from $\bar{x}$ to x in G_F.
- There is also a randomized algorithm for 2-SAT (due to Papadimitriou [1991]) based on a random walk on the line graph with nodes $\{0, 1, \ldots, n\}$. We view being on node i as having a truth assignment τ that is Hamming distance i from some fixed satisfying assignment τ^* if such an assignment exists (i.e. F is satisfiable).
- Start with an arbitrary truth assignment τ and if $F(\tau)$ is true then we are done; else find an arbitrary unsatisfied clause C and randomly choose one of the two variables x_i occurring in C and now change τ to τ' by setting $\tau'(x_i) = 1 - \tau(x_i)$.

The expected time to reach a satisfying assignment

- When we randomly select one the the two literals in C and complement it, we are getting close to τ^* (i.e. moving one edge closer to node 0 on the line) with probability at least $\frac{1}{2}$. (If it turns out that both literal values disagree with τ^*, then we are getting closer to τ^* with probability $= 1$.)
- As we are proceeding in this random walk we might encounter another satisfying assignment which is all the better.
- It remains to bound the expected time to reach node 0 in a random walk on the line where on each random step, the distance to node 0 is reduced by 1 with probability at least $\frac{1}{2}$ and otherwise increased by 1 (but never exceeding distance n). This perhaps biased random walk is at least as good as the case where we randomly increase or decrease the distance by 1 with probability equal to $\frac{1}{2}$.

Claim:

The expected time to hit node 0 is at most $2n^2$.

- To prove the claim one needs some basic facts about Markov chains.

The basics of finite Markov chains

- A finite Markov chain M is a discrete-time random process defined over a set of states S and a matrix $P = \{P_{ij}\}$ of transition probabilities.
- Denote by X_t the state of the Markov chain at time t. It is a memoryless process in that the future behavior of a Markov chain depends only on its current state: $Prob[X_{t+1} = j | X_t = i] = P_{ij}$ and hence $Prob[X_{t+1} = j] = \sum_i Prob[X_{t+1} = j | X_t = i] Prob[X_t = i]$.
- Given an initial state i, denote by r_{ij}^t the probability that the first time the process reaches state j occurs at time t; $r_{ij}^t = Pr[X_t = j \text{ and } X_s \neq j \text{ for } 1 \leq s \leq t-1 | X_0 = i]$
- Let f_{ij} the probability that state j is reachable from initial state i; $f_{ij} = \sum_{t>0} r_{ij}^t$.
- Denote by h_{ij} the expected number of steps to reach state j starting from state i (hitting time); that is, $h_{ij} = \sum_{t>0} t \cdot r_{ij}^t$
- Finally, the *commute time* c_{ij} is the expected number of steps to reach state j starting from state i, and then return to i from j; $c_{ij} = h_{ij} + h_{ji}$

Stationary distributions

- Define $\mathbf{q}^t = (q_1^t, q_2^t, \ldots, q_n^t)$, the state probability vector (the distribution of the chain at time t), as the row vector whose i-th component is the probability that the Markov chain is in state i at time t.
- A distribution π is a stationary distribution for a Markov chain with transition matrix P if $\pi = \pi P$.
- Define the underlying directed graph of a Markov chain as follows: each vertex in the graph corresponds to a state of the Markov chain and there is a directed edge from vertex i to vertex j iff $P_{ij} > 0$. A Markov chain is *irreducible* if its underlying graph consists of a single strongly connected component. We end these preliminary concepts by the following theorem.

Theorem: Existence of a stationary distribution

For any finite, irreducible and aperiodic Markov chain,

(i) There exists a *unique* stationary distribution π.

(ii) For all states i, $h_{ii} < \infty$, and $h_{ii} = 1/\pi_i$.

Back to random walks on graphs

- Let $G = (V, E)$ be a connected, non-bipartite, undirected graph with $|V| = n$ and $|E| = m$. A uniform random walk on G induces a Markov chain M_G as follows: the states of M_G are the vertices of G; and for any $u, v \in V$, $P_{uv} = 1/deg(u)$ if $(u, v) \in E$, and $P_{uv} = 0$ otherwise.
- Denote by $(d_1, d_2, \ldots, d_n)$ the vertex degrees. M_G has a stationary distribution $(d_1/2m, \ldots, d_n/2m)$.
- Let $C_u(G)$ be the expected time to visit every vertex, starting from u and define $C(G) = \max_u C_u(G)$ to be the *cover time* of G.

Theorem: Aleliunas et al [1979]

Let G be a connected undirected graph. Then

1. For each edge (u, v), $C_{u,v} \leq 2m$,
2. $C(G) \leq 2m(n-1)$.

- It follows that the 2-SAT random walk has expected time at most $2n^2$ to find a satisfying assignment in a satisfiable formula. Use Markov inequality to obtain probability of not finding a satisfying assignment.

