Distributed k-Means and k-Median Clustering on General Topologies

Maria Florina Balcan, Steven Ehrlich, Yingyu Liangz

Presented by:
Aishwarya Anandan (aananda2)
Harshad Rai (hrrai2)
Rachneet Kaur (rk4)
Vipul Satone(vsatone2)

ISE Industrial and Enterprise Systems Engineering

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Introduction

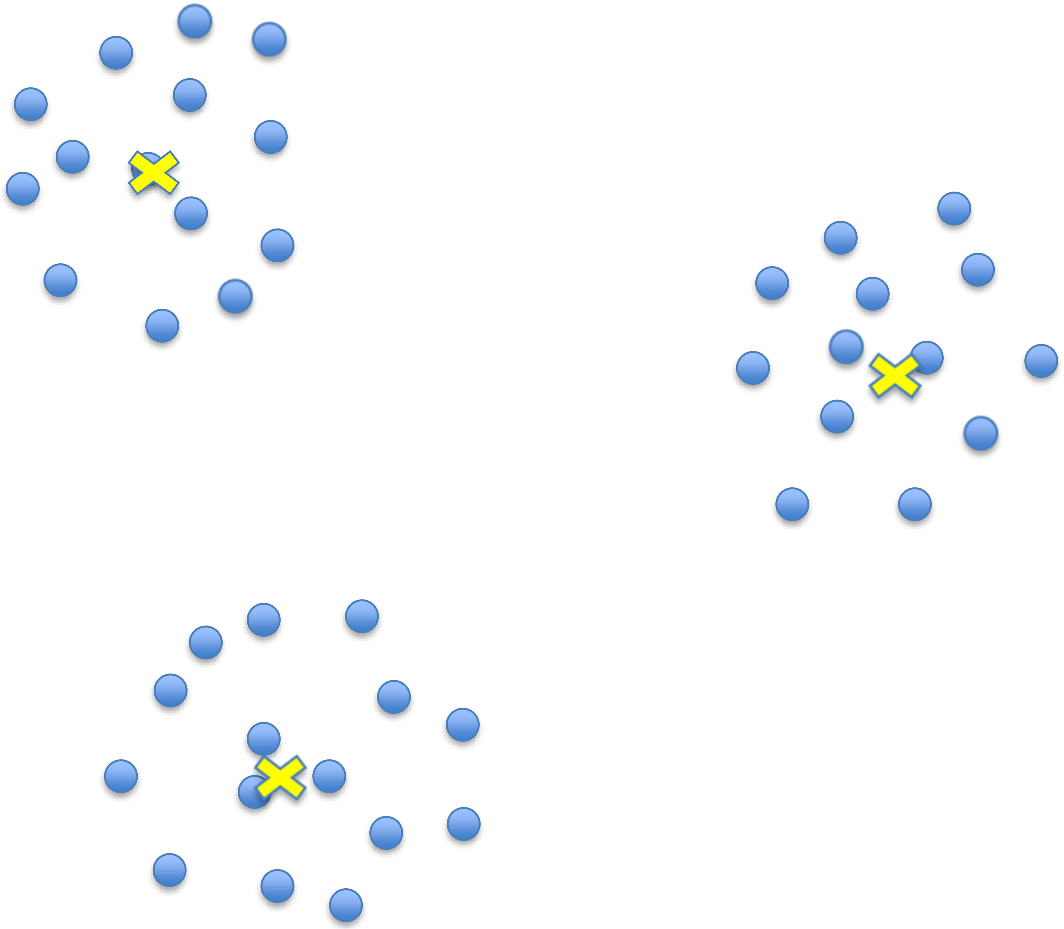
- Era of Big Data
- Data distributed over multiple locations
  - Distributed databases
  - Images and videos over networks
  - Sensors
- Centralized algorithms need to be scaled to distributed settings

Motivation

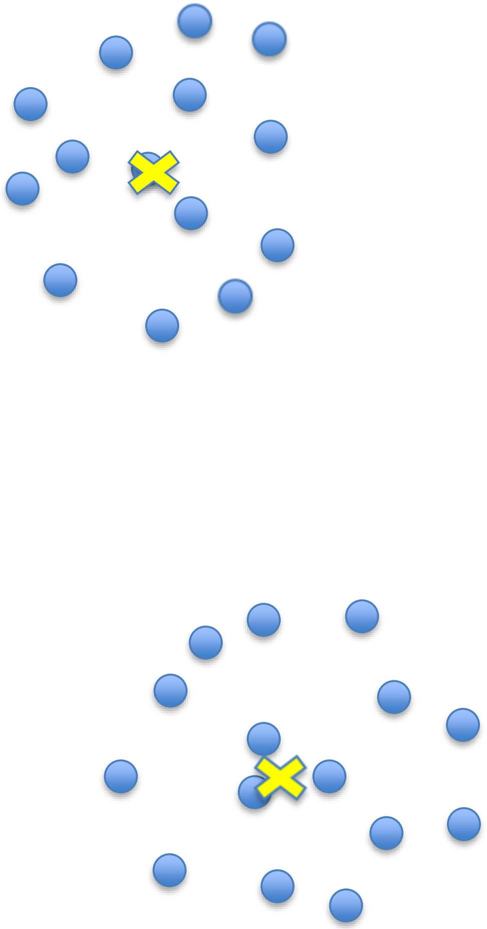
- k-Means and k-Medians are center based clustering algorithms
- Need a method to run k-means and k-medians on distributed data
- Communication cost should be low
- Run clustering algorithms on coreset

What is a coreset/ ϵ-coreset?

- An ϵ-coreset is a weighted set of points whose cost on any set of centers is approximately the cost of the original data on those same centers up to accuracy ϵ.

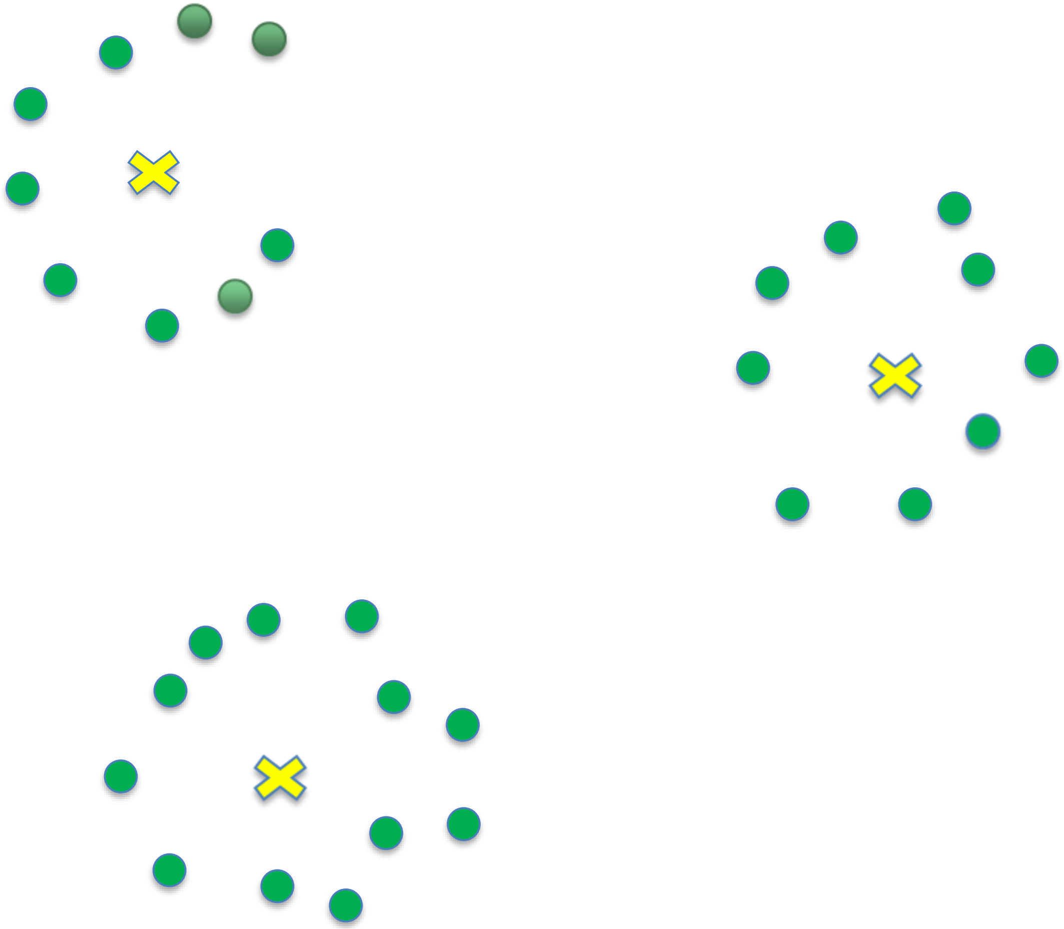


Original data P with centers X



C = Cost(P , X)

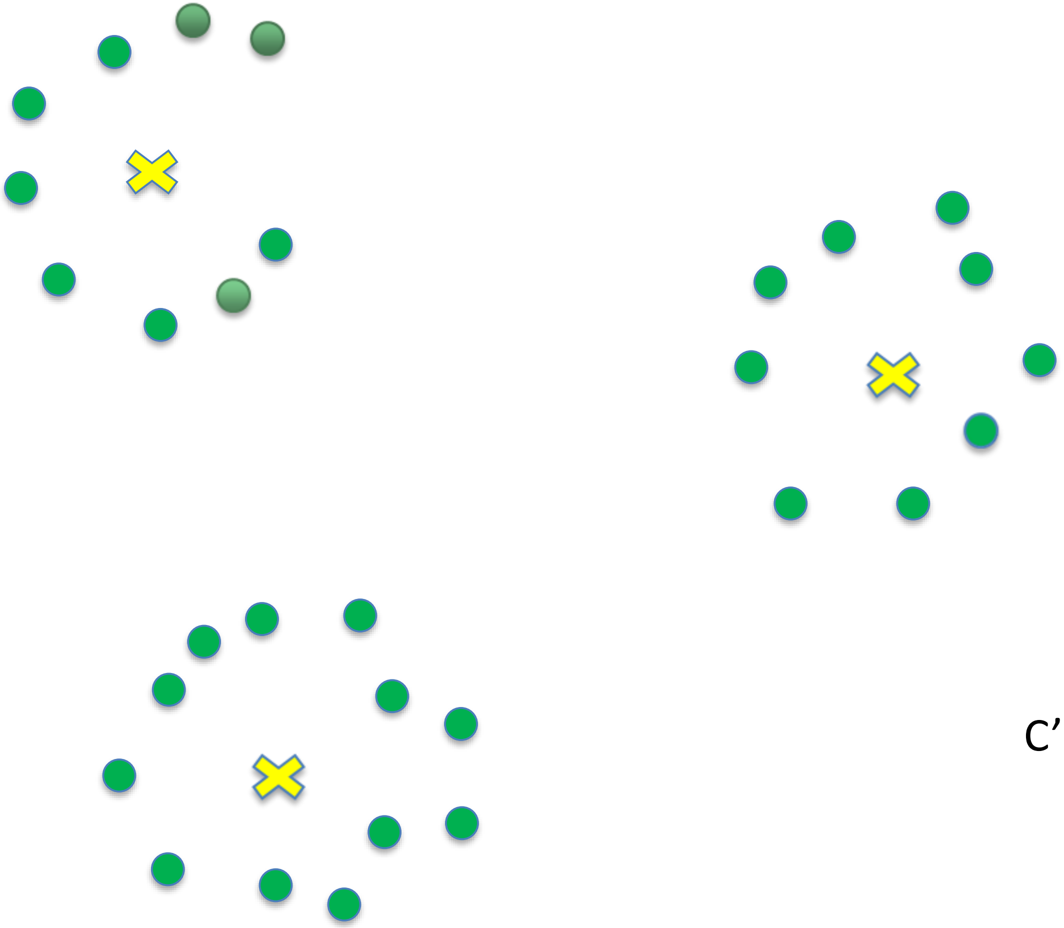
Original data P with centers X



C = Cost(P , X)

P’ = ϵ-coreset

ϵ-coreset P’ with centers X

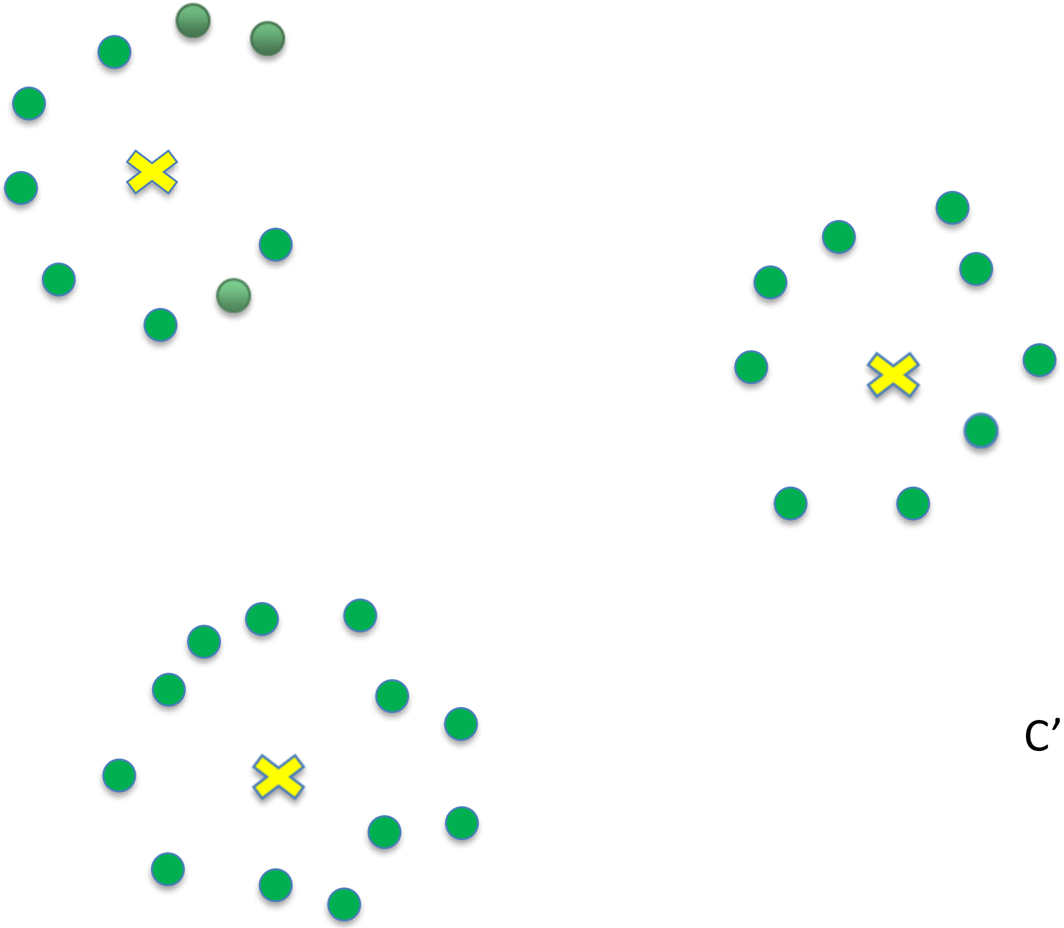


C = Cost(P , X)

P’ = ϵ-coreset

C’ = $\sum w(p')$Cost(p’ , X)

ϵ-coreset P’ with centers X



C = Cost(P , X)

P’ = ϵ-coreset

C’ = $\sum w(p')$Cost(p’ , X)

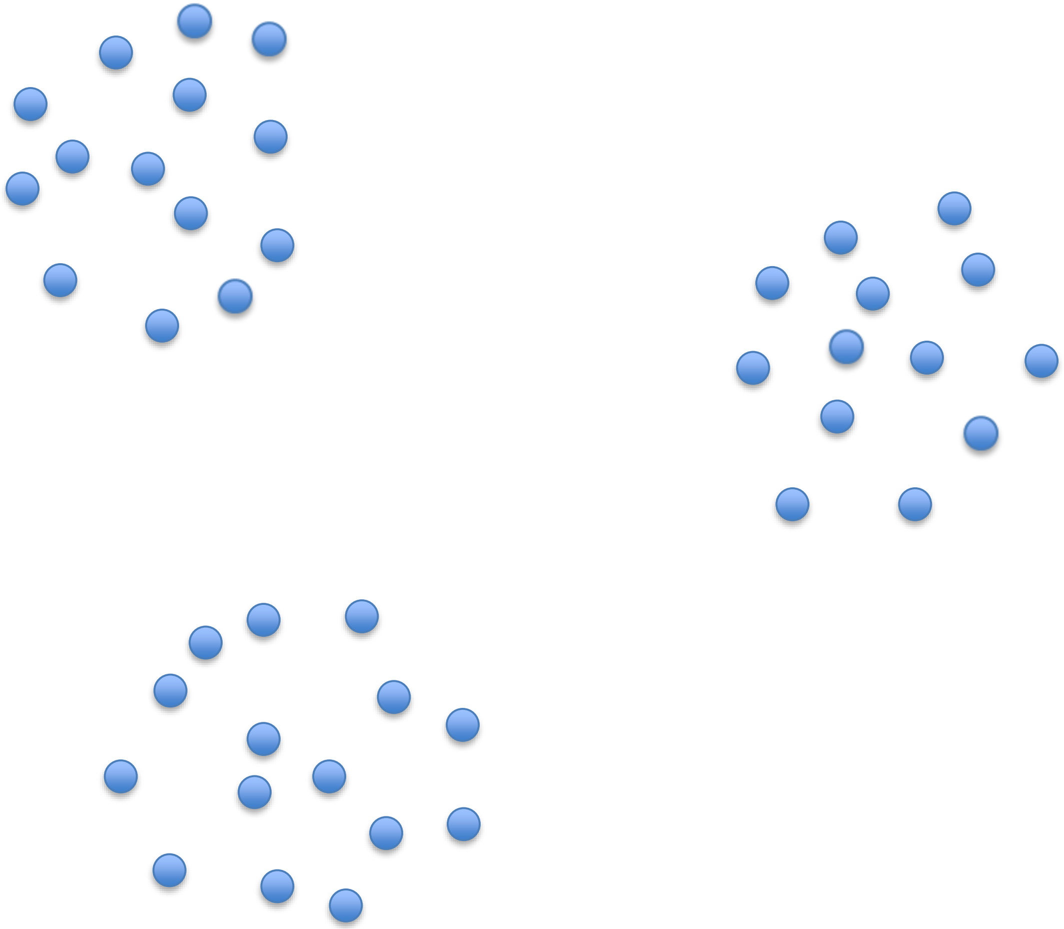
$(1-\epsilon)$Cost(P , X) $\leq \sum w(p')Cost(p',X) \leq (1+\epsilon)$Cost(P , X)

ϵ-coreset P’ with centers X

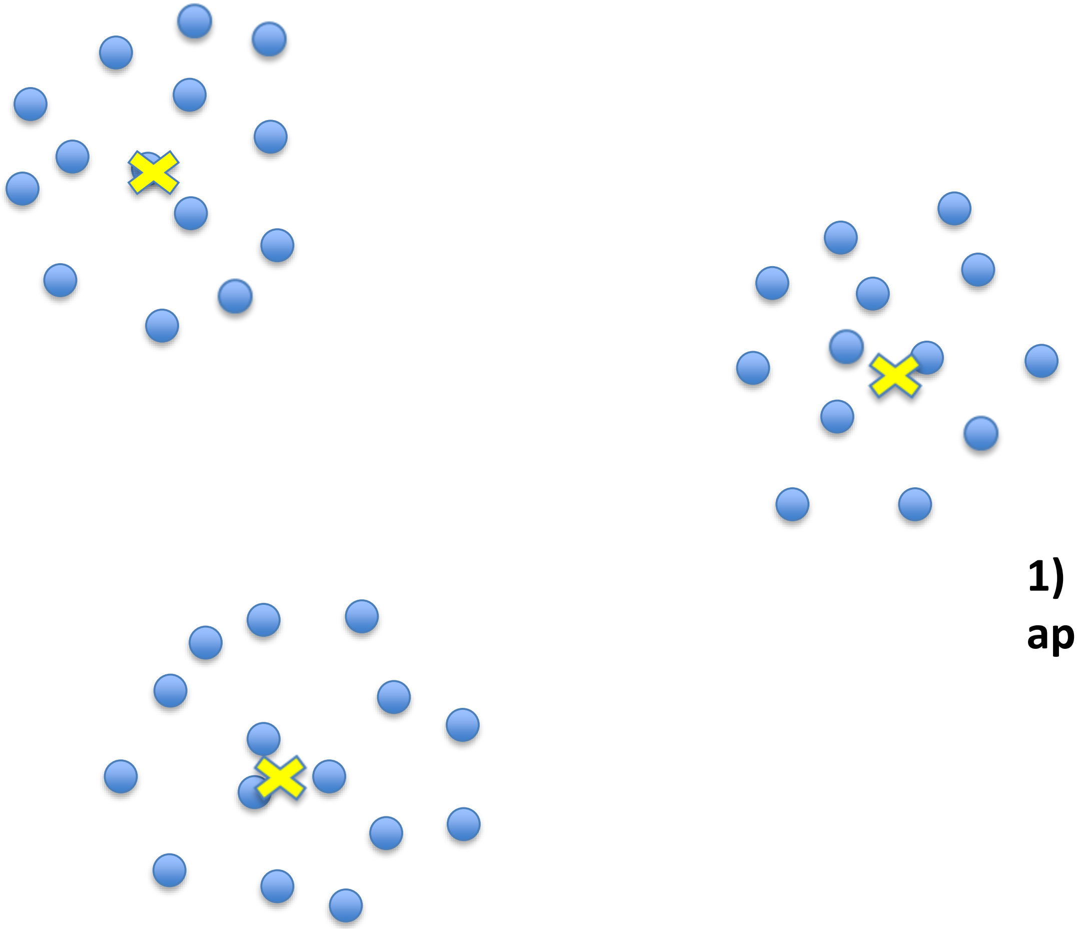
What is a coreset/ ϵ-coreset?

