BICO: BIRCH meets Coresets for \( k \)-means

Hendrik Fichtenberger, Marc Gillé, Melanie Schmidt, Chris Schwiegelshohn, Christian Sohler
The \textit{k}-means Problem

- Given a point set $P \subseteq \mathbb{R}^d$, compute a set $C \subseteq \mathbb{R}^d$ with $|C| = k$ centers which minimizes $\text{cost}(P, C) = \sum_{p \in P \atop c \in C} \min ||c - p||^2$, the sum of the squared distances.
Popular $k$-means algorithms... 

- Lloyd’s algorithm (1982)
- $k$-means++ (2007)
- several approximation algorithms (recent)
Popular $k$-means algorithms...

- Lloyd’s algorithm (1982)
- $k$-means++ (2007)
- several approximation algorithms (recent)

... for Big Data

- BIRCH (1996)
- MacQueen’s $k$-means (1967)
- several approximations using coresets (recent)
Popular $k$-means algorithms…

- Lloyd’s algorithm (1982)
- $k$-means++ (2007)
- several approximation algorithms (recent)

… for Big Data

- BIRCH (1996)
- MacQueen’s $k$-means (1967)
- several approximations using coresets (recent)

Implementability, Speed and good quality?
Popular $k$-means algorithms...

- Lloyd’s algorithm (1982) moderate speed
- $k$-means++ (2007) moderate speed & quality
- several approximation algorithms (recent) quality

...for Big Data

- BIRCH (1996) fast
- MacQueen’s $k$-means (1967) fast
- several approximations using coresets (recent) quality

Implementability, Speed and good quality?
Popular $k$-means algorithms...

- Lloyd’s algorithm (1982)  moderate speed
- $k$-means++ (2007) moderate speed & quality
- several approximation algorithms (recent) quality

...for Big Data

- BIRCH (1996) fast
- MacQueen’s $k$-means (1967) fast
- several approximations using coresets (recent) quality

Implementability, Speed and good quality?

- Stream-KM++ (2010) next slides

BICO: BIRCH meets Coresets for $k$-means
### Popular $k$-means algorithms...

- Lloyd’s algorithm (1982)  
  moderate speed
- $k$-means++ (2007)  
  moderate speed & quality
- several approximation algorithms (recent)  
  quality

### ...for Big Data

- BIRCH (1996)  
  fast
- MacQueen’s $k$-means (1967)  
  fast
- several approximations using coresets (recent)  
  quality

### Implementability, Speed and good quality?

- Stream-KM++ (2010)  
  next slides
- BICO  
  next slides
BICO: BIRCH meets Coresets for $k$-means
Running Time

BICO: BIRCH meets Coresets for $k$-means
How BIRCH computes a summary of the data

**Idea**

- start with BIRCH for the basic design because it is very fast
- analyze its flaws
- develop an improved algorithm based on theoretical observations
How BIRCH computes a summary of the data

**Idea**

- start with BIRCH for the basic design because it is very fast
- analyze its flaws
- develop an improved algorithm based on theoretical observations

**Now**: Description of BIRCH
Idea

- start with BIRCH for the basic design because it is very fast
- analyze its flaws
- develop an improved algorithm based on theoretical observations

Now: Description of BIRCH

Warning

- BIRCH has several phases
- we are only interested in the main phase
- (and a little in the rebuilding phase)
How BIRCH computes a summary of the data

BIRCH

stores points in a tree
each node represents a subset of the input point set
subset is summarized by the number of points, the centroid of the set and the squared distances to the centroid
all points in the same subset get the same center

BICO: BIRCH meets Coresets for $k$-means
How BIRCH computes a summary of the data

**BIRCH**

- stores points in a tree
How BIRCH computes a summary of the data

**BIRCH**

- stores points in a tree
- each node represents a subset of the input point set
How BIRCH computes a summary of the data

BIRCH

- stores points in a tree
- each node represents a subset of the input point set
- subset is summarized by the number of points, the centroid of the set and the squared distances to the centroid
How BIRCH computes a summary of the data

**BIRCH**
- stores points in a tree
- each node represents a subset of the input point set
- subset is summarized by the number of points, the centroid of the set and the squared distances to the centroid
- all points in the same subset get the same center
How BIRCH computes a summary of the data

**BIRCH**

- nodes in the tree represent subsets of points
- points at the same node get the same center
How BIRCH computes a summary of the data

**BIRCH**
- nodes in the tree represent subsets of points
- points at the same node get the same center

**Insertion of a new point**

When a new point $p$ is added to the tree
How BIRCH computes a summary of the data

