Cours ENSL: 
Big Data – Streaming, Sketching, Compression

Olivier Beaumont, Inria Bordeaux Sud-Ouest
Olivier.Beaumont@inria.fr
Introduction
• w.r.t. traditional courses on algorithms
  • Exact algorithms for polynomial problems
  • Approximation algorithms for NP-Complete problems
  • Potentially exponential algorithms for difficult problems (going through an ILP for example)

• Here, we will consider extreme contexts
  • not enough space to transmit input data (sketching) or
  • not enough space to store the data stream (streaming)
  • not enough time to use an algorithm other than a linear complexity one

• Compared to the more "classical" context of algorithms:
  • we aim at solving simple problems and
  • we are looking for approximate solutions only because we have very strong time or space constraints.

• Disclaimer: it is not my research topic, but I like to look at the sketching/streaming papers and I am happy to teach it to you!
Application Context 1: Internet of Things (IoT)

- Connected objects, which take measurements
- The goal is to aggregate data.
- Processing can be done either locally, or on their way (fog computing), or in a data center (cloud computing).
- We must be very energy efficient
  - because objects are often embedded without power supply.
- Energy cost: Communication is the main source of energy consumption, followed by memory movements (from storage), followed by computations (which are inexpensive)
- A good solution is to do as many local computations as possible!
  - but it is known to be difficult (distributed algorithms)
  - especially when the complexity is not linear (e.g. think about quadratic complexity)
- Solution:
  - compress information locally (and on the fly)
  - only send the summaries; summaries must contain enough information!
Application Context 2: Datacenters

- Aggregate construction
- except the network (we can have several levels + infiniband), everything is "linear"
- the distance between certain nodes/data is very large but a strong proximity with certain data stored on disk
- with 1,000 nodes with 1TB of disk and a link at 400 MB/s, we have 1 PB and 400 GB/s (higher than with a HPC system)
- provided the data is loaded locally!
- for 25 TF/s \(10^3 25\text{GFs seti@home}\) in total, ratio 60 (HPC system 40 000)
- in practice, dedicated to linear algorithms and very inefficient for other classes.
- In both contexts, there is a strong need to have data driven algorithms (where placement is imposed by data) whose complexity is linear
Outline of the lectures

• Keywords:
  • Compression, Hashing, Randomized Approximation Algorithms

1. Lecture 1: Two basic theoretical problems
  • Lecture 2: with known lower and upper + randomized and deterministic bounds

2. Lecture 3: Big Data example: Plagiarism detection
  • randomized algorithm + Locality Sensitive Hashing

3. Lecture 4: Randomized Linear Algebra
  • compression beyond Singular Value Decompositions for very large matrices

• Shared Problems
  • Not enough space to store input data
  • Not enough space/time to implement something else than low (linear) complexity algorithms
  • Need for very cheap (online) but dedicated compression algorithm
Sketching – Streaming
• large volume of data generated in a distributed way
  • to be processed locally and compressed before transmission.

• Types of compression?
  • lossless compression
  • compression with losses
  • compression with losses, but tightly controlled loss for a specific function (sketching)

• + we are going to do online (on the fly) compression (streaming)
On-the-fly compression dedicated to a function $f$

- Let $X$ be a stream of numbers (temperatures from a sensor)
- Easy problems?
  - examples: $\min$, $\max$, $\sum$, mean value median?
  - Constraint: compress data and linearize computations
- How?
  - The solution is often to switch to randomized approximation algorithms.
Compression associated to a specific function $f$

- More formally, given $f$ and a stream $X$,
- we want to compress the data $X$ but still be able to compute $\simeq f(X)$.
- Sketching: we are looking for $C_f$ and $g$ such that
  - the storage space $C_f(X)$ is small (compression)
  - from $f(X)$, we can recover $f(X)$, ie $g(C_f(X)) \simeq f(X)$
- Streaming: additional difficulty, the update is performed on the fly.
  - we cannot compute $C_f(\{X, y\})$ from $\{X, y\}$
  - because we cannot store $\{X, y\}$
  - so we need another function $h$ such that $h(C_f(X), \{y\}) = C_f(\{X, y\})$
- and one last difficulty:
  - very often, it is impossible to do in deterministic and exact / deterministic and approximate
  - but only with a randomized and approximation algorithm.
- How to write this?
  - We are looking for an estimator $Z$ such that for given $\alpha$ and $\epsilon$
  - $Pr(|Z - f(X)| \geq \epsilon f(X)) \leq \alpha$. How to read this?
    - the probability of making a mistake by a ratio greater than $\epsilon$ (as small as you want)
    - is smaller than $\alpha$ (as small as you want)
Count the number of visits
Example: count the number of visits / packets