- An ϵ-coreset is a weighted set of points whose cost on any set of centers is approximately the cost of the original data on those same centers up to accuracy ϵ.
- Thus an approximate solution for the coreset is also an approximate solution for the original data.

How are coresets constructed?

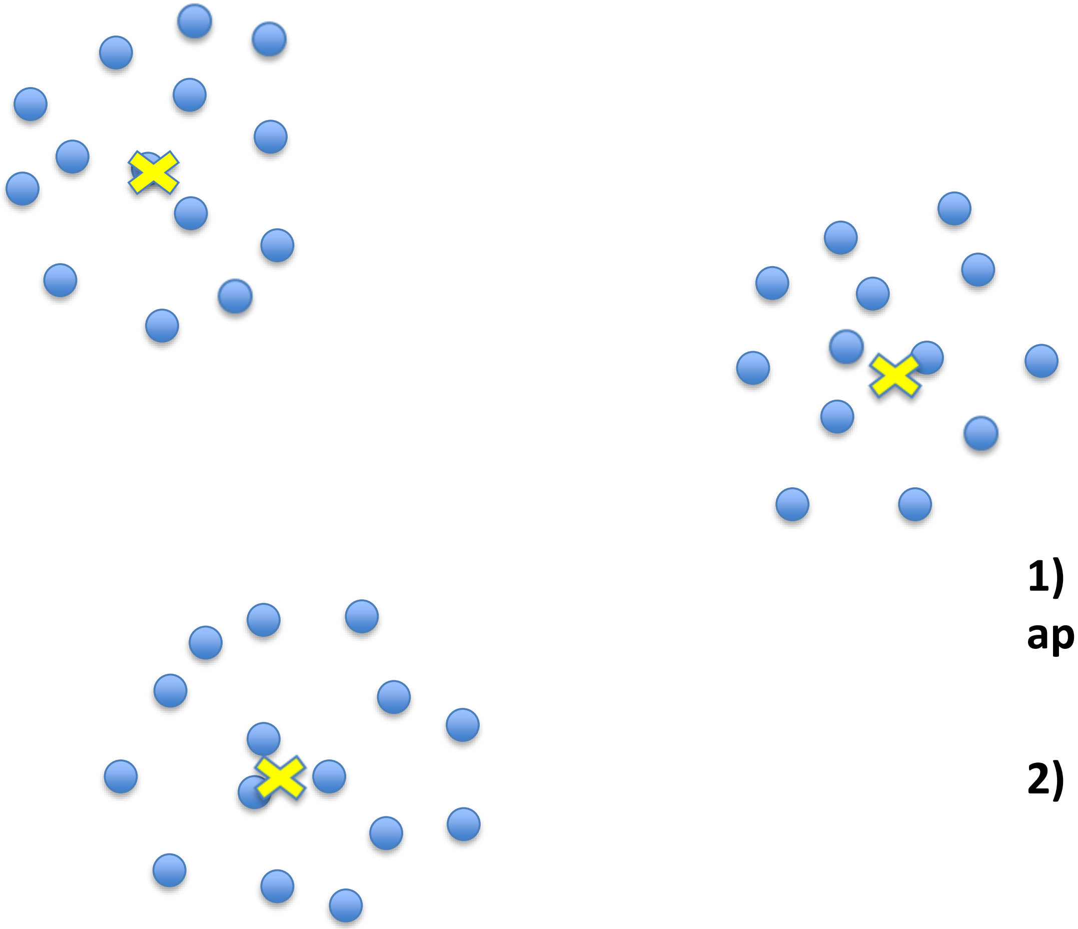


Original data P



1) Compute constant approximate solution

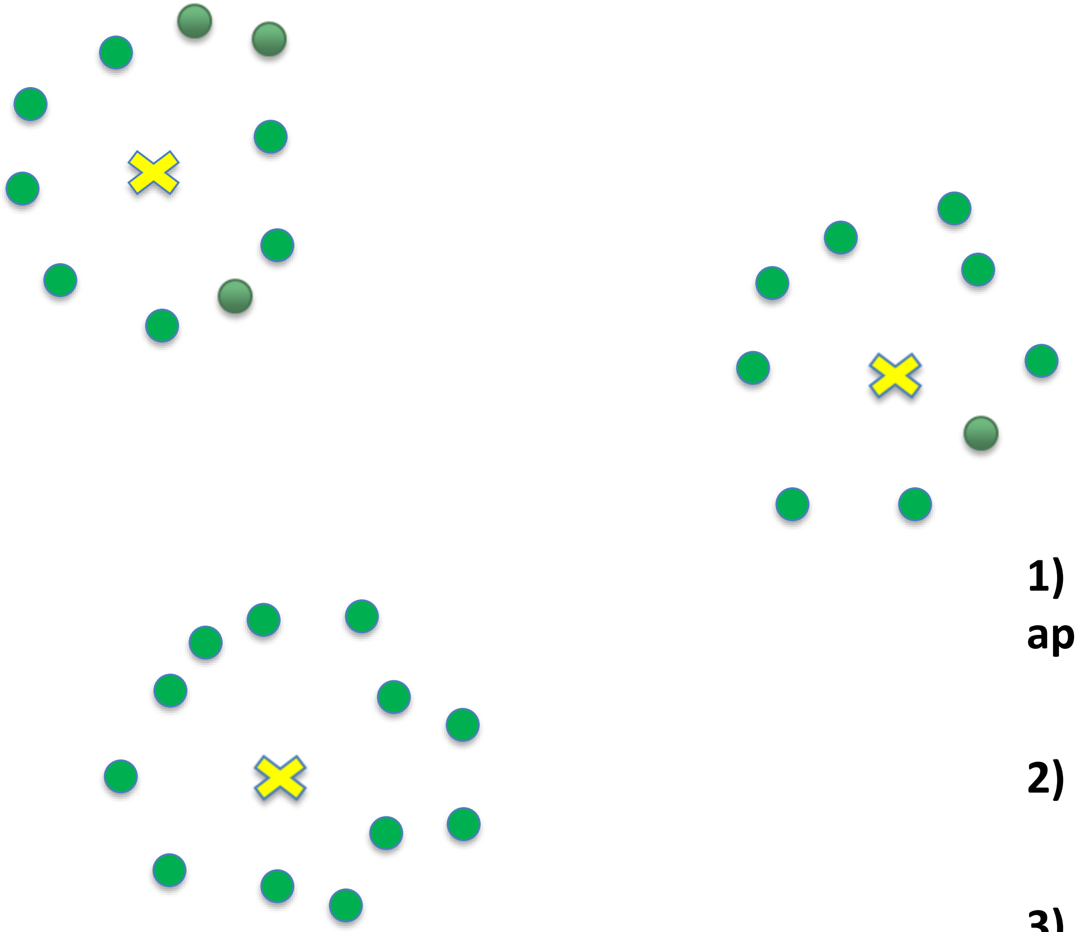
Original data P with centers X



1) Compute constant approximate solution

2) Calculate cost of solution

Original data P with centers X



Coreset P’ with centers X

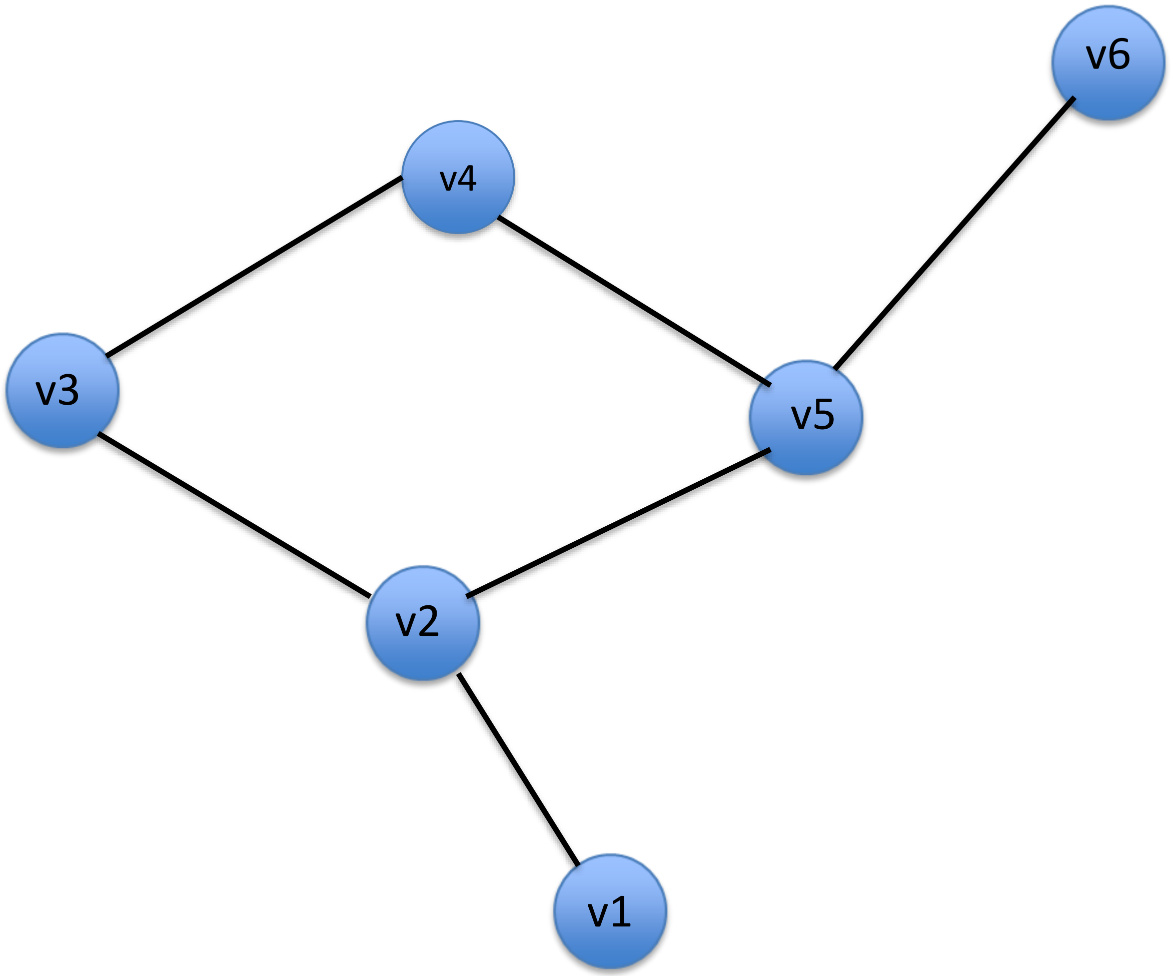
1) Compute constant approximate solution

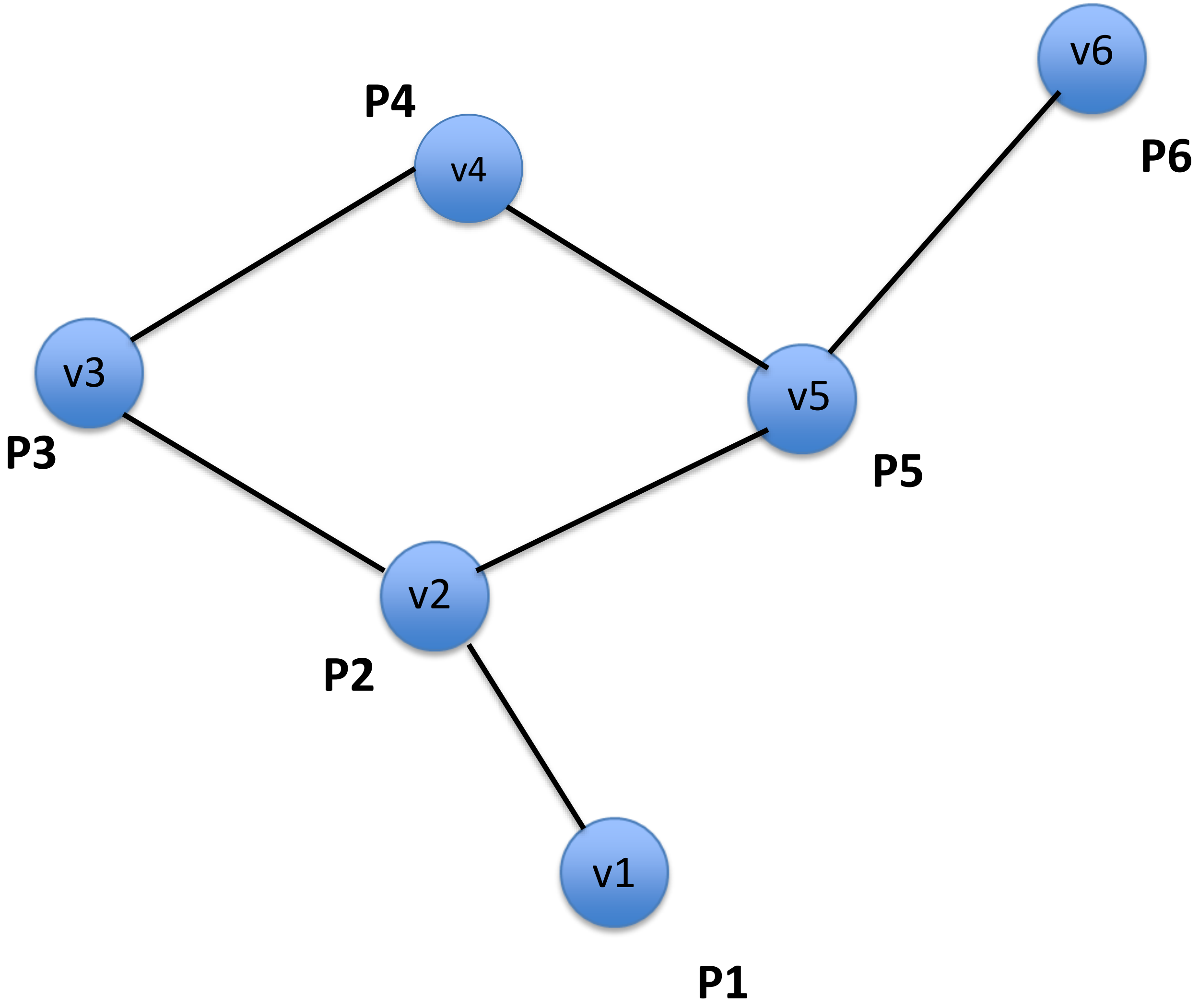
2) Calculate cost of solution

3) Sample points with probability proportional to their contributions to the cost.

Methodology

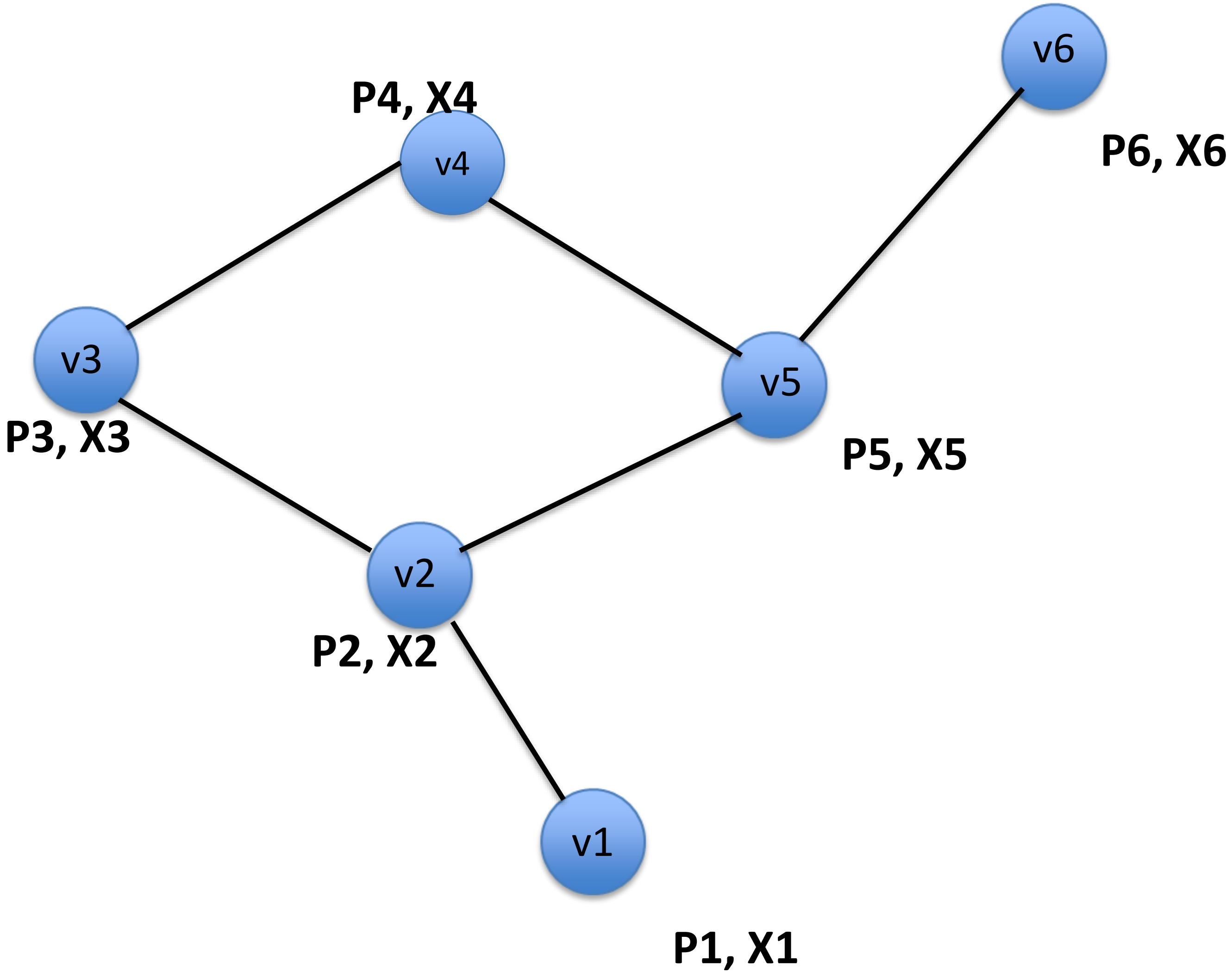
- The dataset is distributed on several nodes in an undirected connected graph.
- A local constant approximation solution is computed
- The total cost of local solution communicated to the other nodes in the graph using a message passing algorithm.
- Based on the cost of nodes and the contribution of the data towards their local solution, the local data are sampled to form local coresets


v6
v4
v3
v5
v2
v1


P4
v4
v6
P6
v3
v5
P3
P5
v2
P2
v1
P1

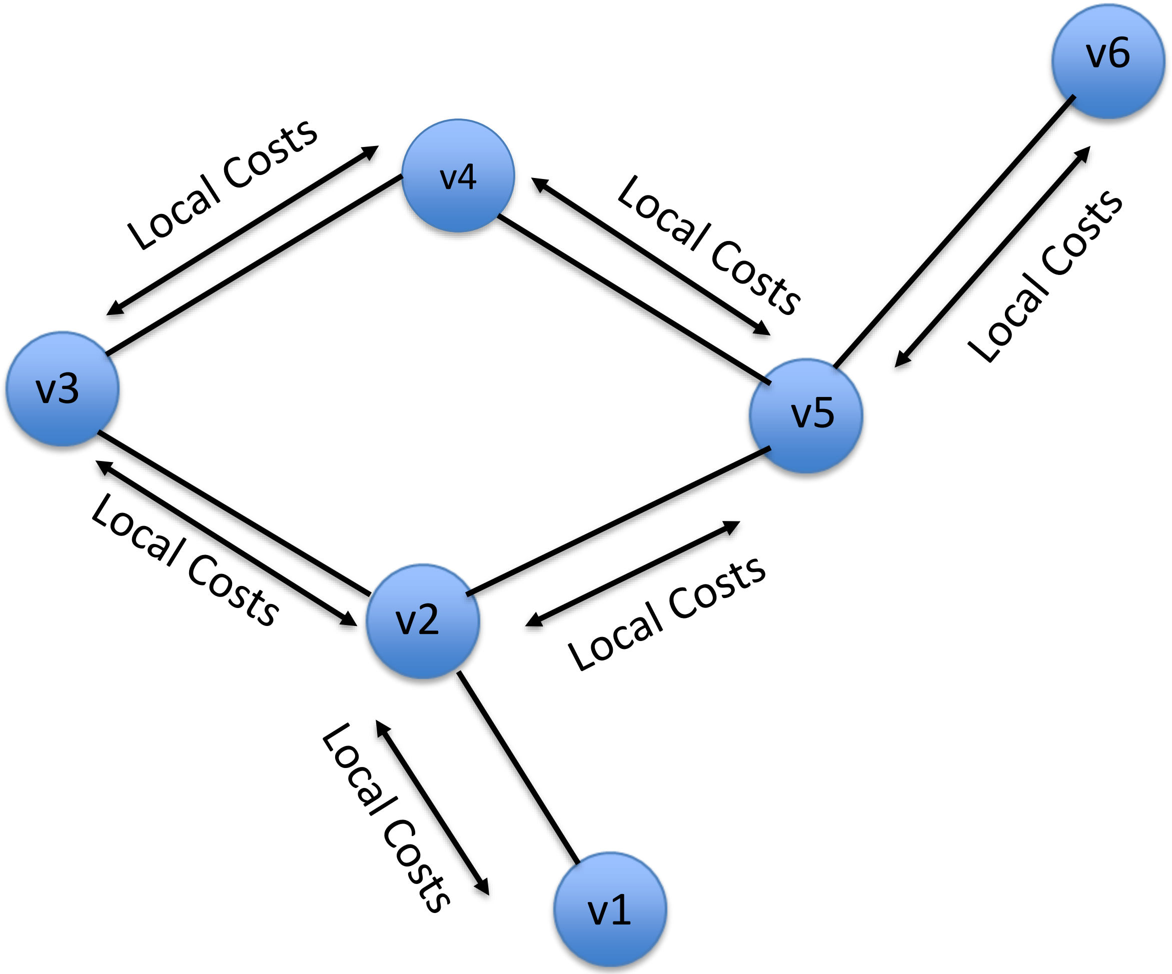
Methodology

- The dataset is distributed on several nodes in an undirected connected graph.
- A local constant approximation solution is computed
- The total cost of local solution communicated to the other nodes in the graph using a message passing algorithm.
- Based on the cost of nodes and the contribution of the data towards their local solution, the local data are sampled to form local coresets


