The Space Complexity of Pass-Efficient Algorithms for Clustering

Kevin L. Chang\textsuperscript{1} Ravi Kannan\textsuperscript{1}

\textsuperscript{1}Department of Computer Science
Yale University

SODA 2006
Streaming Algorithms

TCS model for abstracting computation on massive data sets.

▶ Input is placed in a read-only array.
▶ Elements in the array can only be accessed by a single pass through the entire array.
▶ Algorithm allowed a sublinear amount of working memory in order to perform intermediate calculations, store sketches, etc.
▶ Optimize working memory.

First few applications: median finding [MP80], statistics [AMS96]
Pass-Efficient Algorithms

- Most studies involve a single pass over the data. This artificially limits the power of these algorithms.
- Pass-Efficient Model [DK03] is a more flexible massive data set paradigm.
- Algorithm may make a small, constant number of passes over the data.
- Memory usage should be a constant, independent of $n$. 
Pass-Efficient Algorithms

- Most studies involve a single pass over the data. This artificially limits the power of these algorithms.
- Pass-Efficient Model [DK03] is a more flexible massive data set paradigm.
- Algorithm may make a small, constant number of passes over the data.
- Memory usage should be a constant, independent of $n$.
- We will be concerned with trading the number of passes and memory.
Generative Clustering: Mixtures of Distributions

- \( k \) distributions \( F_1, \ldots, F_k \) over the same universe \( \Omega \).
- Weight \( w_i \geq 0 \) for each \( F_i \), such that \( \sum_{i=1}^{k} w_i = 1 \).
- We “add” these \( k \) distributions to create a new distribution called a mixture.
- The mixture is given by: \( \sum w_i F_i \).
Our Problem

Assume input generated by a mixture of $k$ uniform distributions over $\mathbb{R}$: each $F_i$ is uniform over some continuous interval $(a_i, b_i) \subset \mathbb{R}$.
Our Problem

Assume input generated by a mixture of \( k \) uniform distributions over \( \mathbb{R} \): each \( F_i \) is uniform over some continuous interval \((a_i, b_i) \subset \mathbb{R}\).

Our problem: Given a set of samples in an input array (ordered arbitrarily), learn the density function \( F \) of the mixture.
Our Problem

Assume input generated by a mixture of $k$ uniform distributions over $\mathbb{R}$: each $F_i$ is uniform over some continuous interval $(a_i, b_i) \subset \mathbb{R}$.

Our problem: Given a set of samples in an input array (ordered arbitrarily), learn the density function $F$ of the mixture.

Find a function $G$ such that $\int_{\Omega} |F - G| \leq \epsilon$. 
Our Problem

Assume input generated by a mixture of $k$ uniform distributions over $\mathbb{R}$: each $F_i$ is uniform over some continuous interval $(a_i, b_i) \subset \mathbb{R}$.

Our problem: Given a set of samples in an input array (ordered arbitrarily), learn the density function $F$ of the mixture.

Find a function $G$ such that $\int_\Omega |F - G| \leq \epsilon$.

Can generalize to mixtures of other types of distributions as well.
Motivation

- Unsupervised learning of parameters of generative mixture models from samples is a popular statistical tool.

- Many theory papers consider mixtures of Gaussian distributions in high dimension: learning means, mixing weights, and covariance matrices [Dasgupta99], [AK01], [VW02], etc.
Our results

Suppose we are given samples from a mixture of $k$ uniform distributions in a read-only input array $X$. For any $\ell > 0$,

- A $2\ell$ pass algorithm with error $\epsilon^\ell$ that uses $\tilde{O}(k^3/\epsilon^2 + \ell k/\epsilon)$ RAM for learning mixtures of uniform distributions.
Our results

Suppose we are given samples from a mixture of \( k \) uniform distributions in a read-only input array \( X \). For any \( \ell > 0 \),