- Context
  - a sensor/router sees packets / visits passing through, ....
  - you just want to maintain elementary statistics (number of visits, number of visits over the last 1 hour, standard deviations)
  - Here, we simply want to count the number of visits

- What storage is necessary if we have $n$ visits? $\log n$ bits. Why?
  Pigeonhole principle. If we have strictly less than $\log n$ bits, then we have two events (among the $n$) that will be coded in the same way.

- What happens if we only allow an approximate answer (say, to a factor of $\rho < 2$)? you need at least $\log \log n$ bits. Why? sketch of the proof: if we use $t < \log \log n$ bits, then we will be able to distinguish less than $\log n$ different groups and you can estimate how many groups are needed to count $\{0\}$, $\{0, 1\}$, $\{0, 1, 2\}$, $\{0, 1, ..., 7\}$.

- We will look for a randomized and approximated solution
  - Let us set $\alpha$ and $\epsilon$
  - we are looking for an algorithm that computes $\tilde{n}$, an approximation of $n$
  - that only uses $K \log \log n$ bits storage
  - and such that $Pr(|\tilde{n} - n| \geq \epsilon n) \leq \alpha$
  - $K$ must be a constant...not necessarily a small constant for now!
Crash Course in probabilities

- $Z$ random variable with positive values
- $E(Z)$ is the expectation of $Z$
- definitions and properties?
  - $E(Z) = \int \lambda P(Z = \lambda) d\lambda$ or $E(Z) = \sum_j jP(Z = j)$
  - $E(Z) = \int P(Z \geq \lambda) d\lambda$ or $E(Z) = \sum_j P(Z \geq j)$
  - $E(aX + bY) = aE(X) + bE(Y)$
  - total probabilities (with conditioning) $E(Z) = \sum_j E(Z|Y = j)P(Y = j)$
- To measure the distance from $Z$ to $E(Z)$, we use the variance $V(Z)$
  - Definition?
  - $V(Z) = E((Z - E(Z))^2) = E(Z^2) - E(Z)^2$
  - Properties:
    - $V(aZ) = a^2V(Z)$
    - In general, $V(X + Y) \neq V(X) + V(Y)$ (but it is true if $X$ and $Y$ are independent random variables)
- How to measure the difference between $Z$ to $E(Z)$?
  1. Markov: $Pr(Z \geq \lambda) \leq E(Z)/\lambda$
  2. Chebyshev: $Pr(|Z - E(Z)| \geq \lambda E(Z)) \leq \frac{V(Z)}{\lambda^2 E(Z)^2}$
  3. Chernoff: If $Z_1, \ldots, Z_n$ are Independent Bernouilli rv with $p_i \in [0,1]$ and $Z = \sum Z_i$, then
     $Pr(|Z - E(Z)| \geq \lambda E(Z)) \leq 2 \exp\left(\frac{-\lambda^2 E(Z)}{3}\right)$. 
Morris Algorithm: Counting the number of events

- Step 1: Find an estimator $Z$
  - $Z$ must be small (of order of $\log \log n$)
  - we need to define an additional function $g$
  - such that $E(g(Z)) = n$

- Morris algorithm
  - $Z \to 0$
  - At each event, $Z \to Z + 1$ with probability $1/2^Z$
  - When queried, return $g(Z) = 2^Z - 1$

- What is the space complexity to implement Morris’ algorithm?
- What is the time complexity in the worst case? What is the expected complexity of a step?

- Prove the correctness: $E(2^{Z_n} - 1) = n$ (note $Z_n$ the random variable that denotes $Z$ after $n$ events) Hint: by induction, assuming that $E(2^{Z_n}) = n + 1$ and showing that $E(2^{Z_{n+1}}) = n + 2$

- How to find a probabilistic guarantee of the type $Pr(|f(Z_n) = \tilde{n} - n| \geq \epsilon n) \leq \alpha$? Hint Prove $E(2^{Z_n}) = 3/2n^2 + 3/2n + 1$.