P4, X4
v4
v6
P6, X6
v3
v5
P3, X3
P5, X5
v2
P2, X2
v1
P1, X1

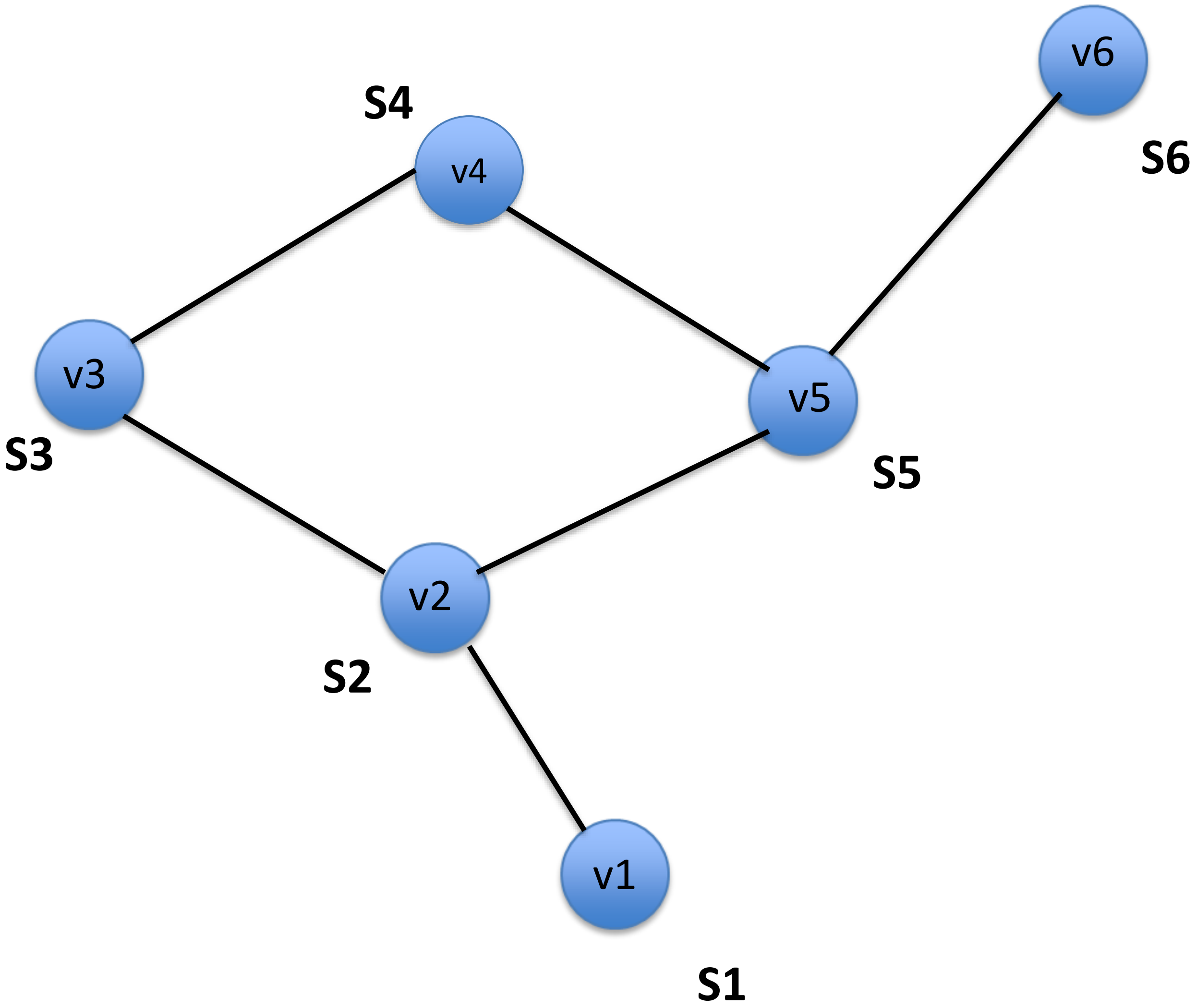
Methodology

- The dataset is distributed on several nodes in an undirected connected graph.
- A local constant approximation solution is computed
- The total cost of local solution communicated to the other nodes in the graph using a message passing algorithm.
- Based on the cost of nodes and the contribution of the data towards their local solution, the local data are sampled to form local coresets

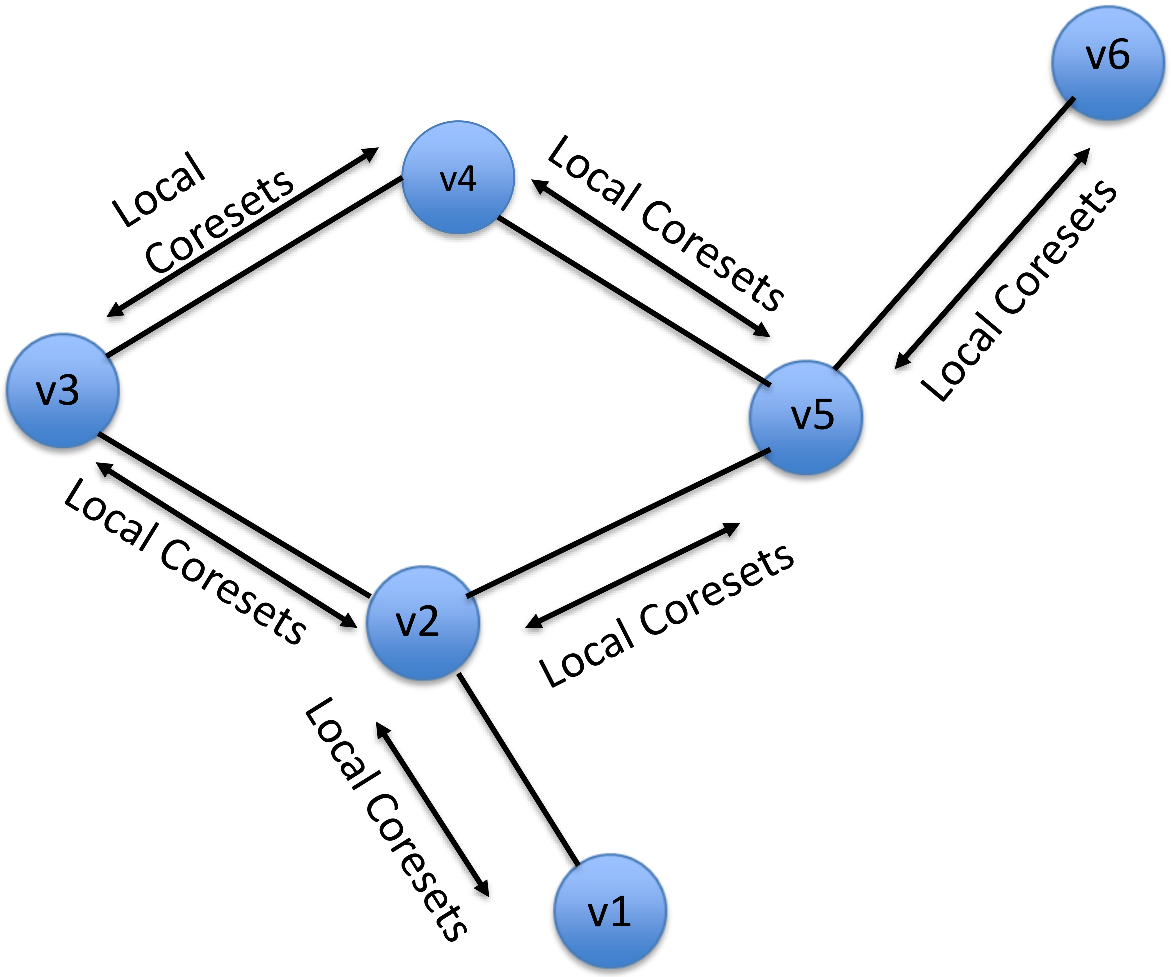

v6
v4
Local Costs
Local Costs
Local Costs
v3
v5
Local Costs
v2
Local Costs
Local Costs
v1

Methodology

- The dataset is distributed on several nodes in an undirected connected graph.
- A local constant approximation solution is computed
- The total cost of local solution communicated to the other nodes in the graph using a message passing algorithm.
- Based on the cost of nodes and the contribution of the data towards their local solution, the local data are sampled to form local coresets


v6
S4
v4
S6
v3
v5
S3
S5
v2
S2
v1
S1

- The message passing algorithm is called once again to combine the local coresets.
- If each node constructs an ϵ-coreset on its local data, then the union of these local coresets is an ϵ-coreset for the entire data.
- The constant approximation algorithm is applied to this global coreset to obtain the final result.

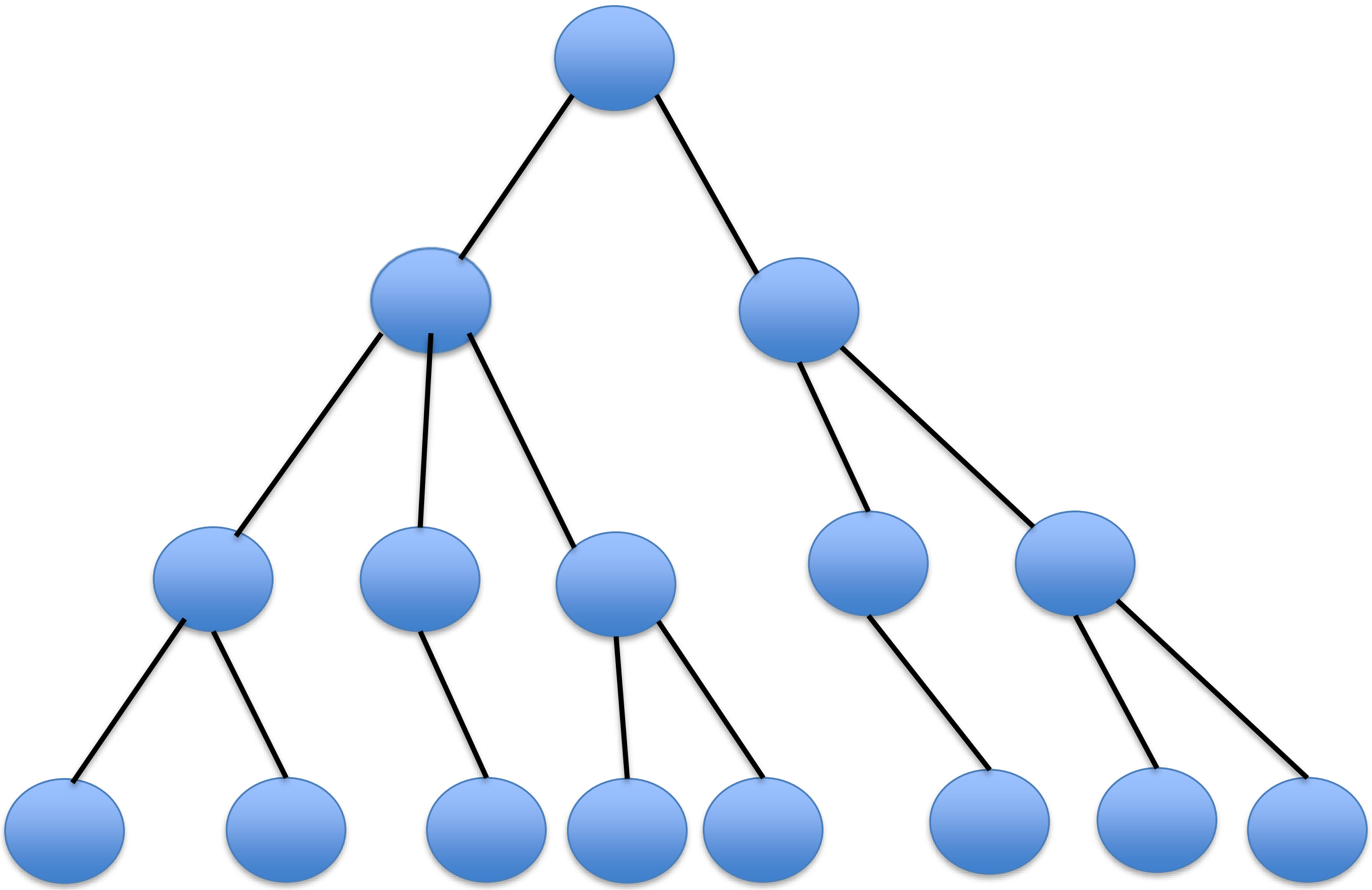

v6
v4
Local Coresets
Local Coresets
Local Coresets
v3
v5
Local Coresets
v2
Local Coresets
Local Coresets
v1

- The message passing algorithm is called once again to combine the local coresets.
- If each node constructs an ϵ-coreset on its local data, then the union of these local coresets is an ϵ-coreset for the entire data.
- The constant approximation algorithm is applied to this global coreset to obtain the final result.

- The message passing algorithm is called once again to combine the local coresets.
- If each node constructs an ϵ-coreset on its local data, then the union of these local coresets is an ϵ-coreset for the entire data.
- The constant approximation algorithm is applied to this global coreset to obtain the final result.

Comparison to Other Coreset Algorithms:

- Zhang et. al.[1] is limited to rooted trees.
  - The communication cost to develop coresets is very high.
  - Although it is possible to find a spanning tree, when the graph has large diameter every tree has large height which greatly increases the size of coresets which must be communicated across the lower levels of the tree.
- D. Feldman et. al.[2] ignores the communication cost and merges coresets in a parallel computation model.



Comparison to Other Coreset Algorithms:

- Zhang et. al.[1] is limited to rooted trees.
  - The communication cost to develop coresets is very high.
  - Although it is possible to find a spanning tree, when the graph has large diameter every tree has large height which greatly increases the size of coresets which must be communicated across the lower levels of the tree.
- D. Feldman et. al.[2] ignores the communication cost and merges coresets in a parallel computation model.

Technical Background

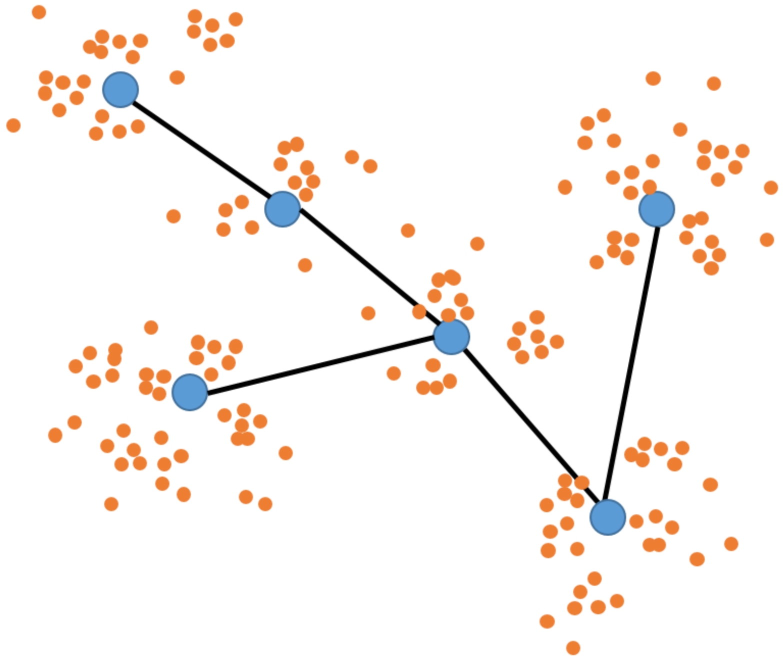
Preliminaries

- Consider two points $p_, q \in R^d$; $d(p, q)$ is the Euclidean distance between p and q
- k-means goal: Find set of k centers $X = \{x_1, x_2, \dots, x_k\}$ which minimize the k-means cost of data $P \subseteq R^d$
- k-means cost = $Cost(P, X) = \sum_{p \in P} d(p, X)^2$
  where $d(p, X) = \min_{x \in X} d(p, X)$
- If P is a weighted dataset with a weight function w, then k-means cost = $Cost(P, X) = \sum_{p \in P} w(p) d(p, X)^2$

- For distributed clustering, we consider
  Set of n nodes $V = \{v_i, 1 \leq i \leq n\}$ which communicate on undirected connected graph $G = (V, E)$, with $m = |E|$edges.
- Each node v_i contains local set of data points P_i and the global dataset is $\mathrm{P} = \bigcup_{i=1}^{n} P_{\;i}$
- $(v_i, v_j) \in E$ indicates that $v_i\ and\ v_j$ can communicate with each other.
- Communication cost = number of points transmitted. For simplicity, assume that there is no latency in communication.

Distributed coreset construction

Input: Data (distributed across different nodes)



- P_i : Set of points in node i
- Round 1 is performed on each node followed by Round 2.
- Only cost values are communicated.