**BIRCH**
- nodes in the tree represent subsets of points
- points at the same node get the same center

**Insertion of a new point**

When a new point \( p \) is added to the tree
- BIRCH searches for the ‘closest’ node according to
  \[
  \sum_{q \in (S \cup \{p\})} (q - \mu(S \cup \{p\}))^2 - \sum_{q \in S} (q - \mu(S))^2
  \]
Introduction

Insights from BIRCH

Coreset Theory

BICO

Experiments

End

How BIRCH computes a summary of the data

**BIRCH**

- nodes in the tree represent subsets of points
- points at the same node get the same center

**Insertion of a new point**

When a new point $p$ is added to the tree

- BIRCH searches for the ‘closest’ node according to
  \[
  \sum_{q \in (S \cup \{p\})} (q - \mu(S \cup \{p\}))^2 - \sum_{q \in S} (q - \mu(S))^2
  \]

- $p$ is added to the node representing subset $S^*$ if
  \[
  \sum_{q \in (S^* \cup \{p\})} (q - \mu_S)^2 / (|S^*| + 1) \leq T^2
  \]
  for a given threshold $T$
How BIRCH computes a summary of the data

- 150 points drawn uniformly around \((-0.5, 0)\) and \((0, 0.5)\)
- 75 points drawn uniformly from \([-4, -2] \times [4, 2]\) as noise
- Centers and partitions computed by BIRCH and BICO

BICO: BIRCH meets Coresets for \(k\)-means
How BIRCH computes a summary of the data

- 150 points drawn uniformly around \((-0.5, 0)\) and \((0, 0.5)\)
- 75 points drawn uniformly from \([-4, -2] \times [4, 2]\) as noise
- Centers and partitions computed by BIRCH and BICO
How BIRCH computes a summary of the data

150 points drawn uniformly around \((-0.5, 0)\) and \((0, 0.5)\)

75 points drawn uniformly from \([-4, -2] \times [4, 2]\) as noise

Centers and partitions computed by BIRCH and BICO
Insights from BIRCH

- Fast point by point updates
- Tree structure
How BIRCH computes a summary of the data

Insights from BIRCH

- Fast point by point updates
- Tree structure
- Insertion decision should be improved

BICO: BIRCH meets Coresets for $k$-means
Coresets

- small summary of given data
- typically of constant or polylogarithmic size
- can be used to approximate the cost of the original data
Coresets

Given a set of points $P$, a weighted subset $S \subset P$ is a $(k, \varepsilon)$-coreset if for all sets $C \subset C$ of $k$ centers it holds

$$\left| \text{cost}_w(S, C) - \text{cost}(P, C) \right| \leq \varepsilon \text{cost}(P, C)$$

where $\text{cost}_w(S, C) = \sum_{p \in S} \min_{c \in C} w(p)(p, c)$. 

BICO: BIRCH meets Coresets for $k$-means
Coresets

Given a set of points \( P \), a weighted subset \( S \subseteq P \) is a \((k, \epsilon)\)-coreset if for all sets \( C \subseteq \mathcal{C} \) of \( k \) centers it holds

\[
|\text{cost}_w(S, C) - \text{cost}(P, C)| \leq \epsilon \text{cost}(P, C)
\]

where \( \text{cost}_w(S, C) = \sum_{p \in S} \min_{c \in C} w(p)(p, c) \).
Coresets

Given a set of points $P$, a weighted subset $S \subset P$ is a \((k, \epsilon)\)-coreset if for all sets $C \subset C$ of $k$ centers it holds

$$|\text{cost}_w(S, C) - \text{cost}(P, C)| \leq \epsilon \text{cost}(P, C)$$

where $\text{cost}_w(S, C) = \sum_{p \in S} \min_{c \in C} w(p)(p, c)$. 

BICO: BIRCH meets Coresets for $k$-means
Coresets

Given a set of points $P$, a weighted subset $S \subset P$ is a $(k, \epsilon)$-coreset if for all sets $C \subset C$ of $k$ centers it holds

$$|\text{cost}_w(S, C) - \text{cost}(P, C)| \leq \epsilon \text{cost}(P, C)$$

where $\text{cost}_w(S, C) = \sum_{p \in S} \min_{c \in C} w(p)(p, c)$. 