- A 2\( \ell \) pass algorithm with error \( \epsilon^\ell \) that uses \( \tilde{O}(k^3/\epsilon^2 + \ell k/\epsilon) \) RAM for learning mixtures of uniform distributions.
- or 2\( \ell \) pass algorithm with error \( \epsilon \) that uses \( \tilde{O}(k^3/\epsilon^{2/\ell}) \) RAM.
Our results

Suppose we are given samples from a mixture of $k$ uniform distributions in a read-only input array $X$. For any $\ell > 0$,

- A $2\ell$ pass algorithm with error $\epsilon^\ell$ that uses $\tilde{O}(k^3/\epsilon^2 + \ell k/\epsilon)$ RAM for learning mixtures of uniform distributions.
- or $2\ell$ pass algorithm with error $\epsilon$ that uses $\tilde{O}(k^3/\epsilon^{2/\ell})$ RAM.
- A lower bound: Any $\ell$ pass, randomized algorithm with error $\epsilon$ needs $\Omega(1/\epsilon^{1/(2\ell-1)} c^{1-2\ell})$ bits of RAM.
Our results

Suppose we are given samples from a mixture of $k$ uniform distributions in a read-only input array $X$. For any $\ell > 0$,

- A $2\ell$ pass algorithm with error $\epsilon^\ell$ that uses $\tilde{O}(k^3/\epsilon^2 + \ell k/\epsilon)$ RAM for learning mixtures of uniform distributions.
- or $2\ell$ pass algorithm with error $\epsilon$ that uses $\tilde{O}(k^3/\epsilon^{2/\ell})$ RAM.
- A lower bound: Any $\ell$ pass, randomized algorithm with error $\epsilon$ needs $\Omega(1/\epsilon^{1/(2\ell-1)}c^{1-2\ell})$ bits of RAM.
- Generalization of our algorithm to mixtures of linear distributions, and two dimensional uniform distributions.
Our results

Suppose we are given samples from a mixture of $k$ uniform distributions in a read-only input array $X$. For any $\ell > 0$,

- A $2\ell$ pass algorithm with error $\epsilon^\ell$ that uses $\tilde{O}(k^3/\epsilon^2 + \ell k/\epsilon)$ RAM for learning mixtures of uniform distributions.
- or $2\ell$ pass algorithm with error $\epsilon$ that uses $\tilde{O}(k^3/\epsilon^{2/\ell})$ RAM.
- A lower bound: Any $\ell$ pass, randomized algorithm with error $\epsilon$ needs $\Omega(1/\epsilon^{1/(2\ell-1)} c^{1-2\ell})$ bits of RAM.
- Generalization of our algorithm to mixtures of linear distributions, and two dimensional uniform distributions.

These algorithms have failure probabilities of $1 - \delta$. 
What does a mixture of $k$ uniform distributions look like?

Thus, our algorithm will assume the sample is drawn from distribution with density given by a step function with $2k - 1$ jumps.
The Algorithm: First Attempt

- Obvious thing to do: Break the domain into bins, and count the number of points in each bin. Estimate $F$ on each bin.
The Algorithm: First Attempt

- Obvious thing to do: Break the domain into bins, and count the number of points in each bin. Estimate $F$ on each bin.
- If you want $\epsilon^\ell$ error, then you will need to store $\Omega(\frac{1}{\epsilon^{2\ell}})$ counters.
The Algorithm: First Attempt

- Obvious thing to do: Break the domain into bins, and count the number of points in each bin. Estimate $F$ on each bin.
- If you want $\epsilon^\ell$ error, then you will need to store $\Omega\left(\frac{1}{\epsilon^{2\ell}}\right)$ counters.
- Too much! We can do much better by making a few more passes.
The Algorithm

Our $2\ell$-pass algorithm for learning $F$ to within error $\epsilon^\ell$:

1. In one pass, draw a sample of size $m = \theta(k^2/\epsilon^2)$. 

Memory usage is $\tilde{O}(k^3/\epsilon^2 + \ell k/\epsilon)$. Requires $|X| = \tilde{\Omega}(k^6 \epsilon^6 \ell \cdot \ell)$ points from $F$. 
The Algorithm

Our $2\ell$-pass algorithm for learning $F$ to within error $\epsilon^\ell$:
1. In one pass, draw a sample of size $m = \theta(k^2/\epsilon^2)$.
2. Partition domain into $2k/\epsilon$ intervals s.t. $\int I F = \Theta(\epsilon/2k)$. 