Extending the random walk idea to k-SAT

- The random walk 2-Sat algorithm might be viewed as a drunken walk (and not an algorithmic paradigm). Or we could view the approach as a local search algorithm that doesn't know when it is making progress on any iteration but does have confidence that such an exploration of the local neighborhood is likely to be successful over time.
- We want to extend the 2-Sat algorithm to k-SAT. However, we know that k-SAT is NP-complete for $k \geq 3$ so our goal now is to improve upon the naive running time of 2^n, for formulas with n variables.
- In 1999, Following some earlier results, Schöning gave a very simple (a good thing) random walk algorithm for k-Sat that provides a substantial improvement in the running time (over the naive 2^n exhaustive search) and this is still almost the fastest (worst case) algorithm known.
- This algorithm was derandomized by Moser and Scheder [2011].
- Beyond the theoretical significance of the result, this is the basis for various Walk-Sat algorithms that are used in practice.

Schöning's k-SAT algorithm

The algorithm is similar to the 2-Sat algorithm with the difference being that one does not allow the random walk to go on too long before trying another random starting assignment. The result is a one-sided error alg running in time $\tilde{O}[(2(1-/1k)]^n$; i.e. $\tilde{O}(\frac{4}{3})^n$ for 3-SAT, etc.

Randomized k-SAT algorithm

```
Choose a random assignment $\tau$
Repeat 3n times    % n = number of variables
**If** $\tau$ satisfies $F$ then stop and accept
**Else** Let $C$ be an arbitrary unsatisfied clause
   Randomly pick and flip one of the literals in $C$
**End If**
```

Claim

If F is satisfiable then the above succeeds with probability p at least $[(1/2)(k/k-1)]^n$. It follows that if we repeat the above process for t trials, then the probability that we fail to find a satisfying assignment is at most $(1-p)^t < e^{-pt}$. Setting $t = c/p$, we obtain error probability $(\frac{1}{e})^c$

A final comment on the complexity status of k SAT

The random walk time bound bound $\tilde{O}[(2(1-/1k)]^n$ can be stated as $\tilde{O}((c_k)^n)$ where $c_k \to 2$ as $k \to \infty$.

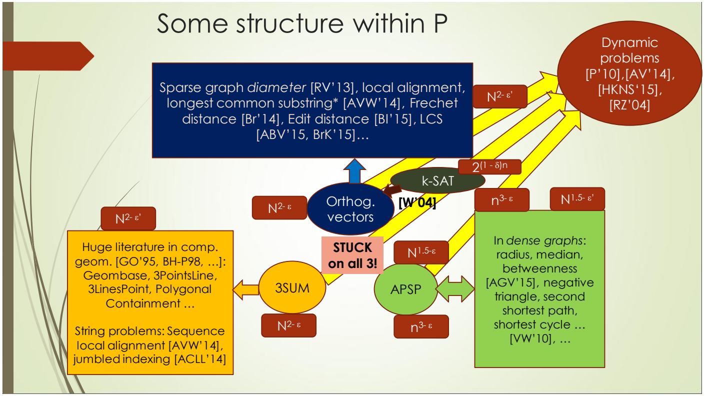
Strong exponential time hypothesis (SETH): There is no deterministic or randomized algorithm for SAT that runs in time c^n for any $c < 2$.

Perhaps surprisingly, the ETH and especially the SETH conjectures imply that for a number of polynomial time computable problems, rather simple algorithms provide approximately the best time bounds. This important observation led to a topic called *fine-grained complexity*.

For example, consider the following orthogonal vectors (OV) problem: Given a set S of n vectors over $\{0,1\}^d$ with $d = \omega(logn)$ (say $d = \lceil(\log_2 n)^2)\rceil$. Determine if S has a pair of orthogonal vectors.

R. Williams [2005] : SETH implies there is no 0-sided randomized algorithm for the OV problem having expected time $n^{2-\epsilon}$ for any $\epsilon > 0$. This in turn (using "fine-grained reductions") implies that the edit distance problem cannot be computed in time $n^{2-\epsilon}$.

The fine-grained landscape



Sparse graph *diameter* [RV'13], local alignment, longest common substring* [AVW'14], Frechet distance [Br'14], Edit distance [BI'15], LCS [ABV'15, BrK'15]…

Huge literature in comp. geom. [GO'95, BH-P98, …]: Geombase, 3PointsLine, 3LinesPoint, Polygonal Containment …

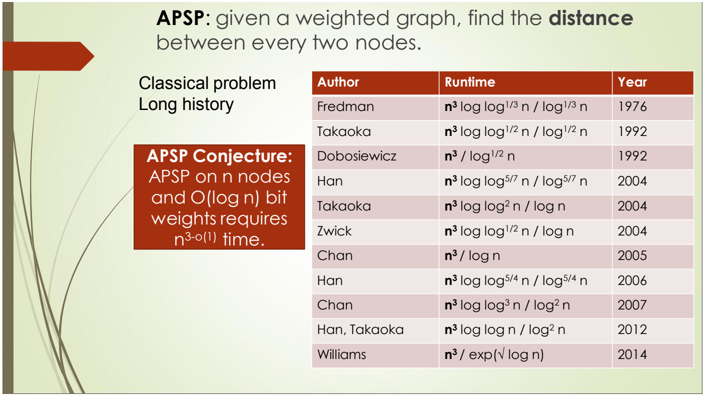
String problems: Sequence local alignment [AVW'14], jumbled indexing [ACLL'14]

In *dense graphs*: radius, median, betweenness [AGV'15], negative triangle, second shortest path, shortest cycle … [VW'10], …

Figure: From V. Williams 2015

The all pairs shortest paths (APSP) problems

We considered the APSP problem early in the term. Even after (or because of) a number of relatively small improvements we still have the APSP conjecture.