Round 1

Compute centers for local data

Compute constant approximation

Communicate costs to nodes

Round 2

Compute number of points to be sampled

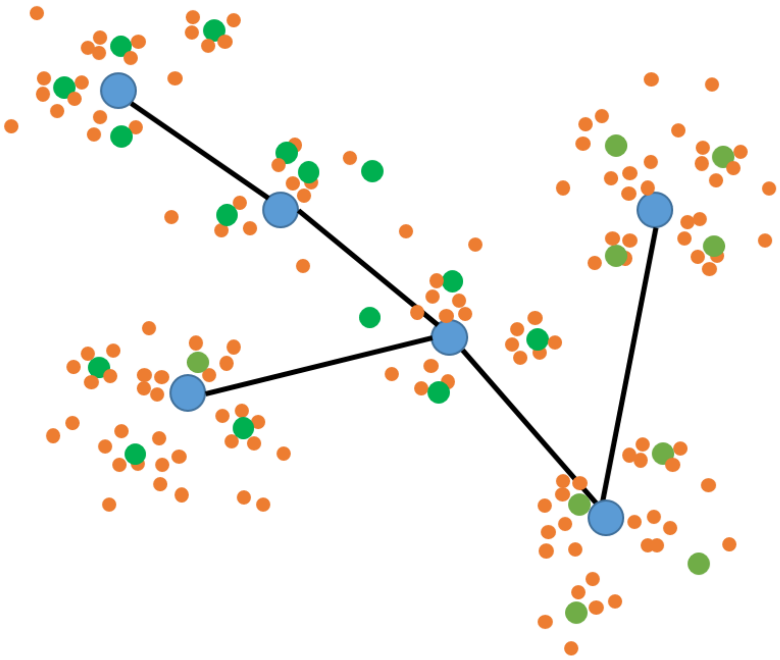
Set weights on local data

Random Sampling of points to coreset

Inclusion of local centers to coreset

Round 1

Step 1: Computing centers for local data



- Perform k-center clustering on node data (Lloyd's Algorithm)
- Output: Set of centers B_i for P_i

Round 1

Compute centers for local data

Compute constant approximation

Communicate costs to nodes

Round 2

Compute number of points to be sampled

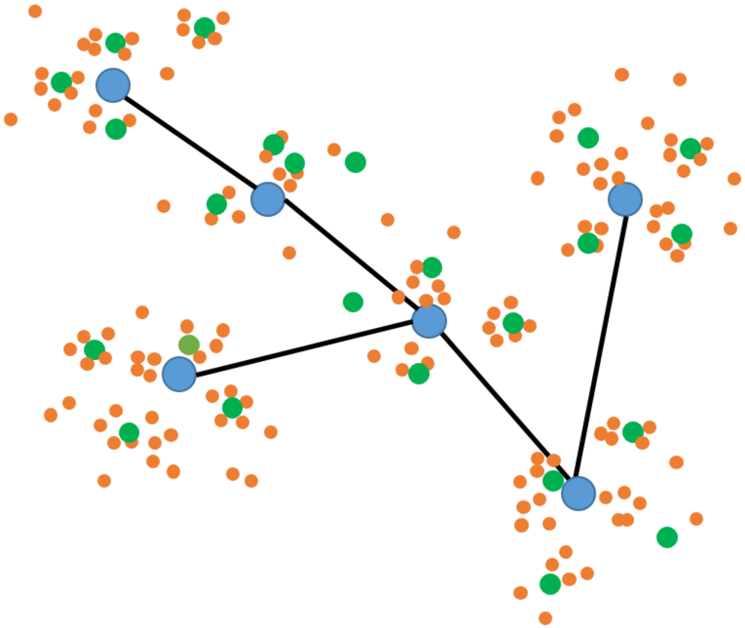
Set weights on local data

Random Sampling of points to coreset

Inclusion of local centers to coreset

Round 1

Step 2: Computing constant approximation



$$Cost(P_i, B_i) = \begin{cases} \sum_{p \in P_i} d(p, B_i) \;\; for\ k - medians \\ \sum_{p \in P_i} d(p, B_i)^2 \;\; for\ k - means \end{cases}$$

where $d(p, B_i) = \min_{b \in B_i} d(p, b)$

Round 1

Compute centers for local data

Compute constant approximation

Communicate costs to nodes

Round 2

Compute number of points to be sampled

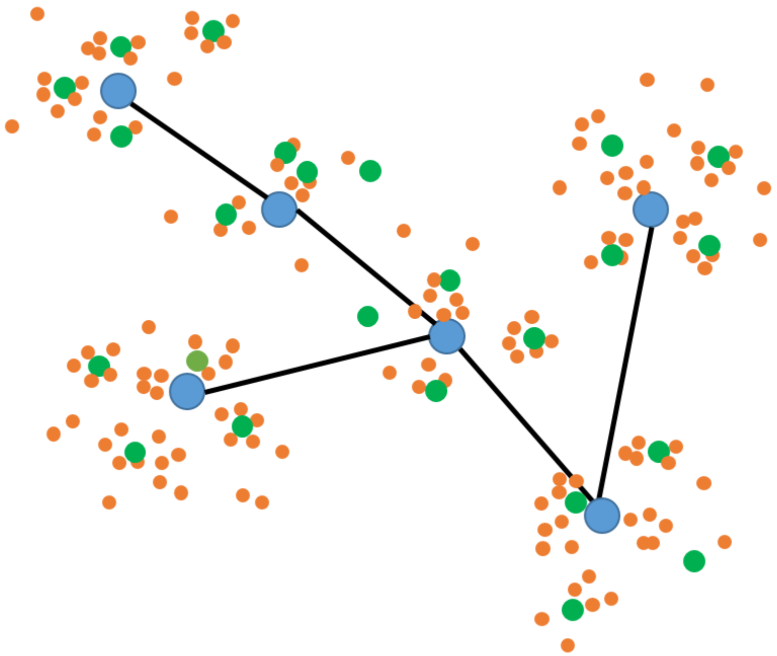
Set weights on local data

Random Sampling of points to coreset

Inclusion of local centers to coreset

Round 1

Step 3: Communicating cost to nodes



- $Cost(P_i, B_i)$ is communicated to each node.
- *Message_Passing()* used for cost communication

Round 1

Compute centers for local data

Compute constant approximation

Communicate costs to nodes

Round 2

Compute number of points to be sampled

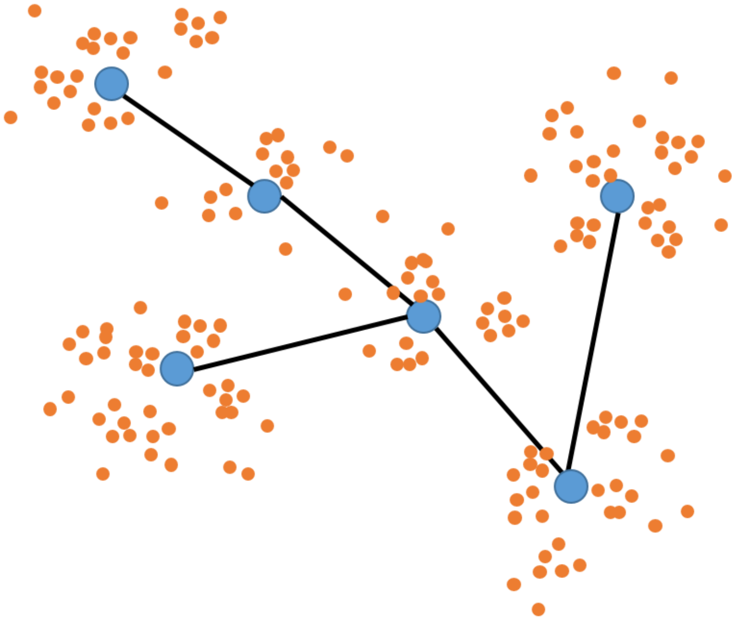
Set weights on local data

Random Sampling of points to coreset

Inclusion of local centers to coreset

Round 2

Step 1: Computing # of points to be sampled



- t = Global coreset size
- Number of points to be sampled on node i, $t_i = \frac{t\ cost(P_i,B_i)}{\sum_{j=1}^{n} cost(P_j,B_j)}$
- Proportional to total cost contribution

Round 1

Compute centers for local data

Compute constant approximation

Communicate costs to nodes

Round 2

Compute number of points to be sampled

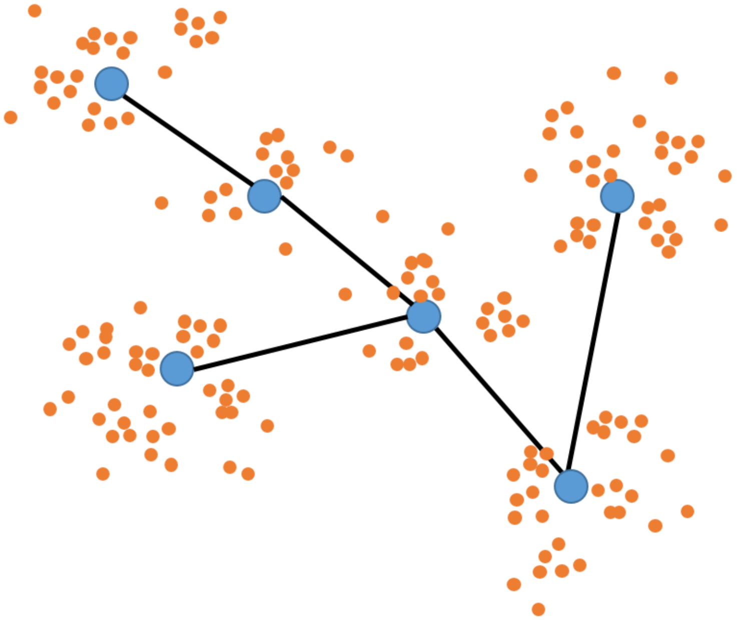
Set weights on local data

Random Sampling of points to coreset

Inclusion of local centers to coreset

Round 2

Step 2: Setting of weights on local data



- Weight of data point p, $m_p = 2cost(p, B_i)\ \forall\, p \in P_i$
- Proportional to distance of point from corresponding local center

Round 1

Compute centers for local data

Compute constant approximation

Communicate costs to nodes

Round 2

Compute number of points to be sampled

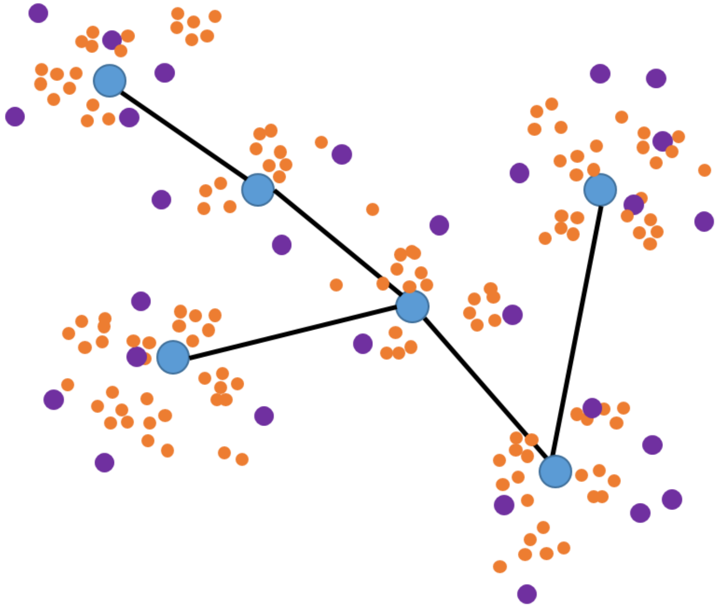
Set weights on local data

Random Sampling of points to coreset

Inclusion of local centers to coreset

Round 2

Step 3: Random sampling of points to coreset



- Non-uniform random sample S_i of t_i points from P_i, where for every $q \in Pi$, we have

$$\Pr[\mathrm{q} = \mathrm{p}] = \frac{m_p}{\sum_{z \in P_i} m_z}$$

- Weight on q, $w_q = \frac{\sum_i \sum_{z \in P_i} m_z}{m_q\, t}$ each $q \in Si$

Round 1

Compute centers for local data

Compute constant approximation

Communicate costs to nodes

Round 2

Compute number of points to be sampled

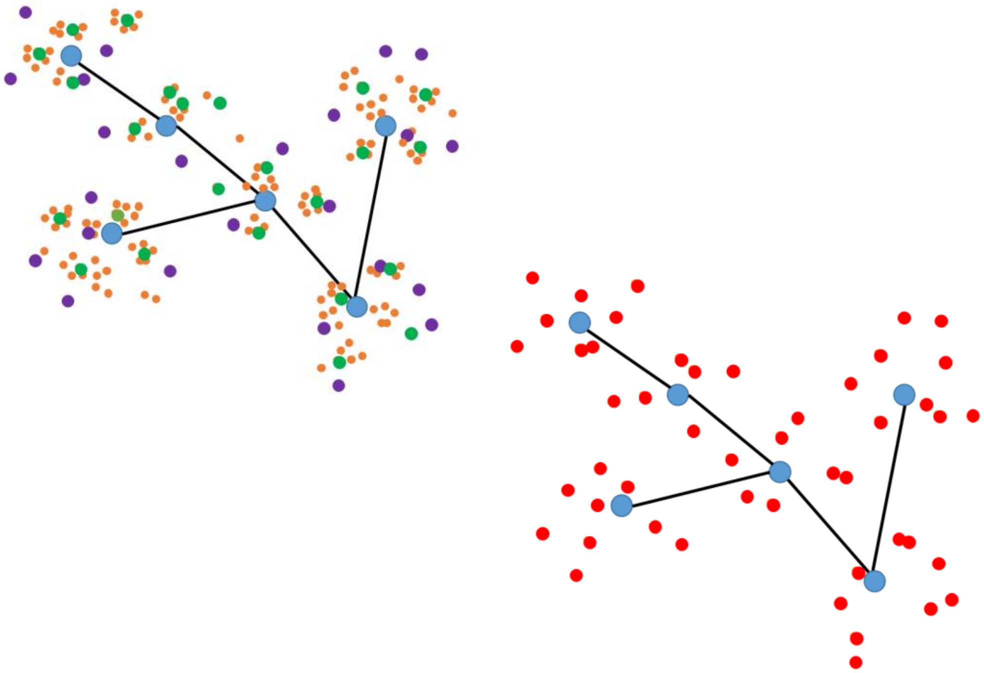
Set weights on local data

Random Sampling of points to coreset

Inclusion of local centers to coreset

Round 2

Step 4: Including local centers to coreset



- Local k-centers (B_i) are included to the coreset
- Weight on b, $w_b = |P_b| - \sum_{k \in P_b \cap S} w_k$, where $P_b = \{ p \in P_i \colon d(p, b) = d(p, B_i) \}$

Round 1

Compute centers for local data

Compute constant approximation

Communicate costs to nodes

Round 2

Compute number of points to be sampled

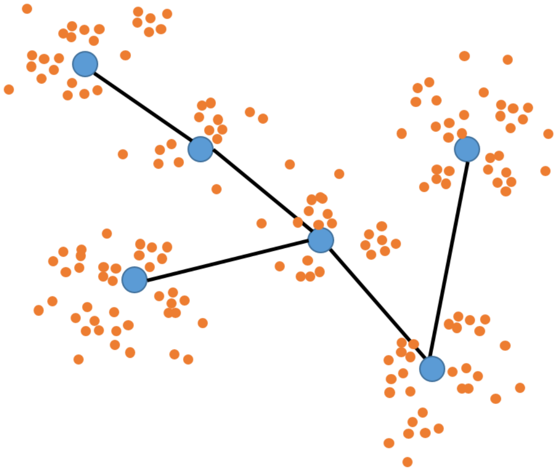
Set weights on local data

Random Sampling of points to coreset

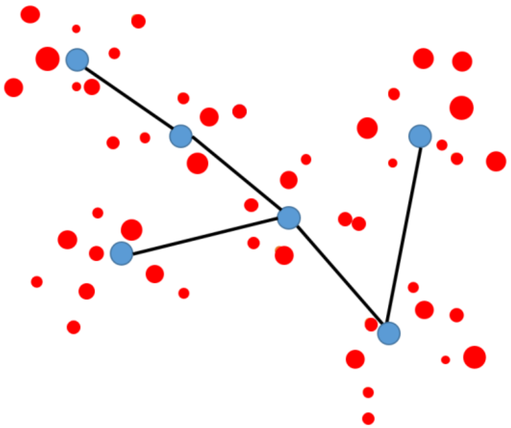
Inclusion of local centers to coreset

Distributed coreset construction

Input: Data (distributed across different nodes)



Output: Weighted Coreset



Round 1

Compute centers for local data

Compute constant approximation

Communicate costs to nodes

Round 2

Compute number of points to be sampled

Set weights on local data

Random Sampling of points to coreset

Inclusion of local centers to coreset

Objective:

- Reduce the total communication cost
- Preserve the theoretical guarantees for approximate clustering cost

Theorem 1:

There exists an algorithm that with probability at least $1 - \delta$, the output is an ϵ-coreset for the global dataset. The communication cost is $O(mn)$ and the number of points in the coresets (*coreset size*) are:

- **k- means:** $O\left(\frac{1}{\epsilon^4}\left(kd + log\left(\frac{1}{\delta}\right)\right) + nk\ log\left(\frac{nk}{\delta}\right)\right)$
- **k-medians:** $O\left(\frac{1}{\epsilon^2}\left(kd + log\left(\frac{1}{\delta}\right)\right) + nk\right)$

Key Parameters

- **Sampling Probability:** Probability with which data points are chosen to be candidates for coresets

$$\Pr[q = p] = \frac{m_p}{\sum_{z \in P} m_z}$$

- **Weights on coreset points:** The weights on coreset points are:

$$w_q = \frac{\sum_i \sum_{z \in P_i} m_z}{m_q\,|S|} \text{ for q} \in S \text{ and } w_b = |P_b| - \sum_{k \in P_b \cap S} w_k \text{ for } b \in \text{B}$$

- **Function applied on the points**: A suitably defined function *f*(*p*) is selected. The function nature varies for k-means and k-medians.

Validity of Sampling probability and Weights

Claim: For a given function *f*(*p*), the sampled coreset points are valid approximations of the original data

To Prove: $E\left[\sum_{q\in S} w_q f(q)\right] = \sum_{p\in P} f(p)$ for chosen probabilities $\Pr[q = p]$ and weights w_p

Proof:

$$E\left[\sum_{q\in S} w_q f(q)\right] = \sum_{q\in S} E\left[w_q f(q)\right] = \sum_{q\in S}\sum_{p\in P} \Pr[q = p] w_p f(p)$$

$$= \sum_{q\in S} \ \sum_{p\in P} \frac{m_p}{\sum_{z\in P} m_z} \frac{\sum_{z\in P} m_z}{m_p|S|} \ f(p)$$

$$= \sum_{q\in S} \ \sum_{p\in P} \frac{1}{|S|} \ f(p) = \sum_{p\in P} f(p)$$

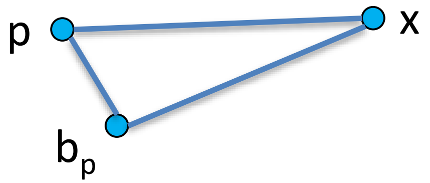
Validity of *f(p)*

Lemma 1:

Fix a set F of functions *f: P* → R+. Let S be an i.i.d. sample drawn from P according to $\{m_p : p \in P\}$. For $p \in P$ and q $\in S$, we have $q = p$ with probability $\frac{m_p}{\sum_{z \in P} m_z}$. Also, define $w_p = \frac{\sum_{z \in P} m_z}{m_p |S|}$. If for a sufficiently large c, we have $|S| \geq \frac{c}{\epsilon^2}(\dim(F, P) + \log \frac{1}{\delta})$, then with a probability at least 1 - δ ,

$\forall\, f \in F$: $|\sum_{p \in P} f(p)$ - $\sum_{q \in S} w_q f(q)| \leq \epsilon\, (\sum_{p \in P} m_p)(max_{p \in P} \frac{f(p)}{m_p})$

- Set $m_p = max_{f \in F} f(p)$. Bounds RHS to $\epsilon\, (\sum_{p \in P} m_p)$
- Natural Approach: Set $f_x(p) \coloneqq cost(p, x)$
- Complication: Suitable upper bound not available for $cost(p, x)$
- Alternative Approach: Set $f_x(p) \coloneqq cost(p, x) - cost(b_p, x) + cost(p, b_p)$



$$0 \leq f_x(p) \leq 2cost(p, b_p)$$

Proof of Theorem 1 for k-medians

Outline: Given that the sampling probabilities, weights on points and $f(p)$ as defined previously, algorithm outputs an ϵ-coreset.

- Set $f_x(p) \coloneqq d(p,x) - d(b_p,x) + d(p,b_p)$. Then, $m_p = 2d(p,b_p)$ and the conditions stated in Lemma 1 are satisfied.
- Then,

$$D = \left|\sum_{p\in P} f_x(p) - \sum_{q\in S} w_q f_x(q)\right| \le O(\epsilon)\sum_{p\in P} d(p,x)$$

- Substituting the definition of $f_x(\circ)$ in the expression for D,

$$D = \left|\sum_{p\in P}\left[d(p,x) - d(b_p,x) + m_p\right] - \sum_{q\in S} w_q\left[d(p,x) - d(b_p,x) + m_q\right]\right|$$

- On further simplification

$$D = \left|\sum_{p\in P} d(p,x) - \sum_{q\in S\cup B} w_q d(q,x)\right| \le O(\epsilon)\sum_{p\in P} d(p,x)$$

which is guarantee that the output of algorithm 1 is an ϵ-coreset

Proof of Theorem 1 for k-means

- **Complication:**

  Difference term $\left|cost(p,x) - cost\left(b_p,x\right)\right| = \left|d(p,x)^2 - d\left(b_p,x\right)^2\right|$ is not bounded. Conditions of Lemma 1 are not met.