Memory usage is $\tilde{O}(k^3/\epsilon^2 + \ell k/\epsilon)$.
Requires $|X| = \tilde{\Omega}(k^6 \epsilon^6 \ell \cdot \ell)$ points from $F$. 
The Algorithm

Our $2\ell$-pass algorithm for learning $F$ to within error $\epsilon^\ell$:

1. In one pass, draw a sample of size $m = \theta(k^2/\epsilon^2)$.
2. Partition domain into $2k/\epsilon$ intervals s.t. $\int_I F = \Theta(\epsilon/2k)$.
3. In one pass, for all $I$, determine if $F$ is very close to constant on $I$. Also count the number of points of $X$ that lie in $I$. 

4. If $F$ is constant on $I$, then $|X \cap I|/|X| \text{length}(I)$ is close to $F$.

5. If not uniform, recurse on $I$ (Zoom in on the trouble).

Memory usage is $\tilde{O}(k^3/\epsilon^2 + \ell k/\epsilon)$.

Requires $|X| = \tilde{\Omega}(k^6 \epsilon^6 \ell \cdot \ell)$ points from $F$. 
The Algorithm

Our $2\ell$-pass algorithm for learning $F$ to within error $\epsilon^\ell$:

1. In one pass, draw a sample of size $m = \theta(k^2/\epsilon^2)$.
2. Partition domain into $2k/\epsilon$ intervals s.t. $\int_I F = \Theta(\epsilon/2k)$.
3. In one pass, for all $I$, determine if $F$ is very close to constant on $I$. Also count the number of points of $X$ that lie in $I$.
4. If $F$ is constant on $I$, then $|X \cap I|/|X|\text{length}(I)$ is close to $F$. 

Memory usage is $\tilde{O}(k^3/\epsilon^2 + \ell k/\epsilon)$.
Requires $|X| = \tilde{\Omega}(k^6 \epsilon^6 \ell \cdot \ell)$ points from $F$. 
The Algorithm

Our $2\ell$-pass algorithm for learning $F$ to within error $\epsilon^\ell$:

1. In one pass, draw a sample of size $m = \theta(k^2/\epsilon^2)$.
2. Partition domain into $2k/\epsilon$ intervals s.t. $\int_I F = \Theta(\epsilon/2k)$.
3. In one pass, for all $I$, determine if $F$ is very close to constant on $I$. Also count the number of points of $X$ that lie in $I$.
4. If $F$ is constant on $I$, then $|X \cap I|/|X|\text{length}(I)$ is close to $F$.
5. If not uniform, recurse on $I$ (Zoom in on the trouble).
The Algorithm

Our $2\ell$-pass algorithm for learning $F$ to within error $\epsilon^\ell$:

1. In one pass, draw a sample of size $m = \theta(k^2/\epsilon^2)$.
2. Partition domain into $2k/\epsilon$ intervals s.t. $\int_I F = \Theta(\epsilon/2k)$.
3. In one pass, for all $I$, determine if $F$ is very close to constant on $I$. Also count the number of points of $X$ that lie in $I$.
4. If $F$ is constant on $I$, then $|X \cap I|/|X|\text{length}(I)$ is close to $F$.
5. If not uniform, recurse on $I$ (Zoom in on the trouble).