APSP: given a weighted graph, find the **distance** between every two nodes.

Classical problem
Long history

APSP Conjecture: APSP on n nodes and O(log n) bit weights requires $n^{3-o(1)}$ time.

Author	Runtime	Year
Fredman	$\mathbf{n^3} \log\log^{1/3} n / \log^{1/3} n$	1976
Takaoka	$\mathbf{n^3} \log\log^{1/2} n / \log^{1/2} n$	1992
Dobosiewicz	$\mathbf{n^3} / \log^{1/2} n$	1992
Han	$\mathbf{n^3} \log\log^{5/7} n / \log^{5/7} n$	2004
Takaoka	$\mathbf{n^3} \log\log^2 n / \log n$	2004
Zwick	$\mathbf{n^3} \log\log^{1/2} n / \log n$	2004
Chan	$\mathbf{n^3} / \log n$	2005
Han	$\mathbf{n^3} \log\log^{5/4} n / \log^{5/4} n$	2006
Chan	$\mathbf{n^3} \log\log^3 n / \log^2 n$	2007
Han, Takaoka	$\mathbf{n^3} \log\log n / \log^2 n$	2012
Williams	$\mathbf{n^3} / \exp(\sqrt{\log n})$	2014

Randomized compositeness testing

One of the most influential randomized algorithms is a polynomial time algorithm for determining if a number is prime or composite.

Quick modern history of primality testing

- Independently, Solovay and Strassen [1977] and Rabin [1980] gave two different randomized polynomial time, 1-sided error algorithms for determining if an n digit number N is composite or prime.
- More precisely, the algorithm always output PRIME if N is prime and outputs COMPOSITE with probability at least $\frac{1}{2}$ if N is composite. That is, the deciding if a number is composite is in the class RP.
- The error probability can be reduced by repeated independent trials; that is t trials would then yield an error (i.e. saying N is prime when it is composite) of at most $\frac{1}{2^t}$ so that "in practice one can be quite confidant" to say that N is prime.

Primality history continued

- The Rabin (1980) algorithm is related to a deterministic polynomial time algorithm by Miller [1976] whose correctness requires the Extended Riemann Hypothesis (ERH), a famous well-believed conjecture in number theory.
- Goldwasser and Kilian [1986] gave a polynomial time 0-zero error algorithm. This then guarantess primality but is only polynomial time in expectation.
- Agrawal, Kayal and Saxena [2002] gave a determinstic polynomial time algorithm.

So why concern ourselves with a randomized algorithm when there is a deterministic algorithm?

- There are polynomials and there are much better polynomials.
- The deterministic (or 0-sided error) algorithms are not nearly as practical as the 1-sided error algorithms.
- These relatively fast algorithms are an essential ingrediant in many cryptographic protocols where large random primes are needed.
- Note that while primality/compositness testing is theoretically and practically solvable, factoring integers is *believed* to be difficult and conjectured to require exponential time (e.g. say $2^{n^{1/3}}$) in some "average sense".
  Note: Integer factoring is not believed to be NP-hard. A decision version of the factoring problem is in NP and $co - NP$.
- Complexity based cryptography often depends on the "average case" difficulty of problems such a integer factoring.

Some basic group theory and number theory

- $Z_N^* = \{a \in Z_n \; gcd(a, N) = 1\}$ is a commutative group under mulitplication (mod N).
- Lagrange Theorem: If H is a subgroup of G then $order(H)$ divides $order(G)$.
- Fermat's Little Theorem: If N is prime then for $a \neq 0(mod \;\, N), a^{N-1} = 1(\; modN)$.
- Furthermore, if N is prime, then Z_N^* is a cyclic group; that is, $\exists g : \{g, g^2, \ldots, g^{N-1}\}(mod \;\, N) = Z_N^*$. This implies that for such a generator g, $g^i \neq 1$ for $1 \leq i \leq N - 2$.
- If N is prime, then $+1$ and -1 are the only square roots of 1.
- The Chinese Remainder Theorem: If N_1 and N_2 are relatively prime, then for all v_1, v_2, there exists a unique non-negative $w < N_1 \cdot N_2$ such that $w = v_1 \; (mod \;\, N_1)$ and $w = v_2 \; (mod \;\, N_2)$.

A simple but not quite correct algorithm

We need two computational facts:

1. $a^i (mod\ \ N)$ can be efficiently computed by repeated squaring mod N.
2. gcd(a,b) can be efficiently computed by the Euclidean algorithm.

A simple randomized algorithm that is "almost correct"

Choose $a\ Z_N \setminus \{0, 1\}$ uniformly at random
If $gcd(a, N) \neq 1$ or $a^{N-1}(mod\ \ N) \neq 1$, then output COMPOSITE
Otherwise output PRIME

This will work if N is not a Carmichael number (also called a false prime); that is, those composite N such that $a^{N-1} = 1(mod\ \ N)$ for all $a \in Z_N^*$. Unfortunately, it was relatively recent (1994) that it was proven that there are infinitely many Carmichael numbers (e.g., 561, 1105, 1129, ...).

