Time series subsequence clustering

Using Toeplitz Inverse Covariance

The Problem:

Clustering subsequences of time series data.

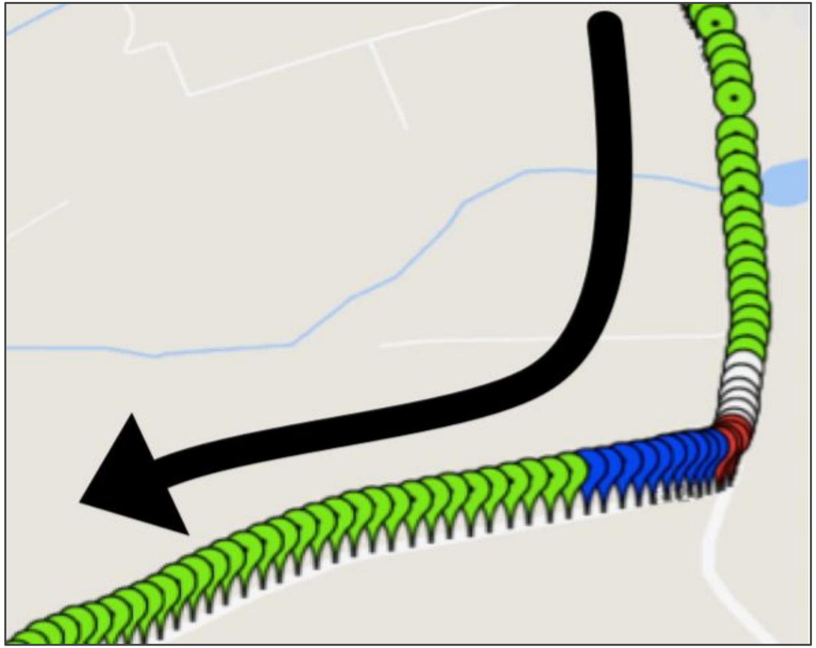
Characteristics of sensor data:

- High dimensional
- High velocity
- Dynamic over time - Evolving
- Heterogenous - Comes in from different data streams

Approach:

- Instead of matching raw values like in distance based similarity measures, look for structural similarity in the data.
- But, structure of each state/cluster is unknown.
- Need for unsupervised clustering algorithm, to discover the states while simultaneously breaking the time series into the sequence of these states.