Memory usage is $\tilde{O}(k^3/\epsilon^2 + \ell k/\epsilon)$. 
The Algorithm

Our $2\ell$-pass algorithm for learning $F$ to within error $\epsilon^\ell$:

1. In one pass, draw a sample of size $m = \theta(k^2/\epsilon^2)$.
2. Partition domain into $2k/\epsilon$ intervals s.t. $\int_I F = \Theta(\epsilon/2k)$.
3. In one pass, for all $I$, determine if $F$ is very close to constant on $I$. Also count the number of points of $X$ that lie in $I$.
4. If $F$ is constant on $I$, then $|X \cap I|/|X| \text{length}(I)$ is close to $F$.
5. If not uniform, recurse on $I$ (Zoom in on the trouble).

Memory usage is $\tilde{O}(k^3/\epsilon^2 + \ell k/\epsilon)$.

Requires $|X| = \tilde{\Omega} \left( \frac{k^6}{\epsilon^6 \ell} \cdot \ell \right)$ points from $F$. 
Zooming In

A Function $F$.

B Partition the domain into intervals.

C Recurse on non-constant interval. It is sampled at a higher rate.
Why it works

- Memory requirement is small: since we only recurse on $2k$ jumps, the number of bins created at each iteration is at most $O(k^2/\epsilon)$.
- Easy to learn $F$ when it is constant. Our estimate is very accurate.
- The weight of bins decreases exponentially at each iteration.
- At the $\ell$th iteration, bins have weight $\epsilon^\ell/4k$.
- Thus, we can estimate $F$ as 0 on $2k$ bins where there is a jump, and incur a total error of at most $\epsilon^\ell/2$. 
Generalizations

Can generalize above algorithm to the following problems, with roughly the same memory usage:

- Uniform distributions over $\mathbb{R}^2$: $F_i$ is uniform over some axis-aligned rectangle: $(a_i, b_i) \times (c_i, d_i) \subset \mathbb{R}^2$.
- Linear distributions over $\mathbb{R}$: The density of $F_i$ is a linear function over some continuous interval $(a_i, b_i) \subset \mathbb{R}$.
Subproblem: Suppose $H$ is the pdf of a mixture of $k$ uniform distributions over an interval $I$, with samples in input array $X$. Determine if $\int_I |H - 1| \leq \frac{\epsilon}{2k}$.
Subproblem: Suppose $H$ is the pdf of a mixture of $k$ uniform distributions over an interval $I$, with samples in input array $X$. Determine if $\int_I |H - 1| \leq \frac{\epsilon \ell}{2k}$.

First Attempt:

1. Partition $I$ into many bins of equal length. Number of bins is $5k^2/\epsilon \ell$.
2. In one pass, determine the number of points from $X$ that lie in each bin.
3. If number of points is roughly equal in each bin, then accept. Otherwise, reject.
Subproblem: Suppose $H$ is the pdf of a mixture of $k$ uniform distributions over an interval $I$, with samples in input array $X$. Determine if $\int_I |H - 1| \leq \epsilon \ell / 2k$.

First Attempt:

1. Partition $I$ into many bins of equal length. Number of bins is $5k^2/\epsilon \ell$.

2. **In one pass, determine the number of points from $X$ that lie in each bin.**

3. If number of points is roughly equal in each bin, then accept. Otherwise, reject.
Subproblem: Suppose $H$ is the pdf of a mixture of $k$ uniform distributions over an interval $I$, with samples in input array $X$. Determine if $\int_I |H - 1| \leq \epsilon^\ell / 2k$.

First Attempt:

1. Partition $I$ into many bins of equal length. Number of bins is $5k^2 / \epsilon^\ell$.

2. **In one pass, determine the number of points from $X$ that lie in each bin.**

3. If number of points is roughly equal in each bin, then accept. Otherwise, reject.

Problem: This will take at least $5k^2 / \epsilon^\ell$ bits of memory. Too much!
Maintaining $\ell_1$ length of a vector

[Indyk00] designed a streaming algorithm for maintaining the $\ell_1$ length of a vector $v$.