The Miller-Rabin 1-sided error algorithm

Miller-Rabin algorithm for testing if N is composite/prime

```
Compute $t, u$ with $t \geq 1$ and $u$ odd such that $N - 1 = (2^t u)$
% Note all computations are $(mod\ N)$
Randomly choose $a \in Z_N \setminus \{0, 1\}$
If $gcd(a, N) \neq 1$ then report COMPOSITE and terminate.
$x_0 = a^u$
For $i = 1, \ldots t$
    $x_i := x_{i=1}^2$
    If $x_i = 1$ and $x_{i-1} \notin \{-1, 1\}$ then report COMPOSITE and terminate.
End For
If $x_t \neq 1$ then report COMPOSITE
Else report PRIME
```

- Claim: Prob[algorithm reports PRIME — N is composite] $\leq \frac{1}{2}$
- Proof relies on the fact that if N is a Carmichael number then $N = N_1 \cdot N_2$ with $gcd(N_1, N_2) = 1$.

The RSA public key encryption

To motivate the importance of efficient primality testing and the believed difficulty (even in a useful sense of “average case”) of factoring large integers, we will quickly discuss RSA public encryption.

What do we want in public encryption?

Suppose ALICE wants a method that various people (BOB and others) can use to send private messages over some public channel.

Let m be a message that BOB wants to send to ALICE. More specifically, we want a public encryption scheme E : $E(m)$ is a computationally easy method that encrypts the message. Alice (and hopefully only ALICE) knows an efficient method D to decode $E(m)$; that is $D(E(m)) = m$.

What follows is just the basics of one public encryption scheme called RSA, named for Rivest, Shamir and Adelman. (This is the Rivest in CLRS.)

RSA continued

We can think of a message m as a large integer. How?

RSA continued

We can think of a message m as a large integer. How?

The additional number theory result we need is Euler's totient theorem which generalizes Fermat's Little Theorem.

Given an integer $N \geq 3$, the totient function $\phi(N)$ is the number of integers in $[1, N]$ relatively prime to N.
In particular, if $N = p \cdot q$ where p and q are primes then $\phi(N) = (p-1)(q-1)$. And if N is prime then $\phi(N) = N - 1$.

Euler's totient theorem: Suppose $gcd(a, N) = 1$, then $a^{\phi(N)} = 1 \ (mod \ N)$.

Let $N = p \cdot q$ for large primes p and q. Let e be relatively prime to $\phi(N)$; that is, $gcd(e, \phi(N) = 1$ and $\exists d, t$ such that $d \cdot e + t \cdot \phi(N) = 1$.

By Euler's theorem, $m^{t\phi(N)} = 1 mod \ N$ and hence $m^{de+t\phi(N)} = m^{de} mod \ N$.

RSA continued

Knowing p, q and hence knowing $\phi(N)$, we can compute d by the extended Euclidean algorithm.

The RSA encryption function is $E(m) = m^e \ (mod \ N)$ where e and N are publicly known so that Bob (and anyone) can send a message to ALICE.

Since ALICE knows d, she can compute the decoding function $D(z) = z^d \ mod \ N$.

$$D(E(m))^d = m^{de} = m^{de+t\phi(N)} = m \ (modN)$$

.

That is, since ALICE knows d because she knows p and q, ALICE can decode the message. The belief is that d cannot be found efficiently without being able to factor N.

Note that the encoding function E and decoding function D consists of exponentiation mod N and that can be done efficiently by “repeated squaring mod N”.

Concluding remarks about public encryption

In RSA (and in any "complexity based" public encryption scheme), we have a public key (e.g. e and N in RSA) and a private key (e.g., d in RSA).

Of coursse, if an intruder $ALLAN$ knows that BOB is only sending one of a few messages, ALLAN can try out each possible message m and find a match $E(m)$.

To avoid this issue, BOB can randomly add a lot of noise g and send the encoding of the message $m \circ g$ where $\circ$ is concatenation.

But what if ALLAN now wants to pretend to be BOB and send a misleading message to ALICE. Since the encoding scheme is public, ALLAN can pretend to be BOB.

We can avoid this issue by BOB first creating his own public e' and private key d'. ALICE now also knows e'. Bob encrypts m by sending $E(m \circ D'(m))$ or even $E(m \circ D'(BOB))$ so that now his message is signed.

Fixed parameter tractable (FPT) problems

Consider the following two related NP-complete problems.

1. Independent set: Given a graph $G = (V, E)$ and positive integer k, does G have an independent set $V' \subseteq V$ of size (at least) k?
2. Vertex cover: Given a graph $G = (V, E)$ and positive integer k, does G have a vertex cover $V' \subseteq V$ of size (at most) k

Clearly, by exhaustive search one can determine (and find) if an appropriate subset V' exists in say time $O(mn^k)$ where $m = |E|, n = |V|$. .

Is this "type" of exponential complexity (i.e., n^k) necessary?
The answer is

Fixed parameter tractable (FPT) problems

Consider the following two related *NP*-complete problems.