- **Fix:** Two categories of points:

❑ **Good points** (costs approximately satisfy the triangle inequality)

❑ **Bad points**

GOOD POINTS:

- Define the good points with centers x as
- $G(x) = \{p \in P : \left|cost(p,x) - cost\left(b_p,x\right)\right| \leq \Delta_p\}$

  where $\Delta_p = \frac{cost\left(b_p,x\right)}{\epsilon}$
- These difference between the costs can be bound as before.

BAD PONTS:

For bad points, it is proven that the difference in cost may be larger than $\frac{cost(b_p,x)}{\epsilon}$ but will be $O(\epsilon \min\{cost(p,x), cost(b_p,x)\})$.

- Hence $f_x(p)$ is defined only over good points as follows:

$$\boldsymbol{f_x(p) = \begin{cases} cost(p,x) - cost(b_p,x) + \Delta_p & if\ p \in G(x) \\ 0 & otherwise \end{cases}}$$

- Then, $D = \left|\sum_{p\in P} cost(p,x) - \sum_{q\in S\cup B} w_q cost(q,x)\right|$ can be decomposed into three terms

$$\sum_{p\in P} f_x(p) - \sum_{q\in S} w_q f_x(q) \quad (1)$$
$$+\sum_{p\in P\setminus \mathrm{G}(x)} [cost(p,x) - cost(b_p,x) + \Delta_p] \quad (2)$$
$$-\sum_{p\in P\setminus \mathrm{G}(x)} w_q [cost(q,x) - cost(b_q,x) + \Delta_q] \quad (3)$$

Proof of Theorem 1 for k-means (cont.)

- It can be showed that each of these terms is bounded by $\boldsymbol{O(\epsilon)cost(P,x)}$

- Sum of three terms is also bounded by $\boldsymbol{O(\epsilon)cost(P,x)}$

- Then,

$$D = \left|\sum_{p\in P} cost(p,x) - \sum_{q\in S\cup B} w_q cost(q,x)\right| \leq O(\epsilon)cost(P,x)$$

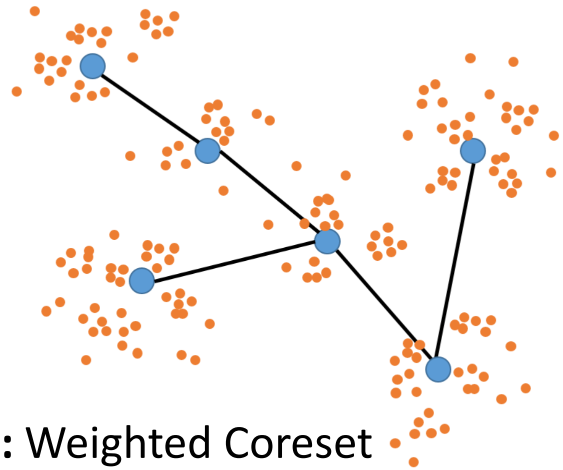
- Hence, by choosing an appropriate ϵ, the result is established.

Pseudo code

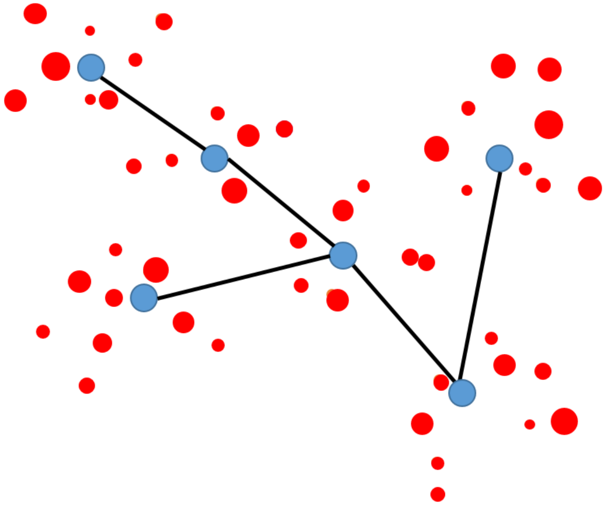
Git Link: https://github.com/harshadrai/Distributed_kMeans

Distributed coreset construction

Input: Data (distributed across different nodes)



Output: Weighted Coreset



Round 1

- Compute centers for local data
- Compute constant approximation
- Communicate costs to nodes

Round 2

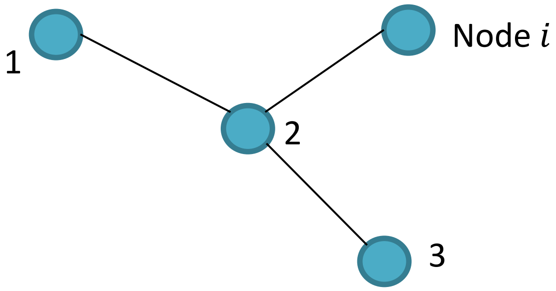
- Compute number of points to be sampled
- Set weights on local data
- Random Sampling of points to coreset
- Inclusion of local centers to coreset

Message Passing

Input:

Message to communicate

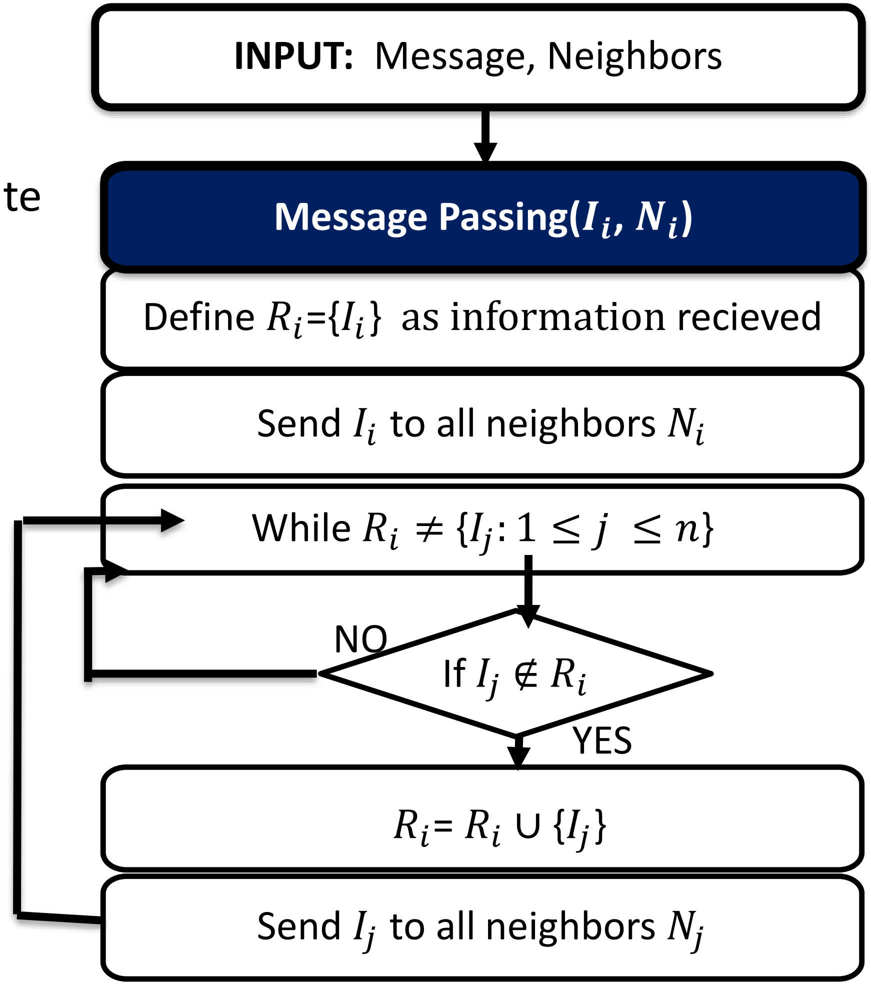
Neighbors to whom we wish to communicate the message



I_i - Message at node i

N_i - Neighbors of node i

$\implies$ N_i - Node 2



INPUT: Message, Neighbors

Message Passing(I_i, N_i)

Define R_i={I_i} as information recieved

Send I_i to all neighbors N_i

While $R_i \neq \{I_j: 1 \leq j \leq n\}$

If $I_j \notin R_i$

NO

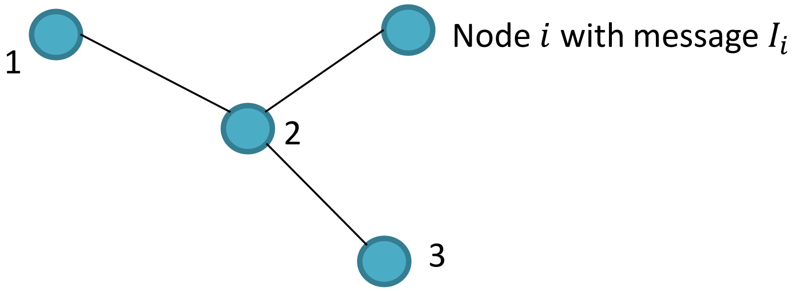
YES

$R_i = R_i \cup \{I_j\}$

Send I_j to all neighbors N_j

Step 1:

Define $R_i = \{I_i\}$ as the information received

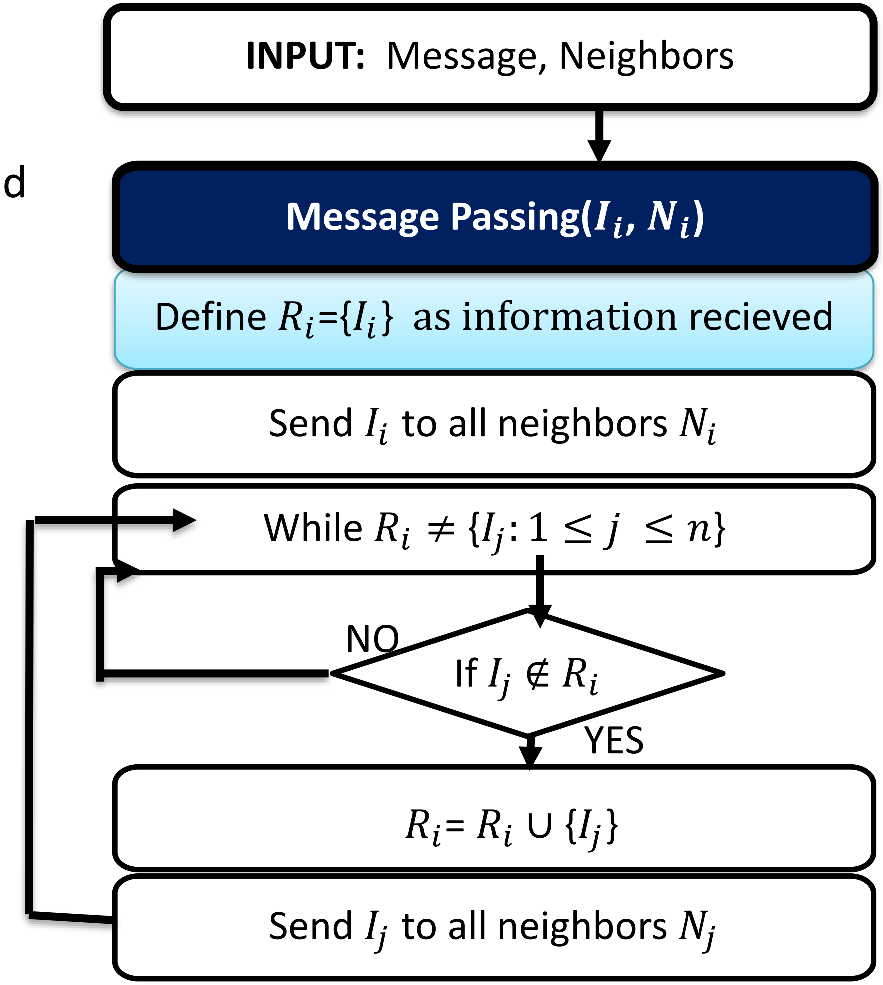


I_i - Message at node i

N_i - Neighbors of node i

$\Longrightarrow$ N_i - Node 2

$R_i = \{I_i\}$



INPUT: Message, Neighbors

Message Passing(I_i, N_i)

Define R_i=$\{I_i\}$ as information recieved

Send I_i to all neighbors N_i

While $R_i \neq \{I_j\!: 1 \leq j \leq n\}$

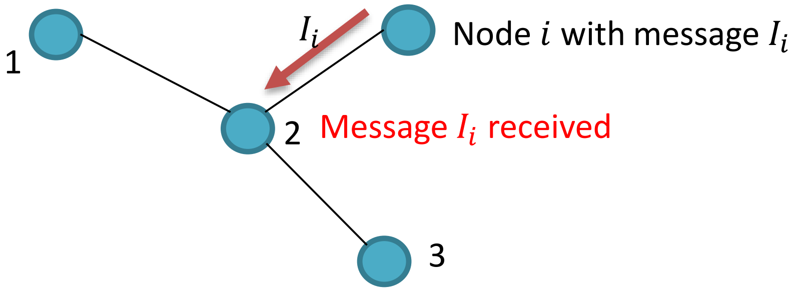
If $I_j \notin R_i$

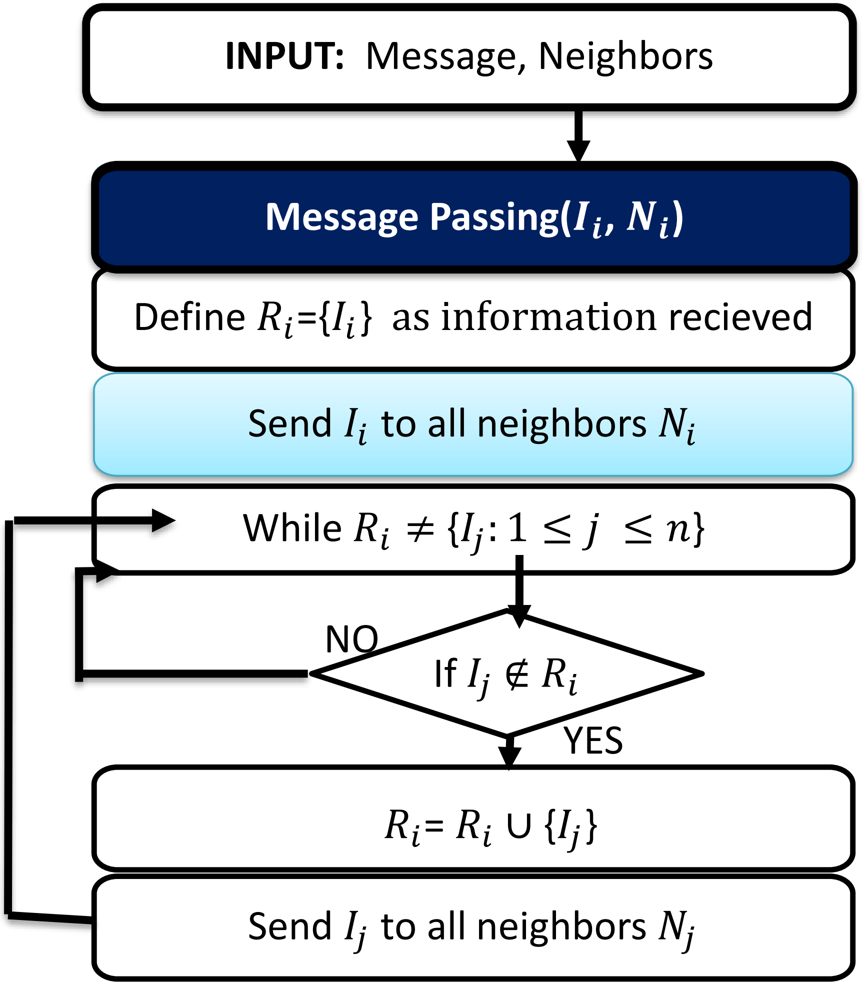
R_i= $R_i \cup \{I_j\}$

Send I_j to all neighbors N_j

Step 2:

Send I_i to all its' neighbors N_i





INPUT: Message, Neighbors

Message Passing(I_i, N_i)

Define R_i={I_i} as information recieved

Send I_i to all neighbors N_i

While $R_i \neq \{I_j: 1 \leq j \leq n\}$

If $I_j \notin R_i$

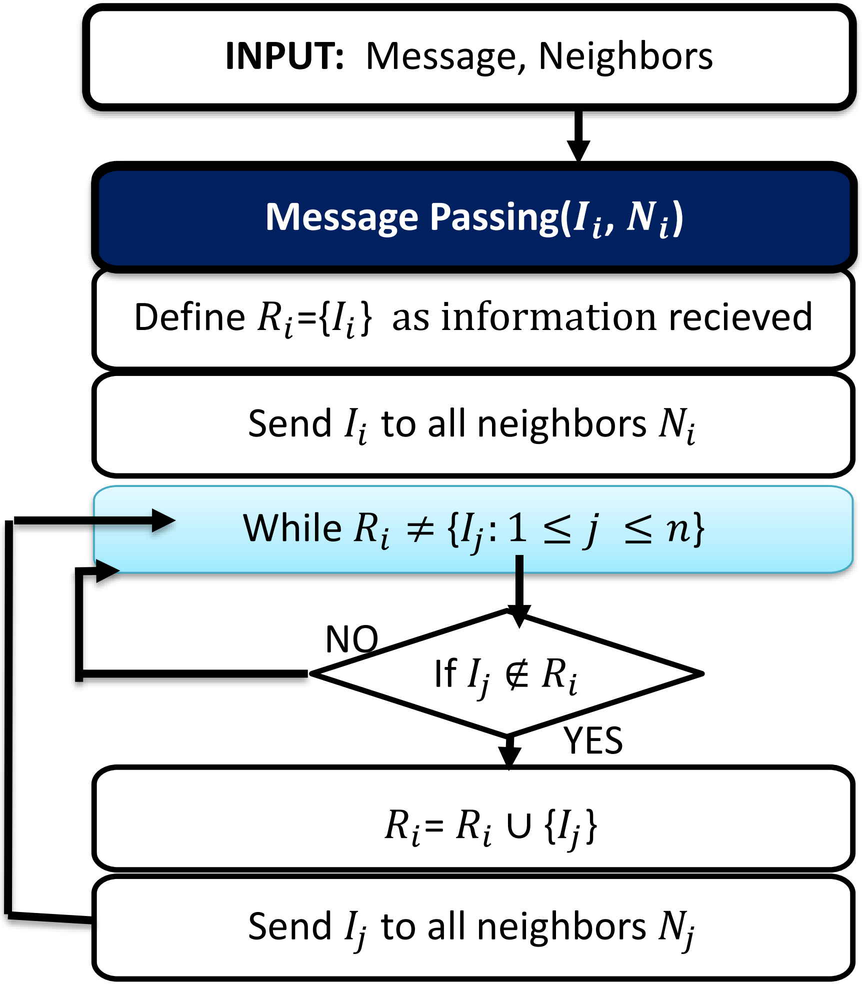
R_i= $R_i \cup \{I_j\}$

Send I_j to all neighbors N_j

Step 3:
Check if information received R_i is same as information present at all the nodes

YES: STOP

NO: Continue to step 4


INPUT: Message, Neighbors
Message Passing(I_i, N_i)
Define R_i={I_i} as information recieved
Send I_i to all neighbors N_i
While $R_i \neq \{I_j: 1 \leq j \leq n\}$
NO
If $I_j \notin R_i$
YES
$R_i = R_i \cup \{I_j\}$
Send I_j to all neighbors N_j