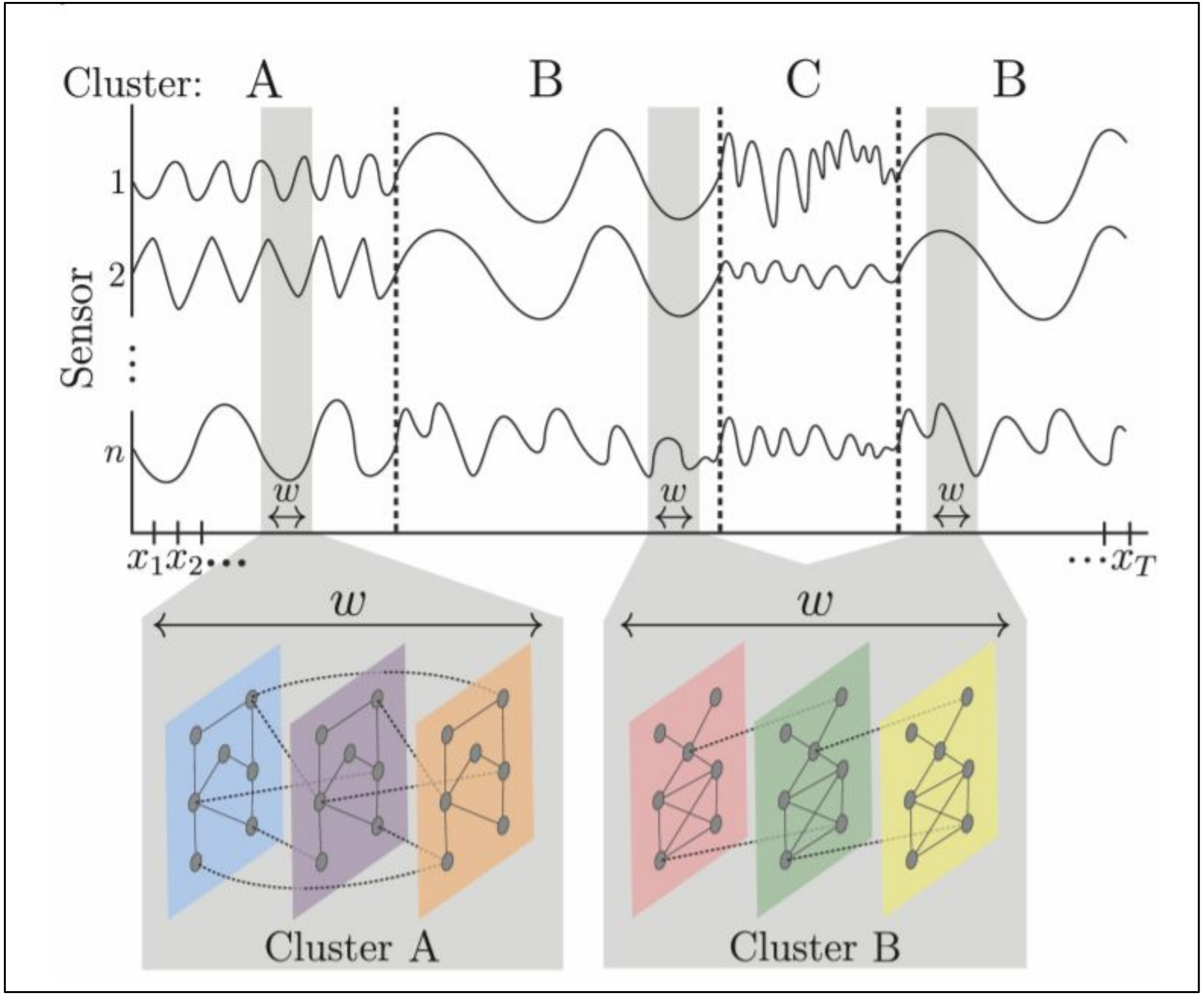
Example:



- Green - Going straight
- White - Slowing down
- Red - Turning
- Blue - Speeding up

Proposed Solution:

- Each point assigned to cluster
  - Points encouraged to be in the cluster to which it’s neighbors belong in order to extract subsequences
  - Points not inspected in isolation, but whole window is considered which provides context for the data
- Each state/cluster is defined using a multilayer correlation network (MRF) between the sensors for a particular window.
  - MRFs encode conditional dependencies among the sensors
  - 2 types of dependencies encoded
    - Intratime - At same time within the sensors
    - Intertime - How sensors current conditions affect the one’s in future



Cluster: A B C B
Sensor
1
2
n
w
$x_1 x_2 \cdots$
$\cdots x_T$
w
Cluster A
w
Cluster B

Proposed Solution:

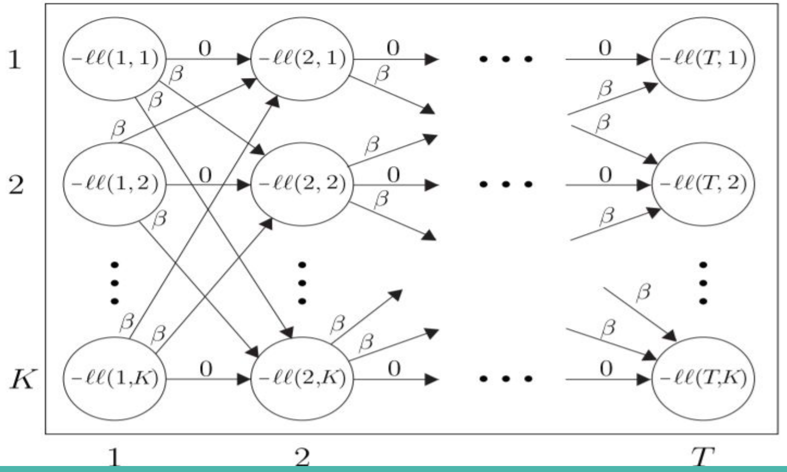
- Two sets of parameters to be optimised simultaneously -
  - ‘p’ - point assignment in time to cluster
  - ‘Theta’ - Cluster parameters defining MRF
- Aim to achieve
  - Sparsity - More interpretability, less overfitting and ability to distinguish between clusters
  - Likelihood - For every point assigned to cluster ‘i’, the cluster distribution should match the point
  - Temporal consistency - Points encouraged to belong cluster to which it’s neighbors belong.
- Hyperparameters - optimised using bayesian information criterion
  - K - # clusters - Can be chosen based on the application
  - Lambda - Sparsity
  - Beta - Temporal consistency