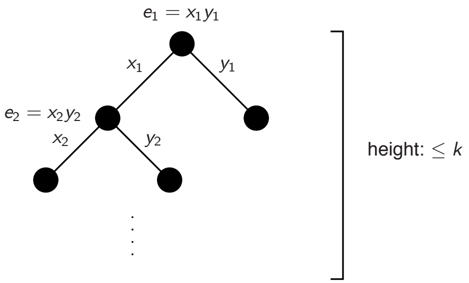
1. Independent set: Given a graph $G = (V, E)$ and positive integer k, does G have an independent set $V' \subseteq V$ of size (at least) k?
2. Vertex cover: Given a graph $G = (V, E)$ and positive integer k, does G have a vertex cover $V' \subseteq V$ of size (at most) k

Clearly, by exhaustive search one can determine (and find) if an appropriate subset V' exists in say time $O(mn^k)$ where $m = |E|, n = |V|$. .

Is this "type" of exponential complexity (i.e., n^k) necessary?
The answer is

1. Yes (i.e. no $n^{o(k)}$ algorithm is *believed* to exist for independent set) and
2. NO (i.e. there is an $O(2^k + n^2)$ time algorithm for vertex cover). The current best result is $O((1.2832)^k k + k|V|)$.

An FPT algorithm for vertex cover



Height of the search tree is $\leq k \Rightarrow$ number of leaves is $\leq 2^k \Rightarrow$ complete search requires $2^k \cdot$ poly steps.

Figure: Figure from Daniel Marx slides

Fixed parameter tractable problems (FPT) continued

For each input we would like to define an appropriate integer parameter k. In the vertex cover problem we can define k to be the size of an optimal vertex cover.

For some problems, the parameter may be a property of the input; for example, the diameter, max degree or average degree of a graph.

Definition: A parameterized problem is FPT if for all n (the length of the input) and k (the parameter), there is a $f(k)n^c$ time algorithm for some constant c.
That is, for all fixed k, there is a polynomial time algorithm.

The study of which problems are FPT and which are believed (according to some hypothesis) to *not* be FPT is an ongoing research topic of wide interest and importance. This is one approach to bridging the gap between theory and practice.

How does one show that a problem is or is not FPT

To have a theory analogous to the theory of *NP* completeness, we need to have an appropriate reduction $\leq_{FPT}$ and a class $\mathcal{C}$ of conjectured hard problems (i.e., problems that are not FPT) such that:

- If $L_1 \leq_{FPT} L_2$ and L_2 is FPT, then L_1 is FPT. We can then show that a given problem is FPT directly (as in the vextex cover algorithm) or using the reduction to a known FPT problem.
- To show a problem L is not FPT, we need to shpw that $L' \leq_{FPT} L$ for some problem L' that is known to be "hard" for all problems in the conjectured hard class $\mathcal{C}$; that is, if $P \in \mathcal{C}$, then $P \leq_{FPT} L'$. That is, L' is complete for the class $\mathcal{C}$.
- And, as in *NP* completeness, we need to show that there is a good hardness candidate (analogous to SAT). one such problem is the Independent Set Problem.

We will need a reduction more refined than just polynomial time. (We also need such reduction when dealing with approximation algorithms.)

Some NP hard problems that are FPT

- Vertex cover of size k
- Simple path of length exactly k
- Does there exist k disjoint triangles
- Drawing a graph in the plane with k edge crossings
- And many more

Some problems believed to not be FPT

- Clique
- Independent set
- Hitting Set
- Set Cover
- And literally hundreds of other problems

Sublinear time algorithms

We now consider a context in which randomization is usually provably more essential. In particular, we will study sublinear time algorithms. Why intuitively is randomization more essential?

Sublinear time algorithms

We now consider a context in which randomization is usually provably more essential. In particular, we will study sublinear time algorithms. Why intuitively is randomization more essential?

- An algorithm is sublinear time if its running time is $o(n)$, where n is the "length" of the input. As such an algorithm must provide an answer without reading the entire input.
- Thus to achieve non-trivial tasks, we almost always have to use randomness in sublinear time algorithms to sample parts of the inputs and/or to randomly keep "snapshots" of what we have seen.
- The subject of sublinear time algorithms is an extensive topic and we will only present two examples.
- The general flavour of sublinear time results will be a tradeoff between the accuracy of the solution and the time bound. There is some relation between this topic and distributed local algorithms.
- This topic will take us beyond search and optimization problems.

A deterministic exception: estimating the diameter in a finite metric space

- We first conisder an exception of a "sublinear time" algorithm that does not use randomization. (Comment: "sublinear in a weak sense".)
- Suppose we are given a finite metric space M (with say n points x_i) where the input is given as n^2 distance values $d(x_i, x_j)$. The problem is to compute the diameter D of the metric space, that is, the maximum distance between any two points.
- For this maximum diameter problem, there is a simple $O(n)$ time (and hence sublinear in n^2, the number of distances) algorithm; namely, choose an arbitrary point $x \in M$ and compute $D = \max_j d(x, x_j)$. By the triangle inequality, D is a 2-approximation of the diameter.
- I say sublinear time in a weak sense because in an implicitly represented distance function (such as d dimensional Euclidean space), the points could be explicitly given as inputs and then the input size is n and not n^2.