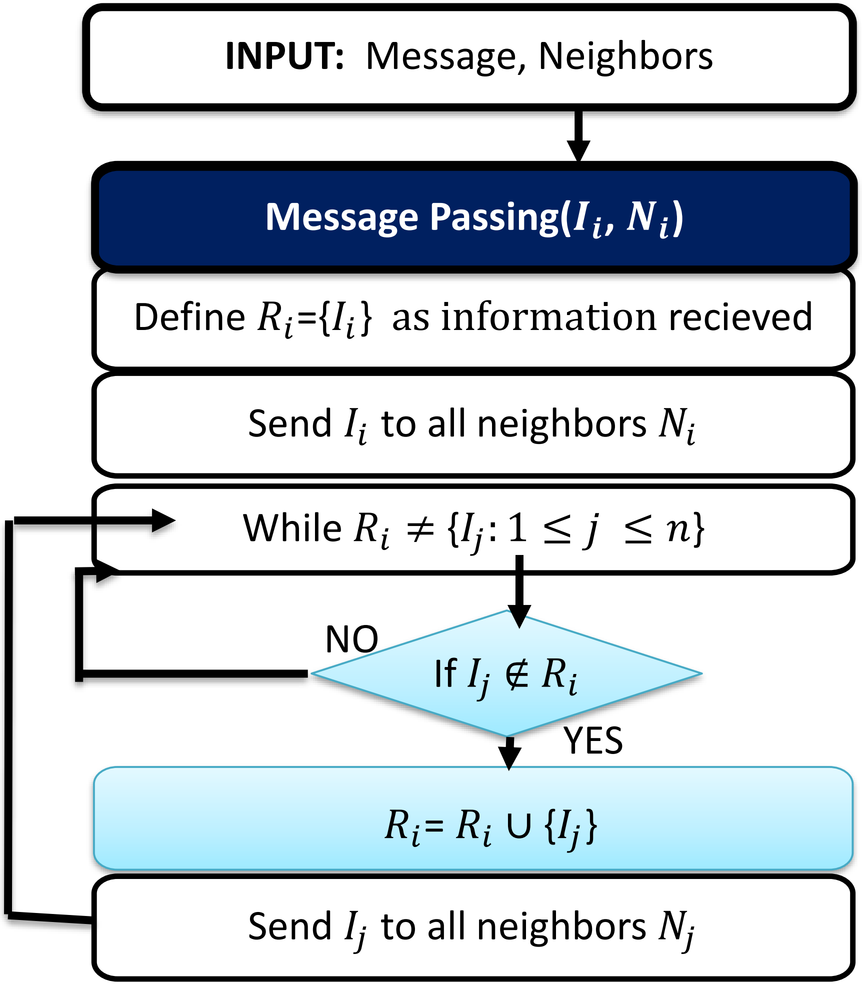
Step 4:

Check if information $I_j \notin R_i$

YES: Extend set of information received by I_j

$R_i = R_i \cup \{I_j\}$

NO: Continue to step 3



INPUT: Message, Neighbors

Message Passing(I_i, N_i)

Define R_i={I_i} as information recieved

Send I_i to all neighbors N_i

While $R_i \neq \{I_j : 1 \leq j \leq n\}$

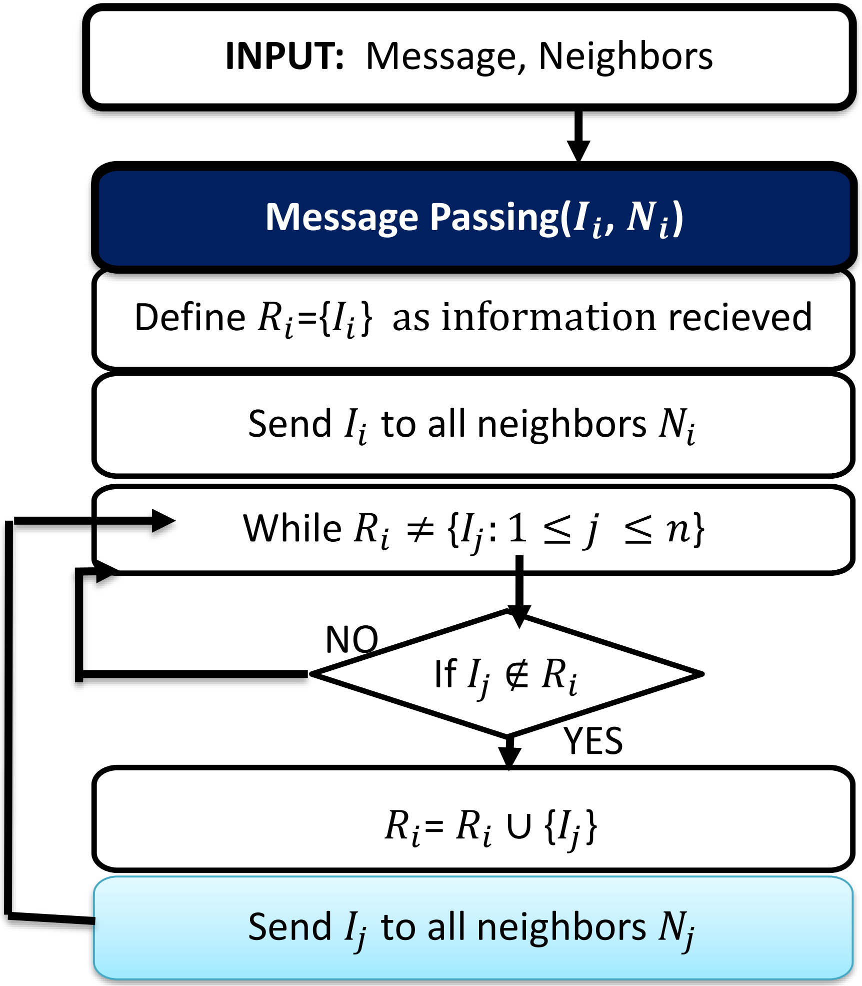
If $I_j \notin R_i$

$R_i = R_i \cup \{I_j\}$

Send I_j to all neighbors N_j

Step 5:

Send I_j to all its' neighbors N_j



INPUT: Message, Neighbors

Message Passing(I_i, N_i)

Define R_i={I_i} as information recieved

Send I_i to all neighbors N_i

While $R_i \neq \{I_j : 1 \leq j \leq n\}$

NO

If $I_j \notin R_i$

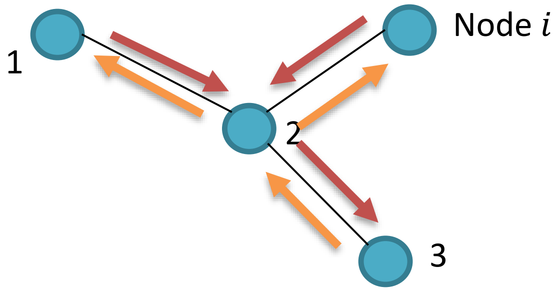
YES

$R_i = R_i \cup \{I_j\}$

Send I_j to all neighbors N_j

Output:

Communication done among all the nodes



- As a result, information is shared globally.

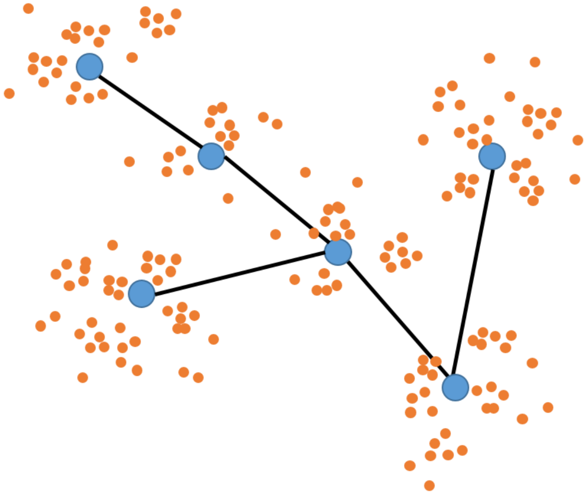
- Communication cost at each step = Size of the message

Distributed clustering on graph

Input:

Data (distributed among nodes)

A_α - α approximation clustering algorithm (**Lloyd's Algorithm**)



- P_i : Set of points in node i
- N_i: Neighbors at node i

Round 1

For each node i

Get ε local coreset D_i using Distributed Coreset construction algorithm

Round 2

For each node i

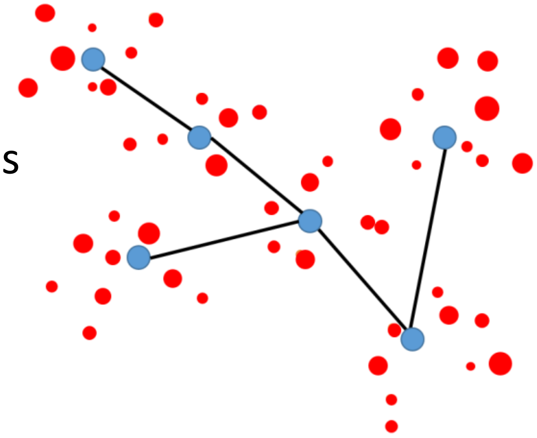
Call Message_Parsing (D_i, N_i)

X = A_α ($\cup$ D_j)

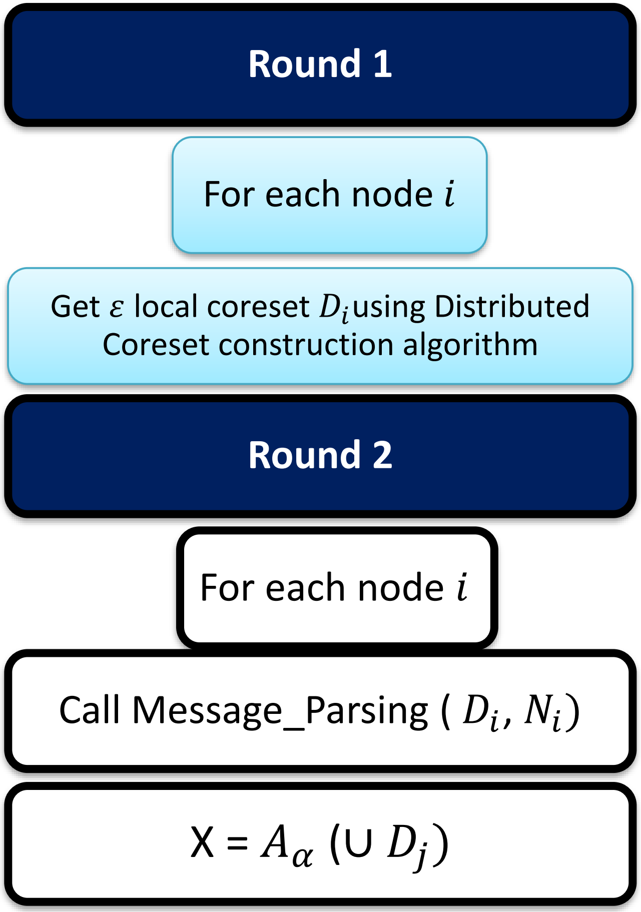
Round 1

Step 1:
Apply Distributed Coreset construction algorithm on data

Output:
Local weighted coresets D_i on each node i



Weighted Coresets



Round 1

For each node i

Get ε local coreset $D_i$using Distributed Coreset construction algorithm

Round 2

For each node i

Call Message_Parsing (D_i, N_i)

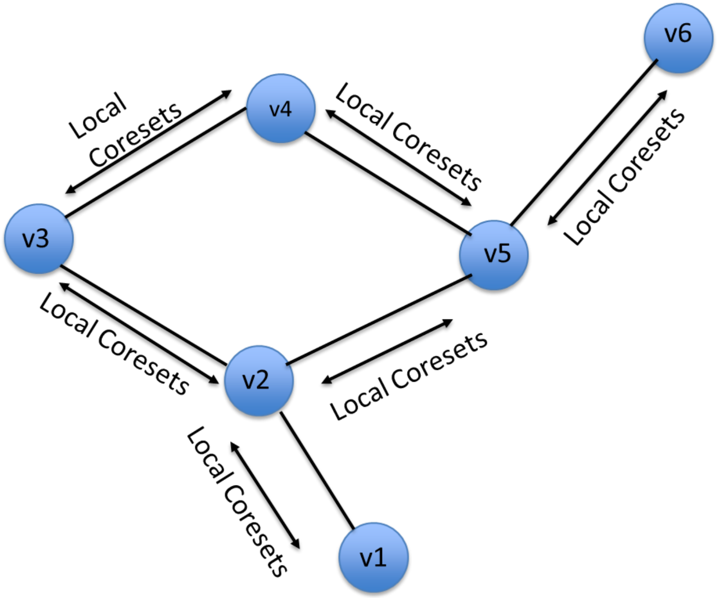
X = A_α (∪ D_j)

Round 2

Step 1:

Distributed coresets are communicated to each node

Using Message Passing algorithm



Round 1

For each node i

Get ε local coreset D_i using Distributed Coreset construction algorithm

Round 2

For each node i

Call Message_Parsing (D_i, N_i)

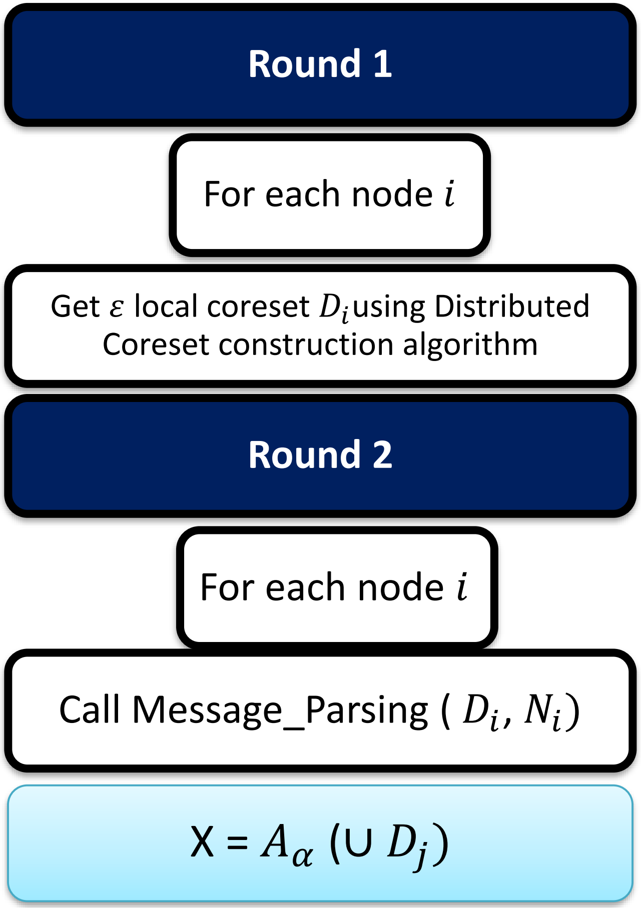
X = A_α (∪ D_j)

Round 2

Step 2:
Lloyd’s clustering algorithm is applied to compute the centers for the global dataset

Output:

Centers of the Global dataset



Round 1

For each node i

Get ε local coreset D_i using Distributed Coreset construction algorithm

Round 2

For each node i

Call Message_Parsing (D_i, N_i)

X = A_α (∪ D_j)

Random Graph

Each edge is generated independently with probability p.

For experiments, we used p = 0.3 and 10 nodes.

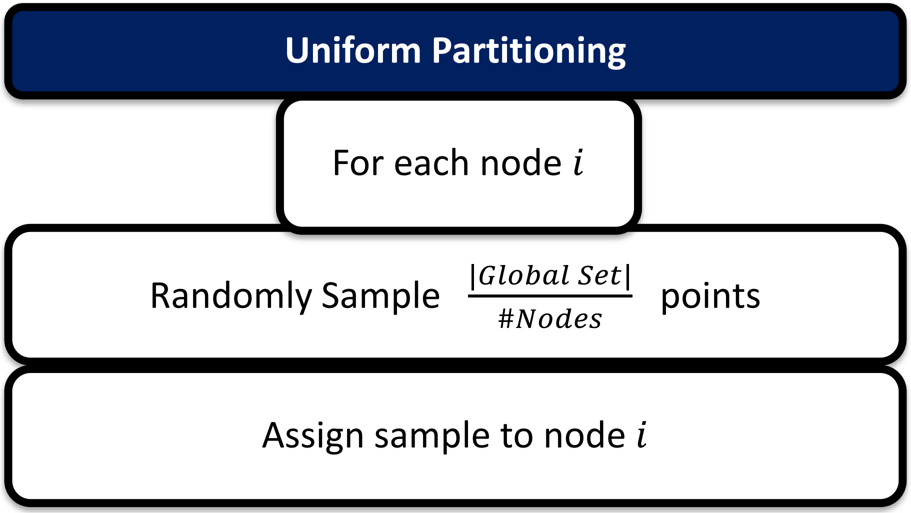
Preferential Graph

Preferential graphs are generated using the preferential attachment process.

Additional edges are added continuously to the graph that are distributed among the nodes as an increasing function of the number of edges the nodes already have.

Uniform Partitioning

Each data point from global dataset is assigned to a local site with equal probability.



Uniform Partitioning

For each node i

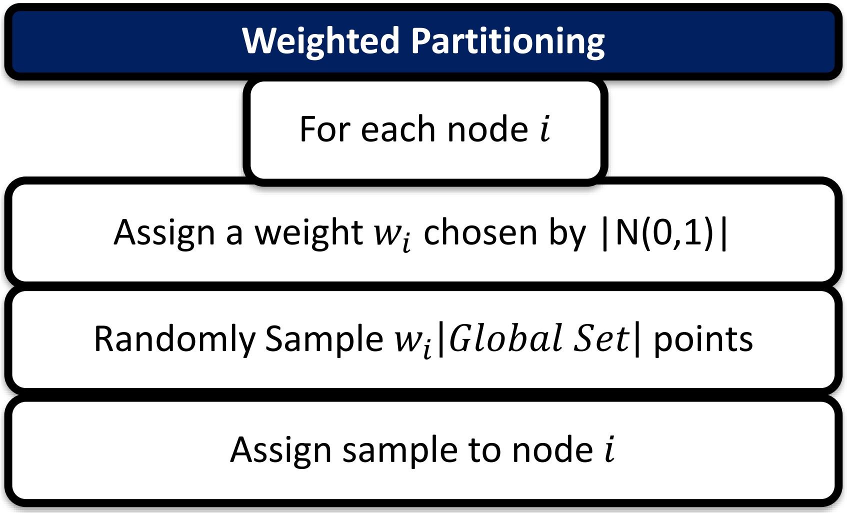
Randomly Sample $\frac{|Global\ Set|}{\#Nodes}$ points

Assign sample to node i

P(assigning a global data point to a node $\boldsymbol{i}$) = $\boldsymbol{\frac{1}{No.\ of\ nodes}}$

Weighted Partitioning

Each data point from global dataset is assigned to a local site with probability proportional to the nodes’ weight.



Weighted Partitioning

For each node i

Assign a weight w_i chosen by |N(0,1)|

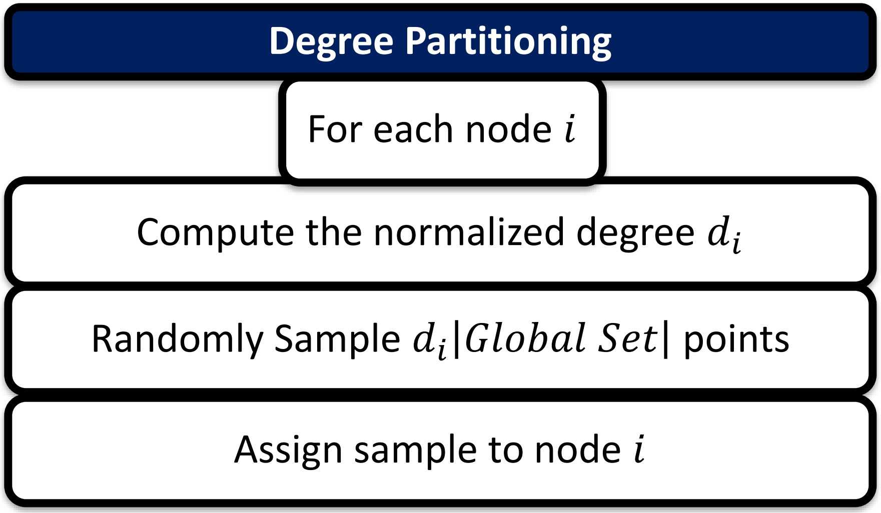
Randomly Sample $w_i|Global\ Set|$ points

Assign sample to node i

P(assigning a global data point to a local site $\boldsymbol{i}$) = w_i

Degree Partitioning

Each data point from global dataset is assigned to a local site with probability proportional to the nodes' degree.



Degree Partitioning

For each node i

Compute the normalized degree d_i

Randomly Sample $d_i|Global\ Set|$ points

Assign sample to node i

$$d_i = \frac{Degree\ (Node\ i)}{\sum_{j \in Nodes} Degree(Node\ j)}$$

P(assigning a global data point to a local site *i*) = d_i

Results

Inferences

- For all the datasets k-means cost ratio was plotted against communication cost.

- k-means cost ratio is defined as the ratio obtained by running Lloyd's algorithm on the coreset and the global data respectively to get two solutions, and computing the ratio between the costs of the two solutions over the global data.

- For all the datasets the k-means cost ratio reduced when communication cost increases. This implies that when larger coresets are generated the solution of the proposed algorithm converges to the k-means solution.

- Same results were obtained on Random and Preferential graph with three types of partitioning namely, uniform partitioning, weighted partitioning and degree partitioning.

- Results were also consistent on different dataset.

