Contents

[**Constrained K-means algorithm – COP-Kmeans** (Wagstaff, Claire, Seth, & Stefan, 2001) 2](#_Toc63081717)

[**COP-Kmeans algorithm pseudocode: (incomplete)** 3](#_Toc63081718)

[**LCC\_CKMeans Protocol** 4](#_Toc63081719)

[**Boosted Constrained K-means Algorithm – BCKM** (Okabe & Yamada, 2018) 5](#_Toc63081720)

[**Algorithm 2 - M-CKmeans** 6](#_Toc63081721)

[**Algorithm 3 - BCKM** 7](#_Toc63081722)

[SCOP-KMeans algorithm by Kiri Wagstaff 12](#_Toc63081723)

[**References** 15](#_Toc63081724)

**LCC scenario as Constrained K-means clustering problem**

# **Constrained K-means algorithm – COP-Kmeans** (Wagstaff, Claire, Seth, & Stefan, 2001)

**COP-Kmeans** (agent **set** A, **must-link constraints** , **cannot-link constraints** )

**INPUT:** Data set of agents, Must-link constraints and Cannot-link constraints

**OUTPUT**: A set of clusters or Empty set

1. Let and be the minimum and maximum number for the cardinality of each cluster as specified in the informal description.

1. Let and be the lower and higher bound of **k** interval:
2. Let be the initial cluster centers.
3. For each agent , assign it to the closest cluster such that is **false**. If **no** such cluster **exist**, increment by one and check whether is not greater than . If incremented is not greater than , then **break** iteration and go to step 3. If it is greater, then **fail** (return {}).
4. For each cluster, update its center by **averaging** all of the points that have been assigned to it.
5. Iterate between **step 4** and **step 5** until **convergence**.
6. Return

For each, return **true**

For each , return **true**

Otherwise, return **false**

(Wagstaff K. L., 2002)

# **COP-Kmeans algorithm pseudocode: (incomplete)**

**INPUT**: TCN Data set D, Must-Link constraints , Cannot-link constraints

1. Set the minimum and maximum number for the cardinality of each cluster.
2. Calculate the minimum and maximum values for K. K represents the number of clusters.

1. **WHILE** *K* is not greater than *Kmax*
2. Let be the initial cluster centers
3. **REPEAT**
4. **FOR** each data point in D
5. Assign it to the closest cluster center Cj such that no constraint is violated (=false)
6. **IF** no such cluster exist **THEN**
7. Since there is no cluster to assign dj without violating a constraint for this k value and cluster centers, increment k by one and go to step 3.
8. **ENDIF**
9. **ENDFOR**
10. **ENDIF**
11. **FOR** each cluster Cj in
12. Update its center by averaging all of the points that have been assigned to it
13. **ENDFOR**
14. **UNTIL** convergence of clustering
15. **IF** converged **THEN**
16. **OUTPUT** the set of clusters (return {})
17. **ENDWHILE**
18. **OUTPUT** empty cluster set (return {})

**INPUT**:

**OUTPUT:** true or false

1. **FOR** each pair in must-link constraints
2. **IF** is not in cluster C **THEN**
3. **RETURN** true
4. **ENDIF**
5. **ENDFOR**
6. **FOR** each pair in cannot-link constraints
7. **IF** is in cluster C **THEN**
8. **RETURN** true
9. **ENDIF**
10. **ENDFOR**
11. **RETURN** false