Sampling the inputs: some examples

- The goal in this area is to minimize execution time while still being able to produce a reasonable answer with sufficiently high probability.
- Recall that by independent trials, we can reduce the probability of error.
- We will consider the two following examples:
  1. Finding an element in an sorted (doubly) linked list [Chazelle,Liu,Magen]
  2. Estimating the average degree in a graph [Feige 2006]

  Note: In many cases, sublinear time algorithms will be “simple” or “reasonably natural” but the analysis might be quite non-trivial.

Finding an element in a sorted list of distinct elements.

- Suppose we have an array $A[i]$ for $1 \leq i \leq n$ where each $A[i]$ is a triple (x_i, p_i, s_i) where the $\{p_i, s_i\}$ constitute a doubly linked list.
- That is, $p_i = j : argmax\{j | x_j < x_i\}$ if such an x_j exists and similarly $q_i = argmin\jmath | x_j > x_i\}$.
- We would like to determine if a given value x occurs in a doubly linked list and if so, output the index j such that $x = x_j$.

A $\sqrt{n}$ algorithm for searching in a sorted linked list

Let $R = \{j_i | 1 \leq i \leq \sqrt{n}\}$ be $\sqrt{n}$ randomly chosen indices.
Access these $\{A[j_i]\}$ to determine the predecessor and successor of x amongst these randomly chosen elements of the list. (There may not be both a predecssor and successor.) Then (alternately) do a brute force linked search (or resp. search for $\sqrt{n}$ steps) in both directions of the linked list to determine whether or not x_k exists.

Finding an element in a sorted list (continued)

Claim:

This is a zero sided (resp, one-sided error algorithm) that runs in expected time $O(\sqrt{n})$ (resp. has constant probability of not find x if it exists).

- Using the Yao principle this expected time can be shown to be optimal.
- The same can be done for a singly linked list if the list is "anchored" ; i.e., we have the index of the smallest element in the list.
- Similar results were shown by Chazelle, Liu and Magen for various geometric problems such as determining whether or not two convex polygons (represented by doubly linked lists of the vertices) intersect.
- Note that most sublinear time algorithms are either randomized 1-sided or 2-sided error algorithms and not 0-sided error algorithms that always compute a correct answer but whose running time is bounded in expectation.

Estimating average degree in a graph

- Given a graph $G = (V, E)$ with $|V| = n$, we want to estimate the average degree d of the vertices. We can assume that G is connected and hence there are at least $n - 1$ edges.
- We want to construct an algorithm that approximates the average degree within a factor less than $(2 + \epsilon)$ with probability at least $3/4$ in time $O(\frac{\sqrt{n}}{poly(\epsilon)})$. We will assume that we can access the degree d_i of any vertex v_i in one step.
- Again, we note that like a number of results in this area, the algorithm is simple but the analysis requires some care.

The (simplified) Feige algorithm in Czumaj and Sohler survey.

Sample $8/\epsilon$ random subsets S_i of V each of size (say) $\frac{\sqrt{n}}{\epsilon^3}$
Compute the average degree a_i of nodes in each S_i.
The output is the minimum of these $\{a_i\}$.

The analysis of the approximation

Since we are sampling subsets to estimate the average degree, we might have estimates that are too low or too high. But it can be shown that with high probability these estimates will not be too bad. More precisely, we need:

1. Lemma 1: $Prob[a_i < \frac{1}{2}(1-\epsilon)\bar{d}] \leq \frac{\epsilon}{64}$
2. Lemma 2: $Prob[a_i > (1+\epsilon)\bar{d}] \leq 1 - \frac{\epsilon}{2}$

The probability bound in Lemma 2 follows from the Markov inequality which is then amplified as usual by the repeated $8/\epsilon$ trials so that the probability that all of the a_i are bigger than $(1+\epsilon)\bar{d}$ is at most $(1-\epsilon/2)^{8/\epsilon} = (1-1/t)^{4t} \leq (1/e)^{4t}$ letting $t = 2/\epsilon$.

The analysis of the average degree (continued)

From Lemma 1, we fall outside the desired bound if any of the repeated trials gives a very small estimate of the average degree but by the union bound this is no worse than the sum of the probabilities for each trial.

It remains to sketch a proof of Lemma 1. Let H be the set of the $\sqrt{\epsilon n}$ highest degree vertices in V and $L = V \setminus H$. Then

- $\sum_{v \in L} d_v \geq (\frac{1}{2} - \epsilon) \sum_{v \in V} d_v$ since there can be at most $\epsilon \cdot n$ edges within H and every edge adjacent to L contribues at least 1 to the sum of degrees of vertices adjacent to L and thereare at least $n-1$ edges. The $\frac{1}{2}$ is becasue we are possibly double counting the contribution of edges within L.
- For a lower bound on the average degree of vertices in a sample set S, the worst case is if all the sampled vertices are in L.