BICO: BIRCH meets Coresets for $k$-means
Coresets

Given a set of points $P$, a weighted subset $S \subset P$ is a $(k, \epsilon)$-coreset if for all sets $C \subset C$ of $k$ centers it holds

$$|\text{cost}_w(S, C) - \text{cost}(P, C)| \leq \epsilon \text{cost}(P, C)$$

where $\text{cost}_w(S, C) = \sum_{p \in S} \min_{c \in C} w(p)(p, c)$.
Coresets

Given a set of points $P$, a weighted subset $S \subset P$ is a $(k, \epsilon)$-coreset if for all sets $C \subset \mathcal{C}$ of $k$ centers it holds

$$|\text{cost}_w(S, C) - \text{cost}(P, C)| \leq \epsilon \text{cost}(P, C)$$

where $\text{cost}_w(S, C) = \sum_{p \in S} \min_{c \in C} w(p)(p, c)$. 
**Merge & Reduce**

- read data in blocks
- compute a coreset for each block $\rightarrow s$
- merge coresets in a tree fashion
- $\Leftrightarrow$ space $s \cdot \log n$
Coresets

Merge & Reduce

- read data in blocks
- compute a coreset for each block → $s$
- merge coresets in a tree fashion
- $\leadsto$ space $s \cdot \log n$

Runtime: No asymptotic increase, but overhead in practice

BIRCH uses point-wise updates :-)

BICO: BIRCH meets Coresets for $k$-means
Insights from Coreset Theory

- Limit the induced error

\[ \Rightarrow \text{Goal: Point set } P' \text{ in each node should induce at most } \varepsilon \cdot \text{cost}(P', C) \text{ error (for an optimal solution } C) \]

\[ \Rightarrow \text{Base insertion decision on induced error} \]

- Replacing all points in a node by the (weighted) centroid is like moving all points to the centroid

- Induced error is connected to the 1-means cost of the set
Insights from Coreset Theory

- Limit the induced error
- Goal: Point set $P'$ in each node should induce at most $\varepsilon \cdot \text{cost}(P', C)$ error (for an optimal solution $C$)
- Base insertion decision on induced error
  - Replacing all points in a node by the (weighted) centroid is like moving all points to the centroid
  - Induced error is connected to the 1-means cost of the set

Side note

- Avoiding Merge & Reduce is a good idea
BIRCH

- nodes in the tree represent subsets of points
- points at the same node get the same center
- improve insertion decision
BIRCH meets Coresets

**BIRCH**
- nodes in the tree represent subsets of points
- points at the same node get the same center
- improve insertion decision

**Adjustments**
- Nodes additionally have a reference point and a range
BIRCH

- nodes in the tree represent subsets of points
- points at the same node get the same center
- improve insertion decision

Adjustments

- Nodes additionally have a reference point and a range
- Closest is now determined by Euclidean distance

BICO: BIRCH meets Coresets for $k$-means
BIRCH meets Coresets

**BIRCH**
- nodes in the tree represent subsets of points
- points at the same node get the same center
- improve insertion decision

**Adjustments**
- Nodes additionally have a reference point and a range
- **Closest** is now determined by Euclidean distance
- We say a node is **full** with regard to a point $p$ if adding $p$ to the node increases its 1-means cost above a threshold $T$
BIRCH meets Coresets

BICO: BIRCH meets Coresets for \( k \)-means

1. Find closest reference point
2. If node is not in range
3. Then create a new node
4. Else add to node if possible
5. If not, go one level down,
6. And find closest child, goto 2.

Threshold \( T \)
Radius \( R \)
BIRCH meets Coresets

1. Find closest reference point
2. If node is not in range
3. Then create a new node
4. Else add to node if possible
5. If not, go one level down,
6. And Find closest child, goto 2.

Threshold $T$
Radius $R_i$
BIRCH meets Coresets

1. Find closest reference point
2. If node is not in range
3. Then create a new node
4. Else add to node if possible
5. If not, go one level down,
6. And Find closest child, goto 2.

Threshold $T$
Radius $R_i$
1. Find closest reference point
2. If node is not in range
3. Then create a new node
4. Else add to node if possible
5. If not, go one level down,
6. And Find closest child, goto 2.