**===========================================================================**

# **LCC\_CKMeans Protocol**

1. **Start**
2. Teacher selects a set of TCNs (or a classroom) and initiates the coordination
3. WPM sends a signal to the agents of selected TCNs. The signal contains information about the list of agents and a dedicated agent who is being selected by WPM (randomly or by considering the resources)
4. Dedicated agent collects **personal info** (no preference) from other coordinating agents, builds the complete personal info and **broadcasts** it to the coordinating agents.
5. Upon receiving the complete info, each agent creates 2 sets: **must-link** constraints and **cannot-link** constraints based on its own preferences
6. Dedicated agent collects these pair of sets from each agent, **merges** them appropriately into 2 **complete** **must-link** and **complete cannot-link** constraints sets.
7. (Optional) Dedicated agent broadcasts the complete must-link and complete cannot-link constraints to the agents such that any of them would be able to solve the problem in case anything happens to the dedicated agent.
8. Dedicated agent **runs COP-Kmeans(**D, must-link, cannot-link**)** algorithm and gives the complete info, must-link, cannot-link constraints as the **input**.
9. Upon finding a solution, dedicated agent sends the **result to WPM** and **other agents**.
10. (Optional) Upon receiving the result, WPM might decide **to kill or not kill** the agents who were involved in the coordination.
11. Teacher selects from **rank list**, **modifies** or **approves** the grouping proposal of agents and then either a grouping result or an “*approved*” message is sent to all involved agents by WPM.
12. Each agent informs the TCN by sending the name of the group which the TCN belongs to. Regardless of the login state of the TCN, agent proceeds to the next step. Because once an agent receives the approved grouping from teacher, it stores the info in LAKR such that the TCN can request it anytime via the Welcome app.
13. Each agent sends a signal to WPM informing that the coordination is **ended**.

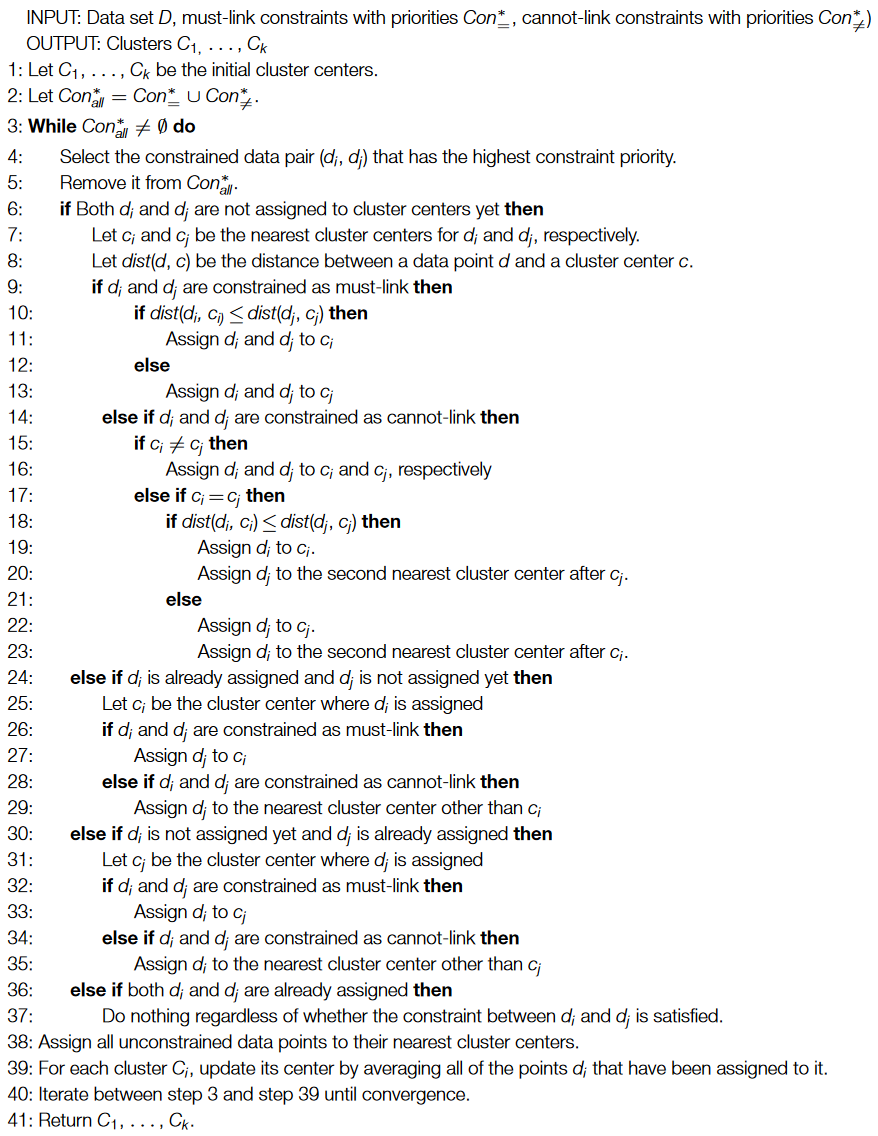
**==========================================================================**