Conclusion of proof sketch for Lemma 1

- Let X_j be the random variable corresonding to the ith sampled vertex in a sampled set S where each such S has size s. By Hoeffding's generalization of the Chernoff bound, we have

$$Prob[(1/s)(\sum_{j}^{s} X_j \leq (1-\epsilon)(1/|L|)[\sum_{v \in L} deg(v)]$$

  is exponentially small; that is, the probability that the average degree in a sampled set is $(1-\epsilon)$ less than the average degree in L is exponentially small.
- But the average degree of a vertex in L is at least $(1/2-\epsilon)$ times the average degree in the graph so that being less than $(1/2-\epsilon)$ the average degree is exponentially small.
- Feige's more detailed analysis shows that a $(2+\epsilon)$ approximation can be obtained using time (i.e., queries) $O(\sqrt{n/d_0}/\epsilon)$ for graphs with average degree at least d_0.

The streaming model

- In the data stream model, the input is a sequence A of inputs $a_1, \ldots, a_m$ where say each $a_i \in \{1, 2, \ldots, n\}$; the stream is assumed to be too large to store in memory.
- We usually assume that m is not known and hence one can think of this model as a type of online or dynamic algorithm that is maintaining (say) current statistics.
- The space available $S(m, n)$ is some sublinear function. The input streams by and one can only store information in space S.
- In some papers, space is measured in bits (which is what we will usually do) and sometimes in words, each word being $O(\log n)$ bits.
- It is also desirable that that each input is processed efficiently, say $\log(m + n)$ and perhaps even in time O(1) (assuming we are counting operations on words as $O(1)$).

The streaming model continued

- The initial (and primary) work in streaming algorithms is to approximately compute some function (say a statistic) of the data or identify some particular element(s) of the data stream.
- Lately, the model has been extended to consider "semi-streaming" algorithms for optimization problems. For example, for a graph problem such as matching for a graph $G = (V, E)$, the goal is to obtain a good approximation using space $\tilde{O}(|V|)$ rather than $O(|E|)$.
- Most results concern the space required for a one pass algorithm. But there are other results concerning the tradeoff between the space and number of passes.

An example of a deterministic streaming algorithms

As in sublinear time, it will turn out that almost all of the results in this area are for randomized algorithms. Here is one exception.

The missing element problem

Suppose we are given a stream $A = a_1, \ldots, a_{n-1}$ and we are promised that the stream A is a permutation of $\{1, \ldots, n\} - \{x\}$ for some integer x in $[1, n]$. The goal is to compute the missing x.

- Space n is obvious using a bit vector $c_j = 1$ iff j has occured.
- Instead we know that $\sum_{j \in A} = n(n+1)/2 - x$.
  So if $s = \sum_{i \in A} a_i$, then $x = n(n+1)/2 - s$.
  This uses only $2 \log n$ space and constant time/item.

Generalizing to k missing elements

Now suppose we are promised a stream A of length $n-k$ whose elements consist of a permutation of $n-k$ distinct elements in $\{1,\ldots,n\}$. We want to find the missing k elements.

- Generalizing the one missing element solution, to the case that there are k missing elements we can (for example) maintain the sum of j^{th} powers $(1 \leq j \leq k)$ $s_j = \sum_{i \in A}(a_i)^j = c_j(n) - \sum_{i \notin A} x_i^j$. Here $c_j(n)$ is the closed form expression for $\sum_{i=1}^{n} i^j$. This results in k equations in k unknowns using space $k^2 \log n$ but without an efficient way to compute the solution.
- As far as I know there may not be an efficient small space streaming algorithm for this problem.
- Using randomization, much more efficient methods are known; namely, there is a streaming alg with space and time/item $O(k \log k \log n)$; it can be shown that $\Omega(k \log(n/k))$ space is necessary.

Another simple example

Consider a problem similar to question 1 in the first assignment. Namely, if there is an element appearing more than $n/2$ times in a stream of n elements, output this element.

Majority streaming algorithm

he following algorithm will always output some b and if there is a majority element, it will be b. To verify that b is actually a majorty element, a second pass is needed.

```
$Stack := \emptyset$
For $i..n$
  If Stack empty then push $a_i$ onto the Stack.
  Else if $a_i$ is the element on top of the stack then push $a_i$ onto the Stack
  Else pop the stack.
End For
```

Some well-studied streaming problems

- Computing frequency moments. Let $A = a_1 \ldots a_m$ be a data stream with $a_i \in [n] = \{1, 2, \ldots n\}$. Let m_i denote the number of occurences of the value i in the stream A. For $k \geq 0$, the k^{th} frequency moment is $F_k = \sum_{i \in [n]} (m_i)^k$. The frequency moments are most often studied for integral k.
    1. $F_1 = m$, the length of the sequence which can be simply computed.
    2. F_0 is the number of distinct elements in the stream
    3. F_2 is a special case of interest called the repeat index (also known as Gini's homogeneity index).
- Finding k-heavy hitters; i.e. those elements appearing at least n/k times in stream A.
- Finding rare or unique elements in A.

What is known about computing F_k?

Given an error bound ϵ and confidence bound δ, the goal in the frequency moment problem is to compute an estimate F'_k such that $Prob[|F_k - F'_k| > \epsilon F_k] \leq \delta$.