Algorithm:

- To achieve globally optimal solution, we use a variation of the expectation maximization (EM) algorithm to alternate between
  - Assigning points - Hold cluster parameters constant and assign points in temporally consistent way
  - Update cluster parameters - Hold the points constant and update.
- Repeat till convergence.

Cluster assignment:

- Assigning ‘T’ subsequences, to ‘K’ clusters in a way that maximizes the likelihood of the data while minimizing the number of times the cluster assignment changes across the time series.
- ‘KT’ possible assignments of points to clusters to choose from.
- Done in O(KT) operations using DP method similar to Viterbi algorithm.



Parameter updation:

M-step : Update the inverse covariances, given the points assigned to each cluster.

- Clusters are represented as Toeplitz Matrices

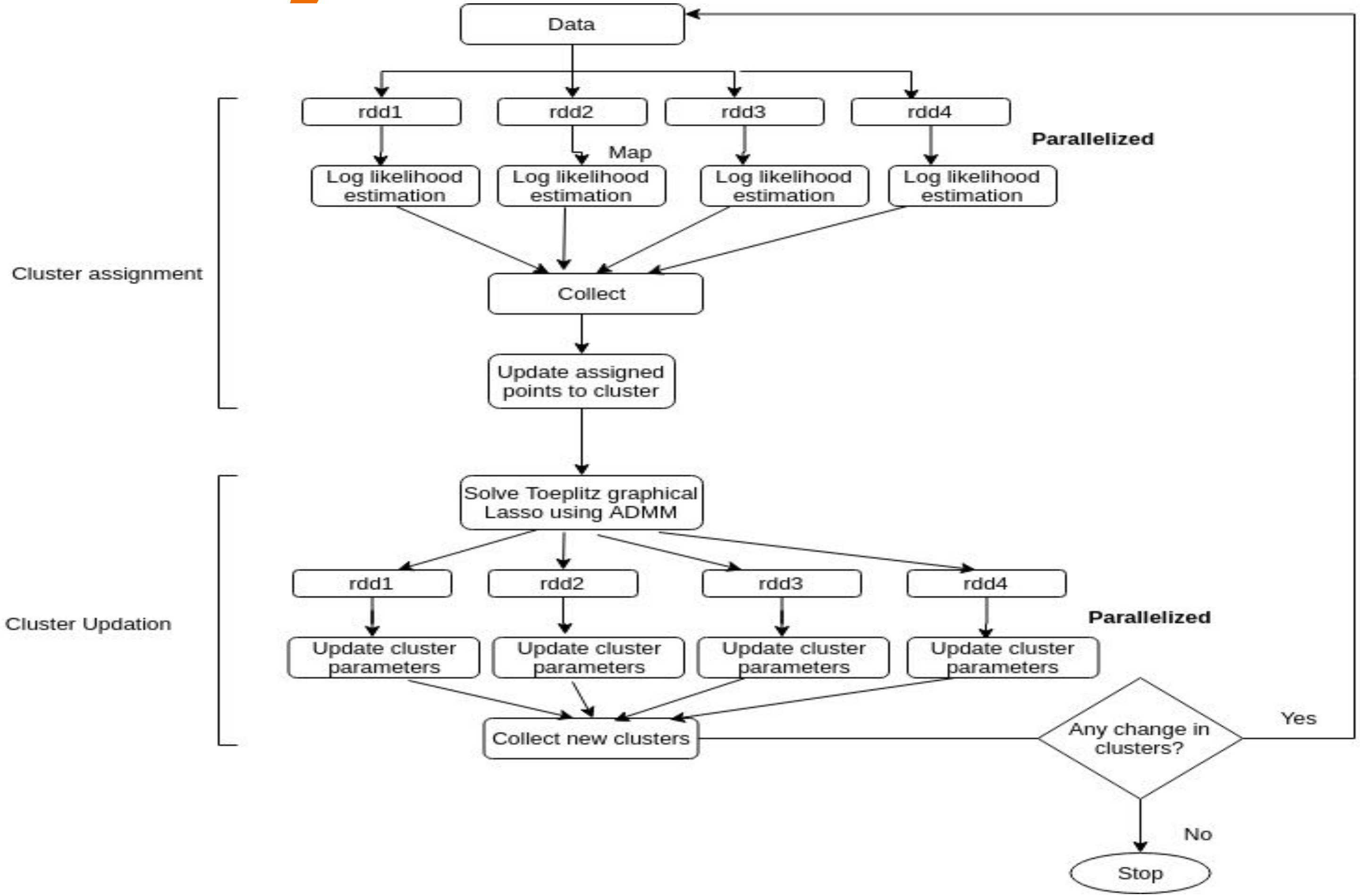
$$\Theta_i = \begin{bmatrix} A^{(0)} & (A^{(1)})^T & (A^{(2)})^T & \cdots & \cdots & (A^{(w-1)})^T \\ A^{(1)} & A^{(0)} & (A^{(1)})^T & \ddots & & \vdots \\ A^{(2)} & A^{(1)} & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & (A^{(1)})^T & (A^{(2)})^T \\ \vdots & & \ddots & A^{(1)} & A^{(0)} & (A^{(1)})^T \\ A^{(w-1)} & \cdots & \cdots & A^{(2)} & A^{(1)} & A^{(0)} \end{bmatrix},$$