Datasets Used

Following 4 datasets were used

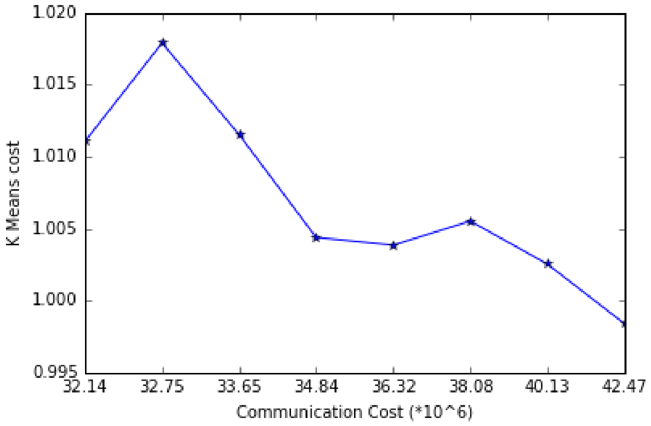
1) **Corel Image Features:** This dataset contains image features extracted from a Corel There are total 68040 observations.

2) **Spambase Data Set:** There are total 4601 observations with 58 dimensions. There were 1813 marked as spam and 2788 marked as non-spam.

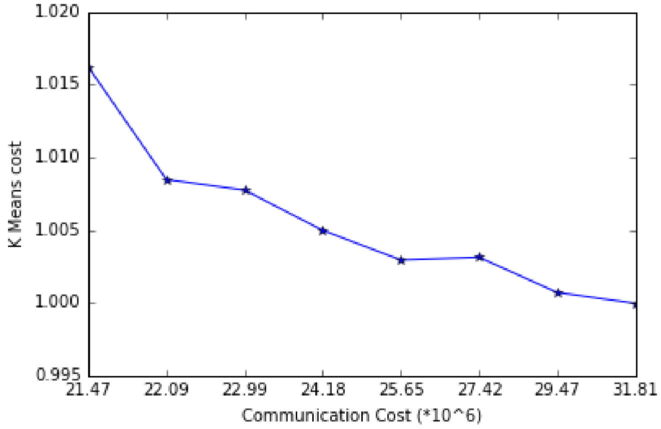
3) **Letter Recognition Dataset:** The dataset was made to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. There are total of 20000 observations each observation has 16 attributes.

4) **SFpark dataset:** The dataset contains data from three hundred parking lots in San Francisco city fitted with Smart Meters. In the processed data the rows represents Parking lots and columns represents time. There were total 300 rows (one for each parking lot) and 2880 columns (24 hours x 15 time slots x 30 days).

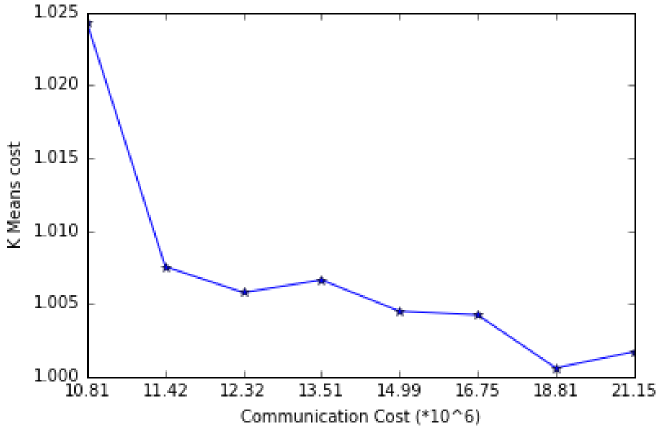
LETTER DATASET



Random Graph
Degree Partitioning

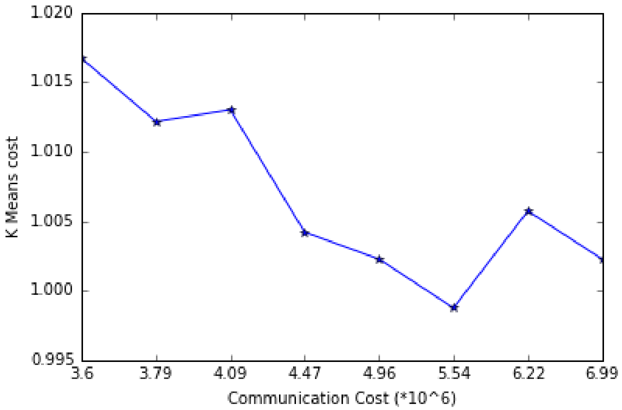


Random Graph
Weighted Partitioning

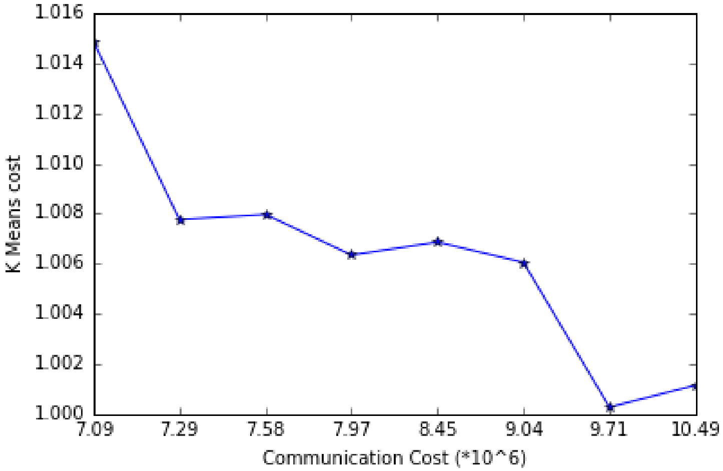


Random Graph
Uniform Partitioning

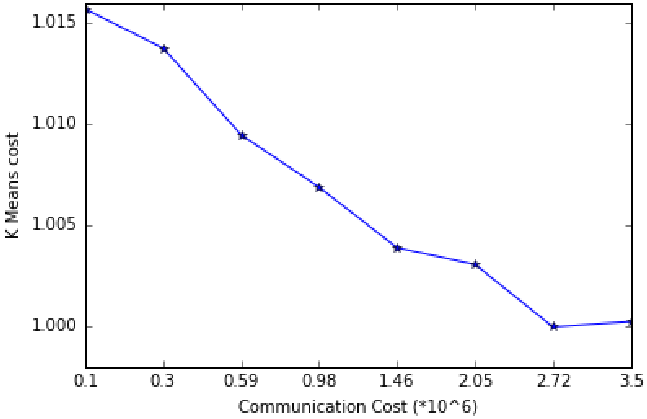
LETTER DATASET



Preferential Graph
Weighted Partitioning

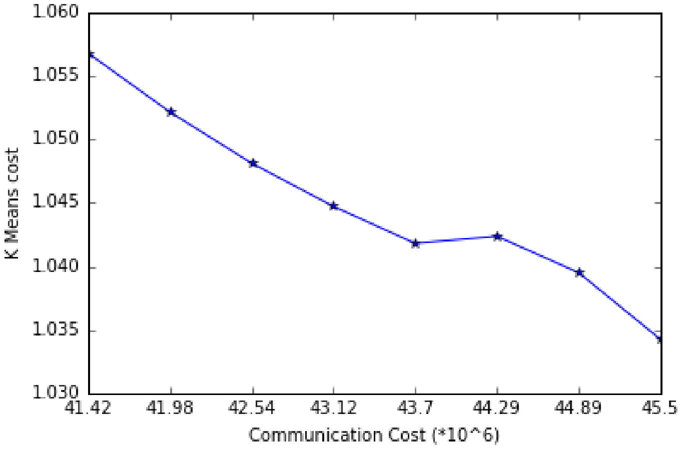


Preferential Graph
Uniform Partitioning

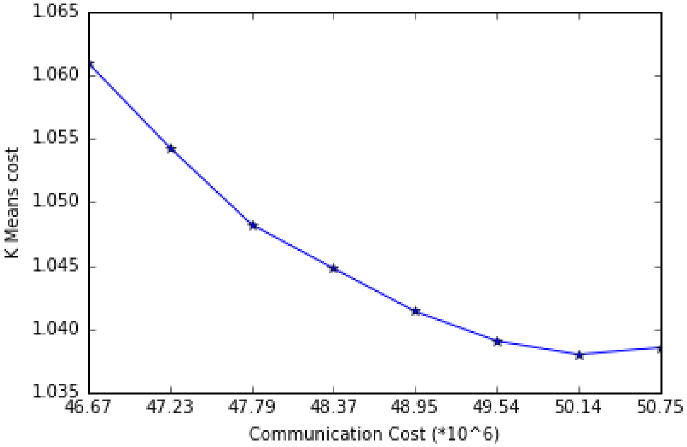


Preferential Graph
Degree Partitioning

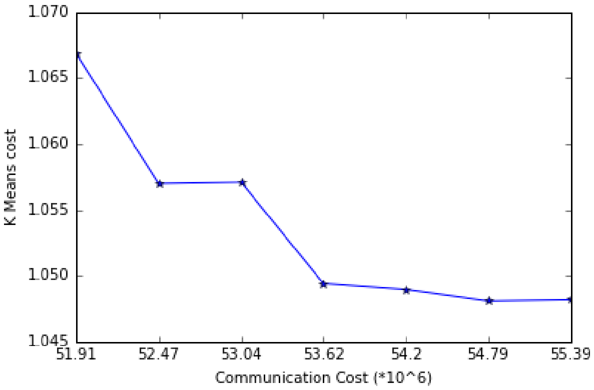
SPAM DATASET



Preferential Graph
Degree Partitioning

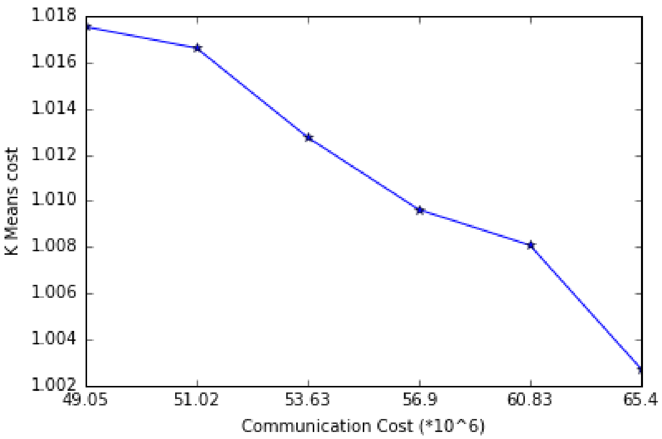


Preferential Graph
Weighted Partitioning

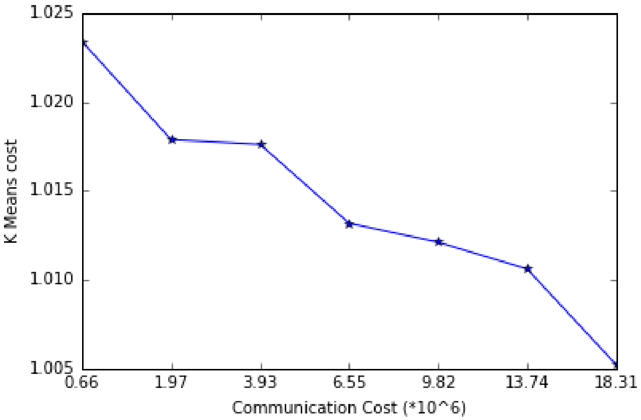


Preferential Graph
Uniform Partitioning

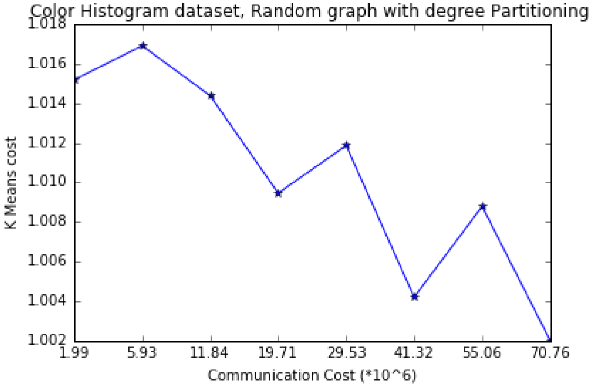
COLOR HISTOGRAM DATASET



Preferential Graph
Degree Partitioning

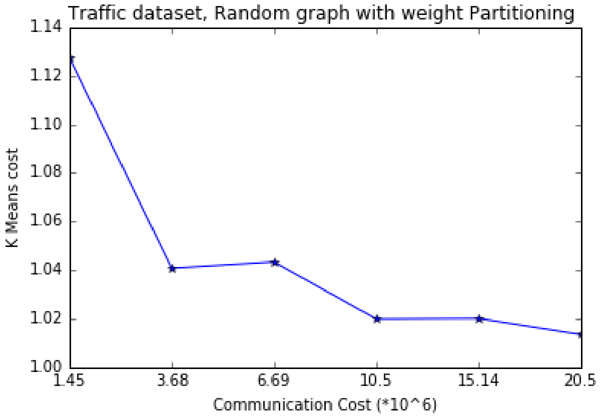


Preferential Graph
Weighted Partitioning

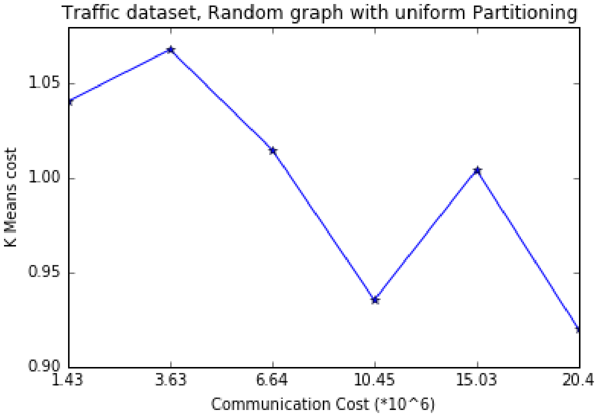


Random Graph
Degree Partitioning

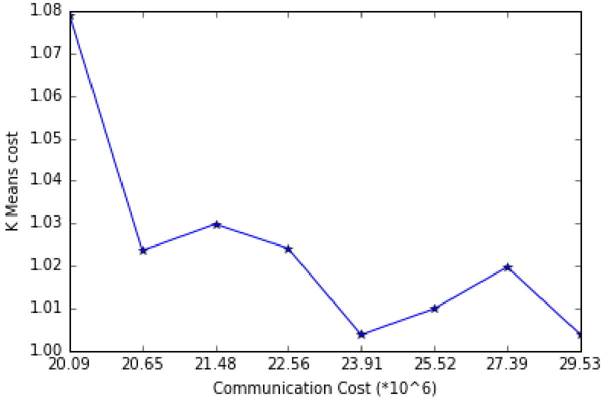
TRAFFIC DATASET



Random Graph
Weighted Partitioning



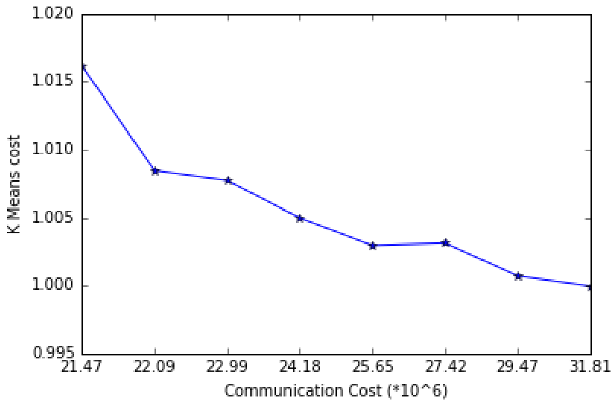
Random Graph
Uniform Partitioning

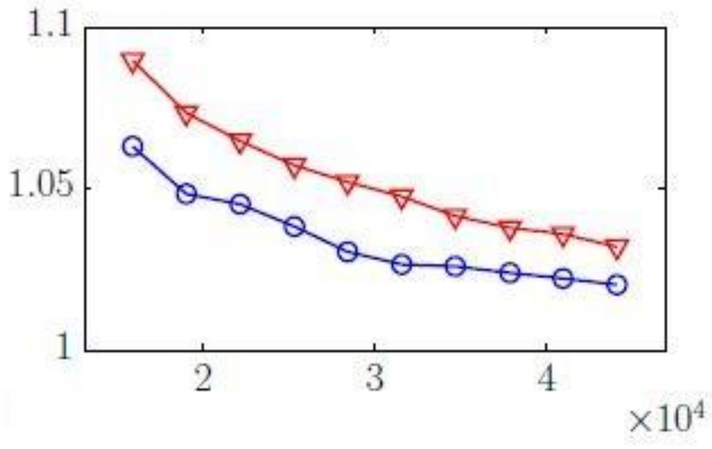


Preferential Graph
Uniform Partitioning

Summary

- Datasets are getting larger in size, so they are being stored on distributed systems. Increasing the efficiency of the algorithm which work on such distributed systems has gained importance.
- Based on the algorithm proposed in the paper a python code was written to perform distributed k-means clustering.
- 4 different datasets from two different sources was used to evaluate the performance of the algorithm.
- The k-means cost ratio reduces with increase in communication cost.
- We were able to replicate results given in the paper.
- Applications and connections to course





Comparison between two graphs
(Letter Random Weighted)

References:

[1] Q. Zhang, J. Liu, and W. Wang. Approximate clustering on distributed data streams. In Proceedings of the IEEE International Conference on Data Engineering, 2008.

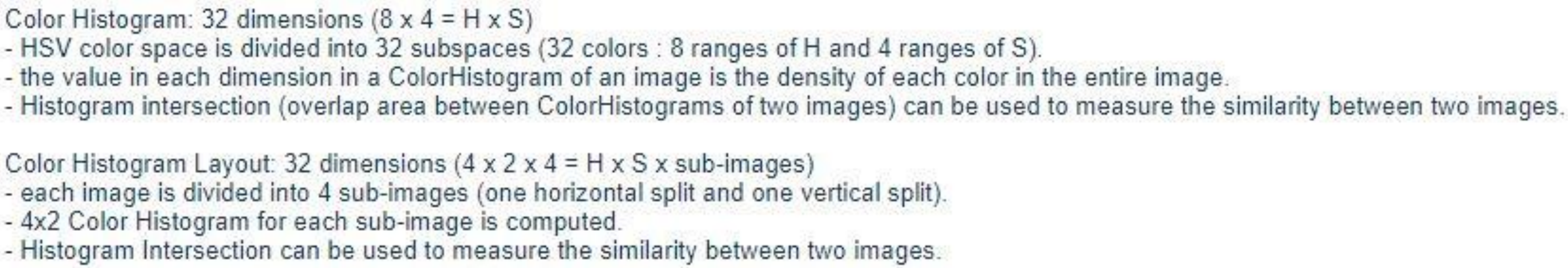
[2] D. Feldman, A. Sugaya, and D. Rus. An effective coreset compression algorithm for large scale sensor networks. In Proceedings of the International Conference on Information Processing in Sensor Networks, 2012.

[3] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[4] S.R.Boselin Prabhu, S.Sophia, S.Maheswaran, M.Navaneethakrishnan. Real-World Applications of Distributed Clustering Mechanism in Dense Wireless Sensor Networks, 2013.

[5] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 2003.

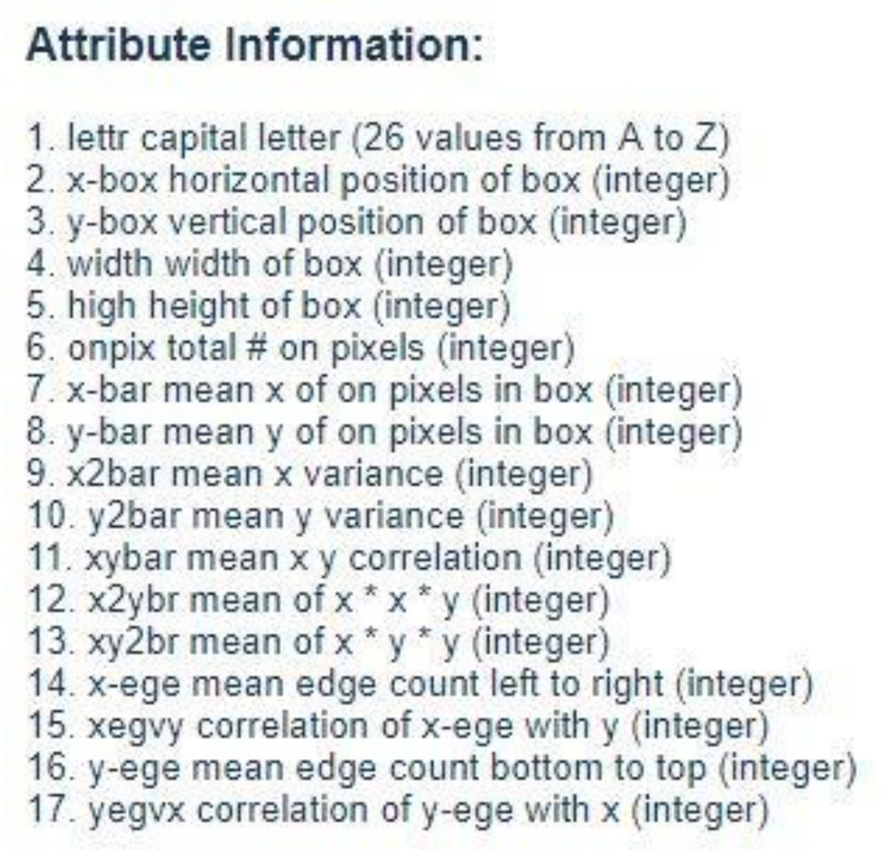
THE END



Color Histogram: 32 dimensions (8 x 4 = H x S)
- HSV color space is divided into 32 subspaces (32 colors : 8 ranges of H and 4 ranges of S).
- the value in each dimension in a ColorHistogram of an image is the density of each color in the entire image.
- Histogram intersection (overlap area between ColorHistograms of two images) can be used to measure the similarity between two images.