- The seminal paper in this regard is by Alon, Matias and Szegedy (AMS) [1999]. AMS establish a number of results:
    1. For $k \geq 3$, there is an $\tilde{O}(m^{1-1/k})$ space algorithm. The $\tilde{O}$ notation hides factors that are polynomial in $\frac{1}{\epsilon}$ and polylogarithmic in $m, n, \frac{1}{\delta}$.
    2. For $k = 0$ and every $c > 2$, there is an $O(\log n)$ space algorithm computing F'_0 such that $Prob[(1/c)F_0 \leq F'_0 \leq cF_0$ does *not* hold$] \leq 2/c$.
    3. For $k = 1$, $\log n$ is obvious to exactly compute the length but an estimate can be obtained with space $O(\log\log n + 1/\epsilon)$
    4. For $k = 2$, they obtain space $\tilde{O}(1) = O(\frac{\log(1/\delta)}{\epsilon^2})(\log n + \log m))$
    5. They also show that for all $k > 5$, there is a (space) lower bound of $\Omega(m^{1-5/k})$.

Results following AMS

- A considerable line of research followed this seminal paper. Notably settling conjectures in AMS:
- The following results apply to real as well as integral k.
  1. An $\tilde{\Omega}(m^{1-2/k})$ space lower bound for all $k > 2$ (Bar Yossef et al [2002]).
  2. Indyk and Woodruff [2005] settle the space bound for $k > 2$ with a matching upper bound of $\tilde{O}(m^{1-2/k})$
- The basic idea behind these randomized approximation algorithms is to define a random variable Y whose expected value is close to F_k and variance is sufficiently small such that this r.v. can be calculated under the space constraint.
- We will just sketch the (non optimal) AMS results for F_k for $k > 2$ and the result for F_2.

The AMS F_k algorithm

Let $s_1 = (\frac{8}{\epsilon^2} m^{1-\frac{1}{k}})/\delta^2$ and $s_2 = 2\log\frac{1}{\delta}$.

AMS algorithm for F_k

The output Y of the algorithm is the median of s_2 random variables $Y_1, Y_2, ..., Y_{s_2}$ where Y_i is the mean of s_1 random variables $X_{ij}, 1 \le j \le s_1$. All X_{ij} are independent identically distributed random variables. Each $X = X_{ij}$ is calculated in the same way as follows: Choose random $p \in [1, \dots, m]$, and then see the value of a_p. Maintain $r = |\{q | q \ge p \text{ and } a_q = a_p\}|$. Define $X = m(r^k - (r-1)^k)$.

- Note that in order to calculate X , we only require storing a_p (i.e. $\log n$ bits) and r (i.e. at most $\log m$ bits). Hence the Each $X = X_{ij}$ is calculated in the same way using only $O(\log n + \log n)$ bits.
- For simplicity we assume the input stream length m is known but it can be estimated and updated as the stream unfolds.
- We need to show that $\mathbf{E}[X] = F_k$ and that the variance $Var[X]$ is small enough so as to use the Chebyshev inequality to show that $Prob[|Y_i - F_k| > \epsilon F_k$ is small.

AMS analysis sketch

- Showing $E[X] = F_k$.

$$\frac{m}{m}[(1^k + (2^k - 1^k) + \ldots + (m_1^k - (m_1 - 1)^k))+$$

$$(1^k + (2^k - 1^k) + \ldots + (m_2^k - (m_2 - 1)^k)) + \ldots ...+$$

$$(1^k + (2^k - 1^k) + \ldots + (m_n^k - (m_n - 1)^k))]$$

(by telescoping)

$$= \sum_{i}^{n} m_i^k$$

$$= F_k$$

AMS analysis continued

- Y is the median of the Y_i. It is a standard probabilistic idea that the median Y of identical r.v.s Y_i (each having constant probability of small deviation from their mean F_k) implies that Y has a high probability of having a small deviation from this mean.
- $E[Y_i] = E[X]$ and $Var[Y_i] \leq Var[X]/s_1 \leq E[X^2]/s_1$.
- The result needed is that $Prob[|Y_i - F_k| > \epsilon F_k] \leq \frac{1}{8}$
- The Y_i values are an average of independent $X = X_{ij}$ variables but they can take on large vales so that instead of Chernoff bounds, AMS use the Chebyshev inequality:

$$Prob[|Y - E[Y]| > \epsilon E[Y]] \leq \frac{Var[Y]}{\epsilon^2 E[Y]}$$

- It remains to show that $E[X^2] \leq kF_1 F_{2k-1}$ and that $F_1 F_{2k-1} \leq n^{1-1/k} F_k^2$

Sketch of F_2 improvement

- They again take the median of $s_2 = 2\log(\frac{1}{\delta})$ random variables Y_i but now each Y_i will be the sum of only a constant number $s_1 = \frac{16}{\epsilon^2}$ of identically distibuted $X = X_{ij}$.
- The key additional idea is that X will not maintain a count for each particular value separately but rather will count an appropriate sum $Z = \sum_{t=1}^{n} b_t m_t$ and set $X = Z^2$.
- Here is how the vector $< b_1, \ldots, b_n > \in \{-1, 1\}^n$ is randomly chosen.
- Let $V = \{v_1, \ldots, v_h\}$ be a set of $O(n^2)$ vectors over $\{-1, 1\}$ where each vector $v_p =< v_{p,1}, \ldots, v_{p,n} > \in V$ is a 4-wise independent vector of length n.
- Then p is selected uniformly in $\{1, \ldots, h\}$ and $< b_1, \ldots, b_n >$ is set to v_p.