A ij refers to the relationship between concurrent values of sensors i and j.

- Alternating direction method of multipliers (ADMM) has been used to update clusters.
- Must be done for each cluster. Each cluster has a dimension of nw*nw.

  N- no of features/sensors, W-windows size

System Diagram



Contribution:

Cluster assignment has 2 parts :

- Log likelihood estimation
- Update points assigned to clusters

Parameter updation has 2 parts :

- Solve Toeplitz Graphical lasso using ADMM
- Update parameters of cluster

Our improvement :

- Parallelize Log likelihood estimation and update cluster parameters.
- As algorithm uses window size, data partitioning must be done before parallelizing.

Remaining are not data parallel. So cannot parallelize.

Results

```
length of cluster # 0 --------> 8067
length of cluster # 1 --------> 3365
length of cluster # 2 --------> 821
length of cluster # 3 --------> 880
length of cluster # 4 --------> 1307
length of cluster # 5 --------> 1613
length of cluster # 6 --------> 1354
length of cluster # 7 --------> 2200
--- 21.263651132583618 seconds ---
```

Original code

```
('length of cluster #', 0, '-------->', 8067)
('length of cluster #', 1, '-------->', 3365)
('length of cluster #', 2, '-------->', 821)
('length of cluster #', 3, '-------->', 880)
('length of cluster #', 4, '-------->', 1307)
('length of cluster #', 5, '-------->', 1613)
('length of cluster #', 6, '-------->', 1354)
('length of cluster #', 7, '-------->', 2200)
--- 88.5034368038 seconds ---
```

Distributed code

- Parallelized algorithm gave same accuracy.
- Took longer time to execute - Spark context overhead.
- Can give speedup if cluster setup is used.

Thank You!