Color Histogram Layout: 32 dimensions (4 x 2 x 4 = H x S x sub-images)
- each image is divided into 4 sub-images (one horizontal split and one vertical split).
- 4x2 Color Histogram for each sub-image is computed.
- Histogram Intersection can be used to measure the similarity between two images.

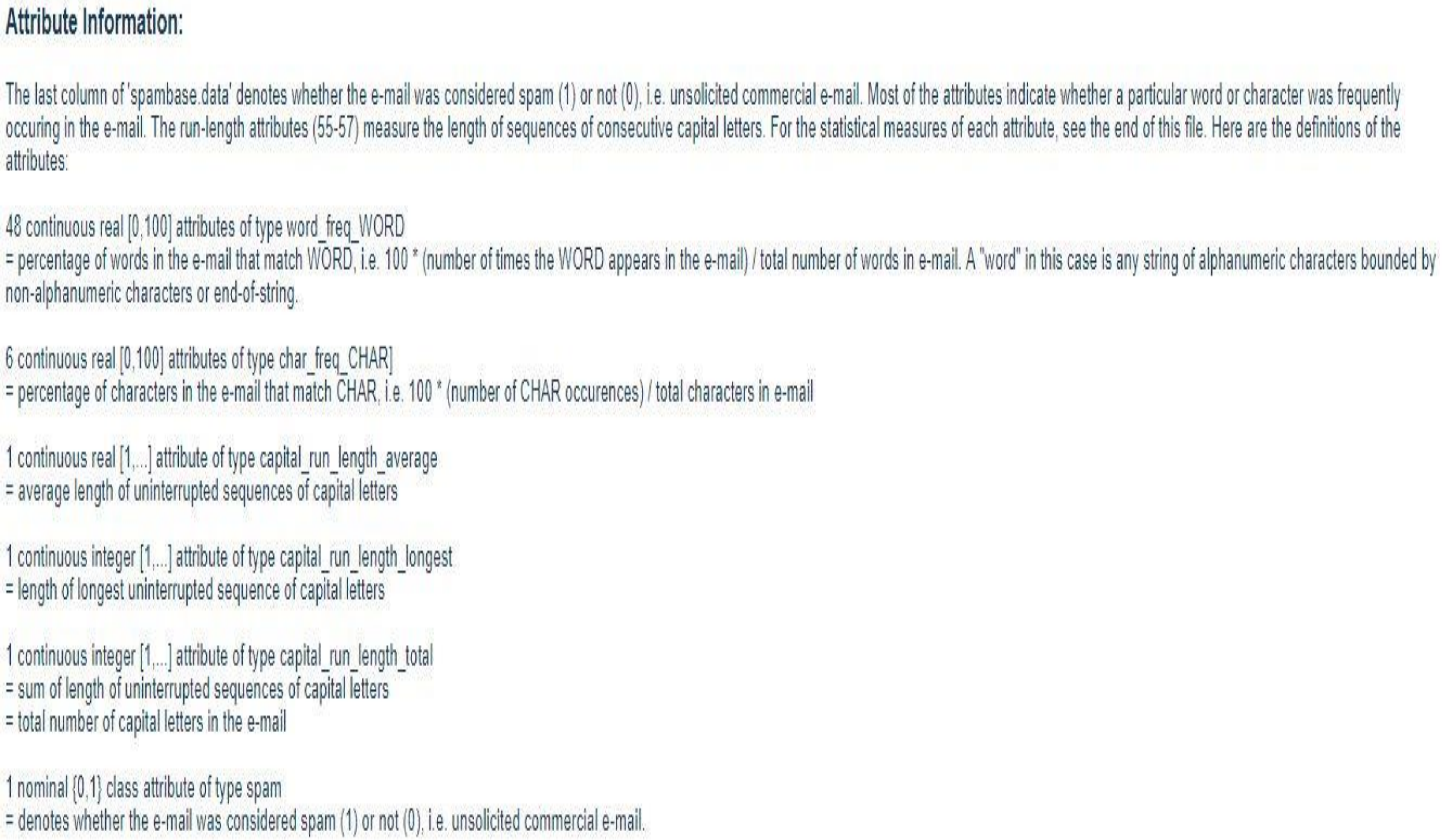
The objective is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15. We typically train on the first 16000 items and then use the resulting model to predict the letter category for the remaining 4000. See the article cited above for more details.



Attribute Information:

1. lettr capital letter (26 values from A to Z)
2. x-box horizontal position of box (integer)
3. y-box vertical position of box (integer)
4. width width of box (integer)
5. high height of box (integer)
6. onpix total # on pixels (integer)
7. x-bar mean x of on pixels in box (integer)
8. y-bar mean y of on pixels in box (integer)
9. x2bar mean x variance (integer)
10. y2bar mean y variance (integer)
11. xybar mean x y correlation (integer)
12. x2ybr mean of x * x * y (integer)
13. xy2br mean of x * y * y (integer)
14. x-ege mean edge count left to right (integer)
15. xegvy correlation of x-ege with y (integer)
16. y-ege mean edge count bottom to top (integer)
17. yegvx correlation of y-ege with x (integer)

EXTRA



Attribute Information:

The last column of 'spambase.data' denotes whether the e-mail was considered spam (1) or not (0), i.e. unsolicited commercial e-mail. Most of the attributes indicate whether a particular word or character was frequently occuring in the e-mail. The run-length attributes (55-57) measure the length of sequences of consecutive capital letters. For the statistical measures of each attribute, see the end of this file. Here are the definitions of the attributes:

48 continuous real [0,100] attributes of type word_freq_WORD
= percentage of words in the e-mail that match WORD, i.e. 100 * (number of times the WORD appears in the e-mail) / total number of words in e-mail. A "word" in this case is any string of alphanumeric characters bounded by non-alphanumeric characters or end-of-string.

6 continuous real [0,100] attributes of type char_freq_CHAR]
= percentage of characters in the e-mail that match CHAR, i.e. 100 * (number of CHAR occurences) / total characters in e-mail

1 continuous real [1,...] attribute of type capital_run_length_average
= average length of uninterrupted sequences of capital letters

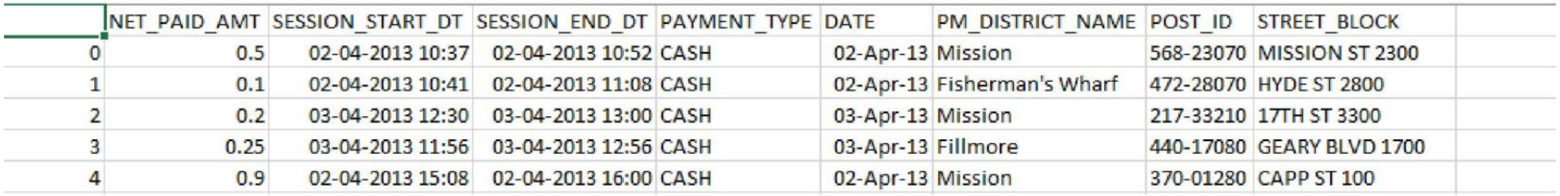
1 continuous integer [1,...] attribute of type capital_run_length_longest
= length of longest uninterrupted sequence of capital letters

1 continuous integer [1,...] attribute of type capital_run_length_total
= sum of length of uninterrupted sequences of capital letters
= total number of capital letters in the e-mail

1 nominal {0,1} class attribute of type spam
= denotes whether the e-mail was considered spam (1) or not (0), i.e. unsolicited commercial e-mail.

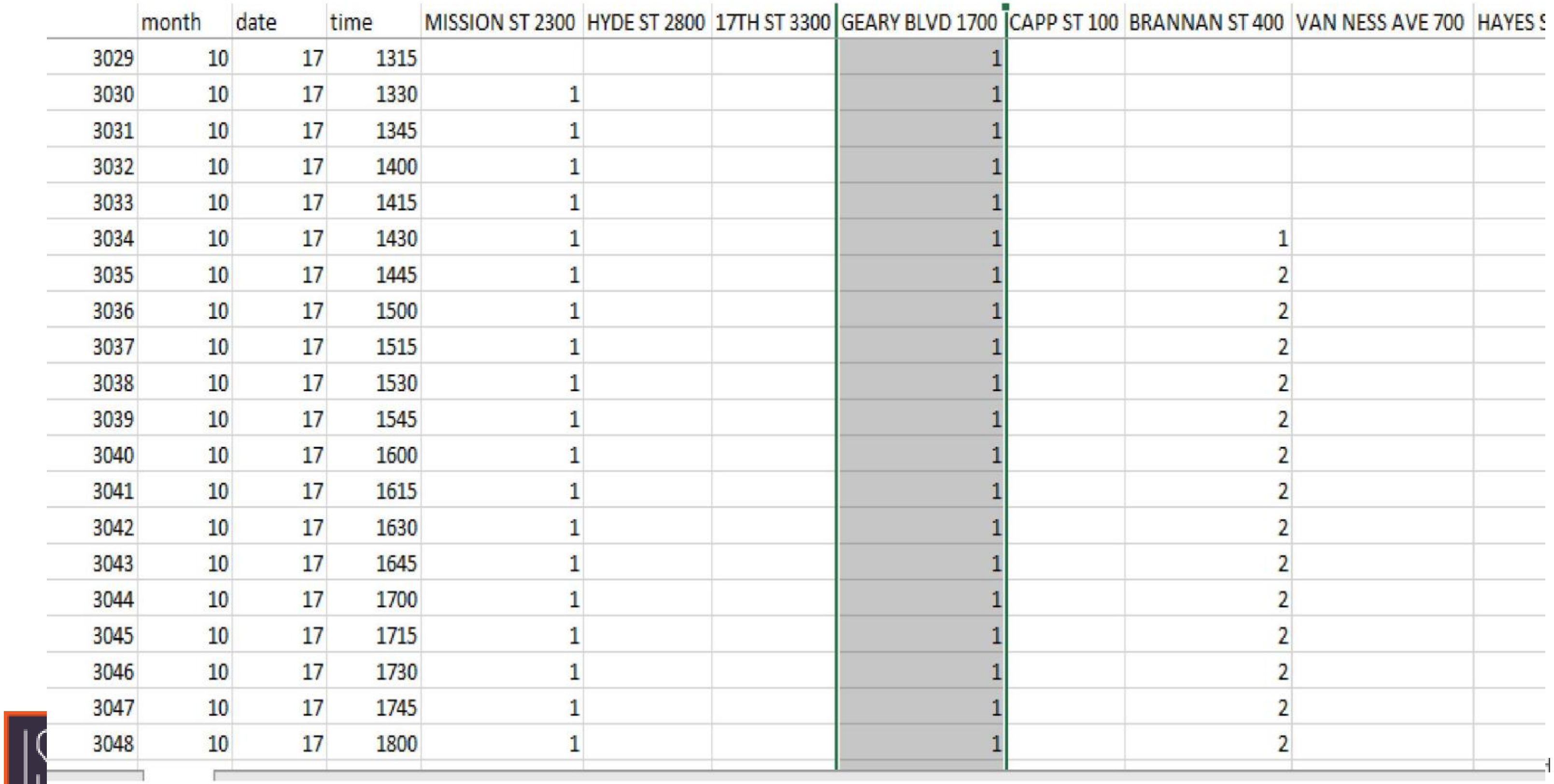
SF Parking Data

Raw data

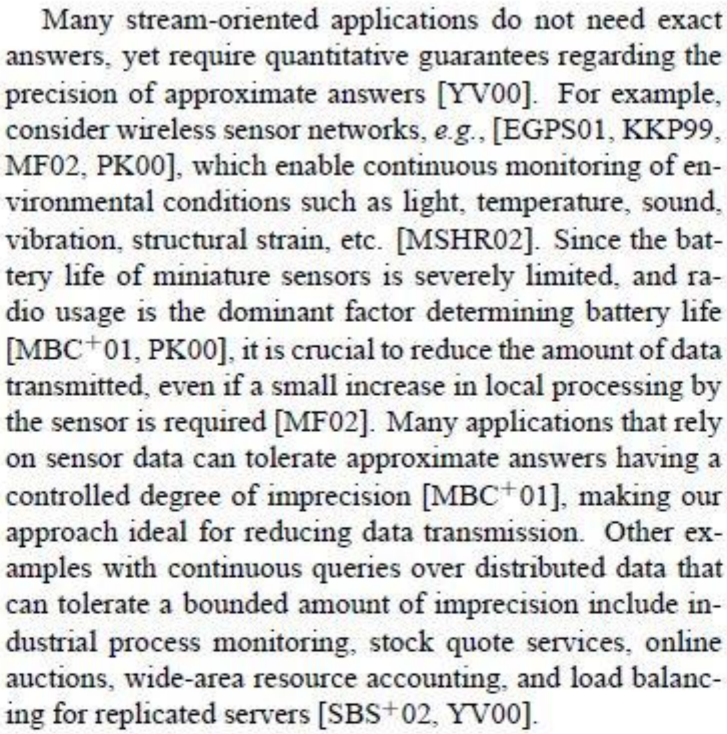


	NET_PAID_AMT	SESSION_START_DT	SESSION_END_DT	PAYMENT_TYPE	DATE	PM_DISTRICT_NAME	POST_ID	STREET_BLOCK
0	0.5	02-04-2013 10:37	02-04-2013 10:52	CASH	02-Apr-13	Mission	568-23070	MISSION ST 2300
1	0.1	02-04-2013 10:41	02-04-2013 11:08	CASH	02-Apr-13	Fisherman's Wharf	472-28070	HYDE ST 2800
2	0.2	03-04-2013 12:30	03-04-2013 13:00	CASH	03-Apr-13	Mission	217-33210	17TH ST 3300
3	0.25	03-04-2013 11:56	03-04-2013 12:56	CASH	03-Apr-13	Fillmore	440-17080	GEARY BLVD 1700
4	0.9	02-04-2013 15:08	02-04-2013 16:00	CASH	02-Apr-13	Mission	370-01280	CAPP ST 100

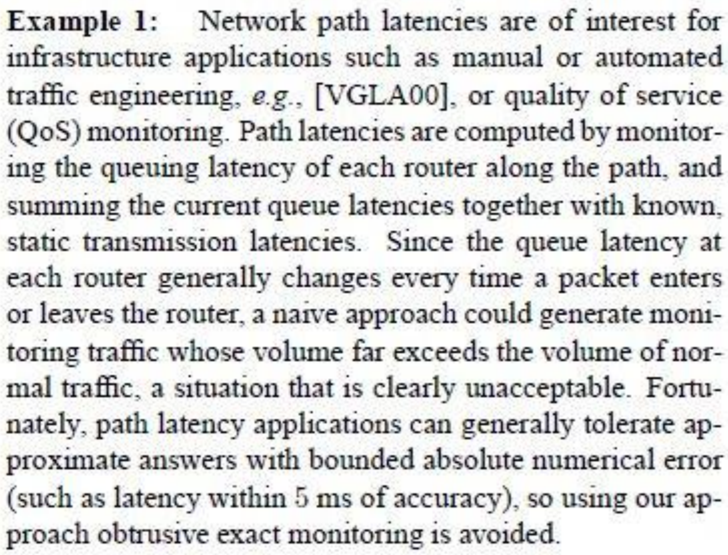
Processed data



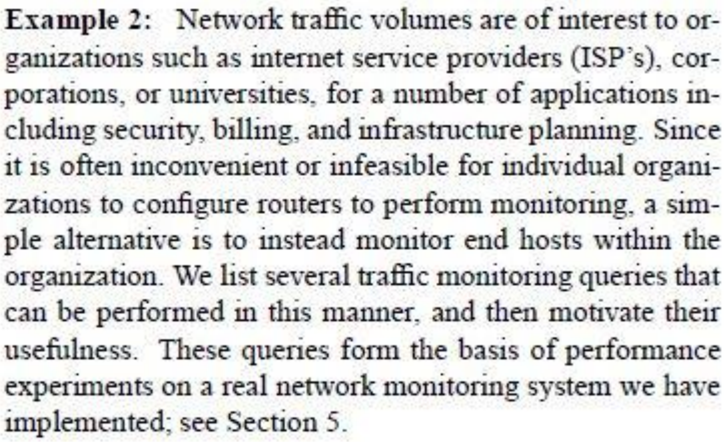
	month	date	time	MISSION ST 2300	HYDE ST 2800	17TH ST 3300	GEARY BLVD 1700	CAPP ST 100	BRANNAN ST 400	VAN NESS AVE 700	HAYES S
3029	10	17	1315				1				
3030	10	17	1330	1			1				
3031	10	17	1345	1			1				
3032	10	17	1400	1			1				
3033	10	17	1415	1			1				
3034	10	17	1430	1			1		1		
3035	10	17	1445	1			1		2		
3036	10	17	1500	1			1		2		
3037	10	17	1515	1			1		2		
3038	10	17	1530	1			1		2		
3039	10	17	1545	1			1		2		
3040	10	17	1600	1			1		2		
3041	10	17	1615	1			1		2		
3042	10	17	1630	1			1		2		
3043	10	17	1645	1			1		2		
3044	10	17	1700	1			1		2		
3045	10	17	1715	1			1		2		
3046	10	17	1730	1			1		2		
3047	10	17	1745	1			1		2		
3048	10	17	1800	1			1		2		



Many stream-oriented applications do not need exact answers, yet require quantitative guarantees regarding the precision of approximate answers [YV00]. For example, consider wireless sensor networks, *e.g.*, [EGPS01, KKP99, MF02, PK00], which enable continuous monitoring of environmental conditions such as light, temperature, sound, vibration, structural strain, etc. [MSHR02]. Since the battery life of miniature sensors is severely limited, and radio usage is the dominant factor determining battery life [MBC+01, PK00], it is crucial to reduce the amount of data transmitted, even if a small increase in local processing by the sensor is required [MF02]. Many applications that rely on sensor data can tolerate approximate answers having a controlled degree of imprecision [MBC+01], making our approach ideal for reducing data transmission. Other examples with continuous queries over distributed data that can tolerate a bounded amount of imprecision include industrial process monitoring, stock quote services, online auctions, wide-area resource accounting, and load balancing for replicated servers [SBS+02, YV00].



Example 1: Network path latencies are of interest for infrastructure applications such as manual or automated traffic engineering, *e.g.*, [VGLA00], or quality of service (QoS) monitoring. Path latencies are computed by monitoring the queuing latency of each router along the path, and summing the current queue latencies together with known, static transmission latencies. Since the queue latency at each router generally changes every time a packet enters or leaves the router, a naive approach could generate monitoring traffic whose volume far exceeds the volume of normal traffic, a situation that is clearly unacceptable. Fortunately, path latency applications can generally tolerate approximate answers with bounded absolute numerical error (such as latency within 5 ms of accuracy), so using our approach obtrusive exact monitoring is avoided.



Example 2: Network traffic volumes are of interest to organizations such as internet service providers (ISP's), corporations, or universities, for a number of applications including security, billing, and infrastructure planning. Since it is often inconvenient or infeasible for individual organizations to configure routers to perform monitoring, a simple alternative is to instead monitor end hosts within the organization. We list several traffic monitoring queries that can be performed in this manner, and then motivate their usefulness. These queries form the basis of performance experiments on a real network monitoring system we have implemented; see Section 5.

Definitions

Definition 1:(ϵ-*Coreset*)

An ϵ-coreset for a set of points P with respect to a center-based cost function is a set of points S and a set of weights w:S→R such that for any set of centers X,

$$(1-\epsilon)cost(P,X) \leq \sum_{p \in S} w(p)cost(p,X) \leq (1+\epsilon)cost(P,X)$$

Definition 2: (*Dimension of the function space*)

Let F be a finite set of functions from a set P to R_0. Then for *f ϵ F,* define the set *B*(f, r) ={ p: f(p) ≤ r}. Then the dimension of the function space D(F, P) is the smallest integer d such that for any G ⊆ P,

$$|\{G \cap B(f,r) : f \in F,\ r \geq 0\}| \leq |G|^d$$