Threshold $T$
Radius $R_i$
1. Find closest reference point
2. If node is not in range
3. Then create a new node
4. Else add to node if possible
5. If not, go one level down,
6. And Find closest child, goto 2.
BIRCH meets Coresets

1. Find closest reference point
2. If node is not in range
3. Then create a new node
4. Else add to node if possible
5. If not, go one level down,
6. And Find closest child, goto 2.
Theorem

For $T \approx \frac{OPT}{k \cdot \log n \cdot 8^d \cdot \varepsilon^{d+2}}$ and $R_i := \sqrt{\frac{T}{8 \cdot 2^i}}$,

- the set of centroids weighted by the number of points in the subset is a $(1 + \varepsilon)$-coreset
- for constant $d$, the number of nodes is $\mathcal{O}(k \cdot \log n \cdot \varepsilon^{-(d+2)})$
Theorem

For $T \approx \frac{OPT}{k \cdot \log n \cdot 8^d \cdot \varepsilon^{d+2}}$ and $R_i := \sqrt{\frac{T}{8 \cdot 2^i}}$, 
- the set of centroids weighted by the number of points in the subset is a $(1 + \varepsilon)$-coreset 
- for constant $d$, the number of nodes is $O(k \cdot \log n \cdot \varepsilon^{-(d+2)})$ 

Problem

We do not know OPT and thus cannot compute $T$!
**BIRCH meets Coresets**

**Rebuilding algorithm**

- Double $T$ when maximum number of nodes is reached
- ‘Rebuild’ the tree according to new $T$
BIRCH meets Coresets

Rebuilding algorithm

- Double $T$ when maximum number of nodes is reached
- ‘Rebuild’ the tree according to new $T$

- Let $T'$ and $R'_i$ be before and $T$ and $R_i$ be after the doubling
- Move all nodes one level down and create empty first level
- Notice that $R_i = \sqrt{T/(8 \cdot 2^i)} = \sqrt{T'/8 \cdot 2^{i-1}} = R'_{i-1}$

⇒ Radius doesn’t change!
BIRCH meets Coresets

BICO: BIRCH meets Coresets for $k$-means
BIRCH meets Coresets

BICO: BIRCH meets Coresets for $k$-means
BIRCH meets Coresets

BICO: BIRCH meets Coresets for \( k \)-means
BIRCH meets Coresets

BICO: BIRCH meets Coresets for $k$-means
BIRCH meets Coresets

BICO: BIRCH meets Coresets for $k$-means
BIRCH meets Coresets

BICO: BIRCH meets Coresets for $k$-means
BIRCH meets Coresets

For every node in the tree:
1. Find ref. point one level higher
2. If none in range, move node and children one level up
3. If it is in range, check $T$
4. If sum is smaller than $T$, merge nodes
5. Else make it a child
For every node in the tree
1. Find ref. point one level higher
2. If none in range, move node and children one level up
3. If it is in range, check \( T \)
4. If sum is smaller than \( T \), merge nodes
6. Else make it a child
For every node in the tree
1. Find ref. point one level higher
2. If none in range, move node and children one level up
3. If it is in range, check $T$
4. If sum is smaller than $T$, merge nodes
6. Else make it a child
BIRCH meets Coresets

For every node in the tree:
1. Find ref. point one level higher
2. If none in range, move node and children one level up
3. If it is in range, check $T$
4. If sum is smaller than $T$, merge nodes
6. Else make it a child
BIRCH meets Coresets

For every node in the tree
1. Find ref. point one level higher
2. If none in range, move node and children one level up
3. If it is in range, check $T$
4. If sum is smaller than $T$, merge nodes
6. Else make it a child
• Merging might result in violations of the range of nodes
Merging might result in violations of the range of nodes.
Merging might result in violations of the range of nodes.

Increases the coreset size by a constant factor.
BIRCH meets Coresets

- Merging might result in violations of the range of nodes
- Increases the coreset size by a constant factor
- But does not destroy the coreset property
Merging might result in violations of the range of nodes
- Increases the coreset size by a constant factor
- But does not destroy the coreset property

⇒ BICO computes a coreset in the data stream setting 😃
Merging might result in violations of the range of nodes
- Increases the coreset size by a constant factor
- But does not destroy the coreset property

⇒ BICO computes a coreset in the data stream setting 😊

... if we compute lower bound on $T$
The actual solution is computed with $k$-means++. 
The actual solution is computed with $k$-means++.

**Adjustments**

- Set coreset size to $200k$
The actual solution is computed with $k$-means++.

**Adjustments**

- Set coreset size to $200k$
- Add heuristic speed-up to find closest reference point
BICO is cool :-) 

The actual solution is computed with $k$-means++. 

Adjustments 
- Set coreset size to $200k$ 
- Add heuristic speed-up to find closest reference point 

Speed-up 
- Project all ref. points to $d$ random 1-dim. subspaces 
- Project new point $p$ to the same subspaces 
- Count how many ref. points are in range of $p$ in every subspace 
- Search nearest neighbor in the shortest list
The actual solution is computed with $k$-means++.