- Data stream consists of pairs $(i, a)$, where $i \in [n]$, $a \in \{-M, \ldots, M\}$.
- Vector $v \in \mathbb{R}^n$ given by: $v_i = \sum_{(i,a)} a$.
- With probability $1 - \delta$, Indyk’s algorithm will approximate $\|v\|_1$ within a constant factor in one pass using at most $O(\log M \log(n/\delta) \log \delta)$ bits of memory.
Sketch of our algorithm

If $X$ is sufficiently large,

- Partition $I$ into $B = \frac{5k^2}{\epsilon^\ell}$ bins of equal length.
Sketch of our algorithm

If $X$ is sufficiently large,

- Partition $I$ into $B = 5k^2/\epsilon^\ell$ bins of equal length.
- Let $n_i$ be the number of points of $X$ in the $i$th bin. If $H$ is uniform on $I$, then expect $n_i \approx |X|/B$. Let $\alpha_i = n_i - \eta|x|$ be the difference.
Sketch of our algorithm

If $X$ is sufficiently large,

- Partition $I$ into $B = 5k^2/\epsilon^\ell$ bins of equal length.
- Let $n_i$ be the number of points of $X$ in the $i$th bin. If $H$ is uniform on $I$, then expect $n_i \approx |X|/B$. Let $\alpha_i = n_i - \eta|x|$ be the difference.
- If $F$ is uniform on $I$, then $||\alpha||_1$ will be very small. If $H$ is more than $\epsilon^\ell$ in distance from uniform, then $||\alpha||_1$ will be large.
Sketch of our algorithm

If $X$ is sufficiently large,

- Partition $I$ into $B = 5k^2/\epsilon^\ell$ bins of equal length.
- Let $n_i$ be the number of points of $X$ in the $i$th bin. If $H$ is uniform on $I$, then expect $n_i \approx |X|/B$. Let $\alpha_i = n_i - \eta |x|$ be the difference.

- If $F$ is uniform on $I$, then $\|\alpha\|_1$ will be very small. If $H$ is more than $\epsilon^\ell$ in distance from uniform, then $\|\alpha\|_1$ will be large.

- In one pass over $X$, use Indyk’s algorithm to approximate $\|\alpha\|_1$. Reject if estimate is too large.
Sketch of our algorithm

If $X$ is sufficiently large,

- Partition $I$ into $B = 5k^2/\epsilon^\ell$ bins of equal length.
- Let $n_i$ be the number of points of $X$ in the $i$th bin. If $H$ is uniform on $I$, then expect $n_i \approx |X|/B$. Let $\alpha_i = n_i - \eta|x|$ be the difference.
- If $F$ is uniform on $I$, then $||\alpha||_1$ will be very small. If $H$ is more than $\epsilon^\ell$ in distance from uniform, then $||\alpha||_1$ will be large.
- In one pass over $X$, use Indyk’s algorithm to approximate $||\alpha||_1$. Reject if estimate is too large.

Uses $O((\log k + \ell \log(1/\epsilon)) \log(1/\delta))$ bits of memory.
Proving Lower Bounds for the Generalized Learning Problem

Prove lower bounds for slightly stronger problem:

- $F$ is the pdf of a mixture of $1/\epsilon^\ell$ uniform distributions.
- Let $t \in [0, 1]$ be the largest number such that $F$ is a step distribution with at most $k$ steps on $[0, t]$.
- Find a function $G$ and number $t' > t$ such that
  \[ \int_0^{t'} |F - G| < \alpha. \]
**Theorem:** Any \( \ell \)-pass randomized algorithm that solves the Generalized Learning Problem for \( k = 3 \) and error \( \epsilon^\ell \) must use at least \( \Omega\left(\frac{1}{\epsilon^{1/2} c^{-2\ell+1}}\right) \) bits of memory.
**Theorem:** Any $\ell$-pass randomized algorithm that solves the Generalized Learning Problem for $k = 3$ and error $\epsilon^\ell$ must use at least $\Omega(1/\epsilon^{1/2} c^{-2\ell+1})$ bits of memory.