- Conclusion? Is this unexpected?
From Morris to Morris+ and Morris+++ 

• 2nd step: How to get a useful bound?
• Objective: to reduce the variance (the expectation is already what we want). How to do it?
  • Classic idea: do the same experience many times and average them
• Morris algorithm +
  • Morris is used to compute independent \( Z_n^{(1)}, Z_n^{(2)}, \ldots, Z_n^{(K)} \)
  • On demand, compute
    \[
    Y_n = \frac{\sum_{i=1}^{K} (2Z_n^{(i)})^K - 1}{K}.
    \]
• Questions:
  • Which space complexity to implement Morris+'s algorithm?
  • What time complexity?
  • Establish the correctness: \( E(2Y_n - 1) = n \)
  • What is the new guarantee obtained with Chebyshev? How many counters should be maintained?
• How can we do even better?
  • Morris+++ = Morris+(1/3) and median
  • proof with Chernoff: If \( Z_1, \ldots, Z_n \) are Independent Bernouilli rv with \( p_i \in [0,1] \) and \( Z = \sum Z_i \), then
    \[
    Pr(|Z - E(Z)| \geq \lambda E(Z)) \leq 2 \exp\left(-\frac{\lambda^2 E(Z)}{3}\right).
    \]
How to count the number of unique visitors
2nd example: how to count the number of unique visitors

Context

- It is assumed that visitors are identified by their address \((i_k \in [1, n])\)
- We observe a flow of \(m\) visits \(i_1, \ldots, i_m\) with \(i_k \in [1, n]\)
- How many different visitors?
- Deterministic and trivial algorithms:
  - if \(n\) is small, if \(n\) is big... and in front of what?
  - solution in \(n: n\) bit array
  - solution in \(m \log n\): we keep the whole stream!
- We will see a bit later
  - that we cannot do better with exact and deterministic algorithms
  - that we cannot do better with approximated and deterministic algorithms
- How to do if you cannot store \(n\) bits
  - but only \(O(\log^k n)\) for a certain \(k\)?
- we will see that it is again possible by using both randomization and approximation.
- and that no deterministic exact or deterministic approximation can do it with this space constraint.
We will start with an idealized algorithm (which cannot be implemented in practice).

- Let us choose a random $h$ function from $[1, n]$ to $[0, 1]$
- Why idealized?
  - Problem 1: to store such a random function, you must define the images for in each of the $n$ points... at least $\Omega(n)$ bits
  - Problem 2: and in addition we would have to store real values!
  - We will come back to these two problems in a moment.....
  - Let us assume for now that storing such a function costs $\Theta(1)$
- How do you keep track of the number of unique visitors?
- We will keep $Z \rightarrow \min_{i \in \text{stream}} h(i)$. Intuition?
  - If you see the same visitor $k$ times, it won’t change $Z$
  - If we see $t$ different visitors, then the values taken by $h$ split $[0, 1]$ in $t + 1$ intervals...and all should have the same size in expectation... and this size is $\frac{1}{t+1}$ including the first !
- so you should return $\frac{1}{Z} - 1$ !
Proof of correctness

- Let’s prove that \( E(Z) = \frac{1}{t+1} \).
- \( E(Z) = \int_0^{+\infty} P(Z \geq \lambda)d\lambda \).
- Homework (optional):
  - Prove that \( E(Z^2) = \frac{2}{(t+1)(t+2)} \)
  - What is the guarantee obtained when applying Chebychev?
  - Let us maintain \( q = \frac{1}{\epsilon^2 \alpha} \) FM instances
  - How to define the new estimator? (be careful not saying that \( E(1/Z) \neq 1/E(Z) \), but rather end the calculations by hand when you have an estimator very close to \( \frac{1}{t+1} \) !)
  - In order to lower the number of required copies of FM, use the 2 step approach: first obtain a guarantee with failure probability \( \frac{1}{3} \) and then use the median of several such experiments to lower the number of copies.
  - You should be able to replace \( \frac{1}{\alpha} \) by \( \log \frac{1}{\alpha} \).