**Adjustments**

- Set coreset size to $200k$
- Add heuristic speed-up to find closest reference point

**Speed-up**

- Project all ref. points to $d$ random 1-dim. subspaces
- Project new point $p$ to the same subspaces
- Count how many ref. points are in range of $p$ in every subspace
- Search nearest neighbor in the shortest list
Data Sets

- Data Sets used in StreamKM++ paper from UCI repository: Tower, CoverType, Census and BigCross (cross product)
- CalTech128 by René Grzeszick, group of Prof. Fink
- consists of 128 SIFT descriptors of an object database
Data Sets

- Data Sets used in StreamKM++ paper from UCI repository: Tower, CoverType, Census and BigCross (cross product)
- CalTech128 by René Grzeszick, group of Prof. Fink
- consists of 128 SIFT descriptors of an object database

Data Set Sizes

<table>
<thead>
<tr>
<th></th>
<th>BigCross</th>
<th>CalTech128</th>
<th>Census</th>
<th>CoverType</th>
<th>Tower</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>11620300</td>
<td>3168383</td>
<td>2458285</td>
<td>581012</td>
<td>4915200</td>
</tr>
<tr>
<td>$d$</td>
<td>57</td>
<td>128</td>
<td>68</td>
<td>55</td>
<td>3</td>
</tr>
<tr>
<td>$n \cdot d$</td>
<td>662357100</td>
<td>405553024</td>
<td>167163380</td>
<td>31955660</td>
<td>14745600</td>
</tr>
</tbody>
</table>

BICO: BIRCH meets Coresets for $k$-means
Implementations

- Author’s implementations for StreamKM++ and BIRCH
- Implementation for MacQueen’s $k$-means from ESMERALDA (framework by group of Prof. Fink)
Implementations

- Author’s implementations for StreamKM++ and BIRCH
- Implementation for MacQueen’s $k$-means from ESMERALDA (framework by group of Prof. Fink)

Experiments

- Experiments done on mud1-6 and mud8
- 100 runs for every test instance
- Values shown in the diagrams are mean values

BICO: BIRCH meets Coresets for $k$-means
BICO: BIRCH meets Coresets for $k$-means
BICO: BIRCH meets Coresets for k-means
BICO is cool :-)
BICO is cool :-)
BICO is cool :-)

Running time

BICO: BIRCH meets Coresets for k-means
BICO: BIRCH meets Coresets for $k$-means

Running time

CalTech

Number of Centers

CalTech

Number of Centers
Running time

BICO: BIRCH meets Coresets for $k$-means
### Trade-Off

<table>
<thead>
<tr>
<th></th>
<th>BIRCH</th>
<th>MQ</th>
<th>$m = 200k$</th>
<th>$100k$</th>
<th>$25k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>616</td>
<td>4241</td>
<td>20271</td>
<td>5444</td>
<td>618</td>
</tr>
<tr>
<td>Costs</td>
<td>$58 \cdot 10^{10}$</td>
<td>$72 \cdot 10^{10}$</td>
<td>$51 \cdot 10^{10}$</td>
<td>$52 \cdot 10^{10}$</td>
<td>$55 \cdot 10^{10}$</td>
</tr>
</tbody>
</table>

- Tests run on on BigCross with $k = 1000$
- BICO is less than 1.5 times slower than MacQueen with $m = 100k$ while still computing reasonable costs
- Faster than BIRCH for $m = 25k$, still much better cost than BIRCH and MacQueen
BICO is cool :-)

Ziele

- Implementierung in bekanntem Framework / Anbindung
- Baustein für andere Algorithmen
- Vergleich verschiedener Strategien für Nearest Neighbor
BICO: BIRCH meets Coresets for $k$-means

<table>
<thead>
<tr>
<th></th>
<th>BIRCH</th>
<th>MQ</th>
<th>$m = 200k$</th>
<th>100k</th>
<th>25k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Costs</td>
<td>616</td>
<td>4241</td>
<td>20271</td>
<td>5444</td>
<td>618</td>
</tr>
<tr>
<td></td>
<td>$58 \cdot 10^{10}$</td>
<td>$72 \cdot 10^{10}$</td>
<td>$51 \cdot 10^{10}$</td>
<td>$52 \cdot 10^{10}$</td>
<td>$55 \cdot 10^{10}$</td>
</tr>
</tbody>
</table>

Thank you for your attention!