There exists an $\ell$-pass algorithm that will solve the problem using at most $\tilde{O}(1/\epsilon^4)$ bits of memory.
Alternatively:

- **Lower Bound**: For error $\epsilon$, must use at least $\Omega(1/\epsilon^{1/(2\ell-1)}c^{-2\ell+1})$
- **Upper bound**: For error $\epsilon$, can solve the problem using at most $\tilde{O}(1/\epsilon^{4/\ell})$. 
A communication problem

- Two players: Alice and Bob receive $a, b \in \{0, 1\}^n$ respectively.
- Neither player knows the other’s input.
- They want to compute $\text{GT}_n(a, b) = 1$ if $a > b$, 0 otherwise. May pass $r$ messages to each other to do so.
- Let $R^r(\text{GT}_n)$ denote the size of the largest message that must be passed. Known: $R^r(\text{GT}_n) = \Omega(n^{1/r}c^{1-r})$ [MNSW98].
Main Idea of Proving Lower Bound

- Assume there exists an $\ell$-pass algorithm $P$ that solves GLP with error $\epsilon^\ell$ and uses $M(P)$ bits of memory.
- Show that above algorithm will induce a $2\ell - 1$ round protocol for communication game that solves GT problem for $n = 1/\epsilon^\ell$.
- Then show that $M(P) \geq R^{2\ell-1}(GT_n) = \Omega(1/\epsilon^{1/2}c^{-2\ell+1})$. 
Inducing a protocol

▶ Suppose Alice gets the string $a_1 a_2 a_3 = 001$ and Bob gets the string $b_1 b_2 b_3 = 000$. 
Inducing a protocol

- Suppose Alice gets the string $a_1 a_2 a_3 = 001$ and Bob gets the string $b_1 b_2 b_3 = 000$.
- They each construct a pdf:

\[
\begin{align*}
\text{Alice} & : \quad a_1 a_3 a_2 = 1 0 1 0 1 \\
\text{Bob} & : \quad b_1 b_2 b_3 = 000 11 1
\end{align*}
\]
Suppose Alice gets the string $a_1 a_2 a_3 = 001$ and Bob gets the string $b_1 b_2 b_3 = 000$.

They each construct a pdf:

Alice

```
  0  a1  1  0  a2  1  0  a3  1
```

Bob

```
  1  b1  0  1  b2  0  1  b3  0
```

They produce samples from their respective distribution and simulate a *single* input array for GLP. Alice and Bob each have one half of the input array.
They can simulate $P$ on the entire input array, which contains samples from the “sum” of these two distributions.
Protocol continued

- They can simulate $P$ on the entire input array, which contains samples from the “sum” of these two distributions.
- Sum is a mixture of $n$ uniform distributions.
Protocol continued

- They can simulate $P$ on the entire input array, which contains samples from the “sum” of these two distributions.
- Sum is a mixture of $n$ uniform distributions.
- By learning the first three steps of this distribution, they can deduce who has the larger number.
Example, continued

\[ a_1 a_2 a_3 = 001 \text{ and } b_1 b_2 b_3 = 000. \]
How to Simulate $P$ by Passing Messages

- Technique from [AMS96].
- Alice simulates $P$ on her half, sends $P$’s memory to Bob. Bob simulates the rest of the pass on his half, and sends the memory back to Alice. Continue like this for $\ell$ passes.
- They pass a total of $2\ell - 1$ messages of size at most $M(P)$. Thus, $M(P) \geq R^{2\ell-1}(\text{GT}_n)$. 

Future Work

- Generalize the algorithm to learn mixtures of uniform distributions in dimension greater than 2.
- Design pass-efficient algorithms for learning mixtures of Normal Distributions in high dimension.
- **Heuristic** for learning mixture of 1-D normal distributions: Approximate mixture as $k$ piecewise constant or uniform, and use our algorithm. See how this works empirically.
Thanks for listening!