# **Boosted Constrained K-means Algorithm – BCKM** (Okabe & Yamada, 2018)

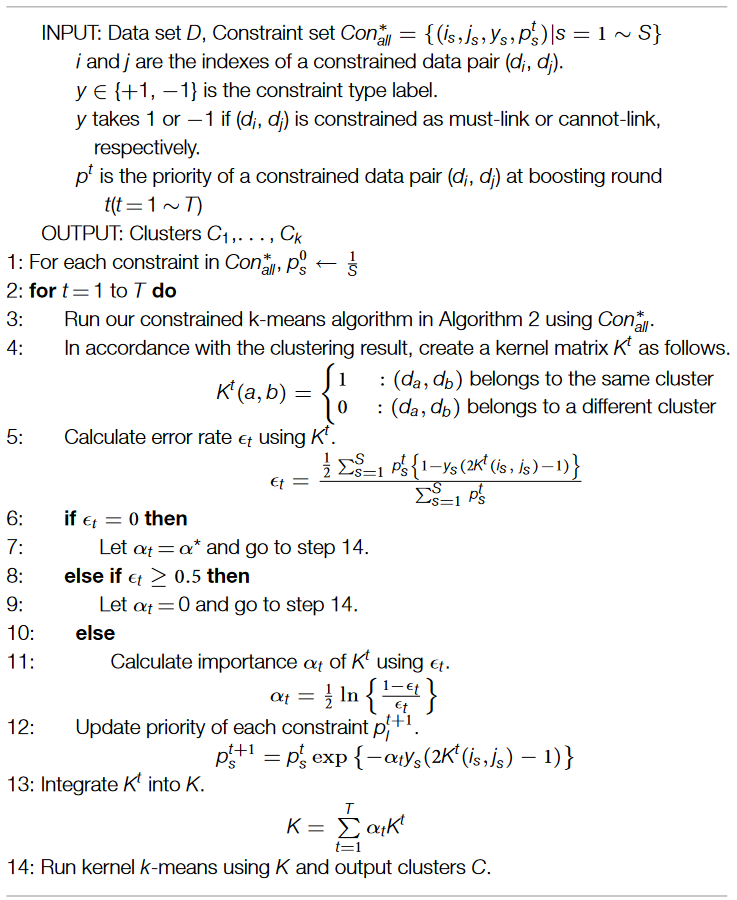
BCKM is an improved constrained k-means algorithm in terms of performance. It contains traditional COP-Kmeans and metric learning techniques. Although its computation time is worse than COP-Kmeans, it is fast enough according to the authors. In terms of performance, it is better than the most of the constrained clustering algorithms. BCKM algorithm is described in algorithm 3. It makes use of a modified constrained-kmeans algorithm which is described in algorithm 2. Both algorithms are taken from the paper.

**Modified Constrained K-means Algorithm – M-CKmeans** (Okabe & Yamada, 2018)

## **Algorithm 2 - M-CKmeans**



## **Algorithm 3 - BCKM**



Implications:

COP-KMeans:

* Constraints are treated as hard and if any of them is not satisfied, the algorithm aborts.

BCOP-KMeans:

* Finds a solution even when some constraints are not satisfied.

Course constraints should be integrated into BCOP-KMeans.

What about the hard cons part of cop-kmeans?

LCC\_BCOP-KMeans:



Ordered Cop-Kmeans algorithm



(Likas & Tzortzis, 2008)

(Likas & Tzortzis, 2008)

SCOP-KMeans algorithm by Kiri Wagstaff (Wagstaff K. L., 2002)

**SCOP-KMEANS** (number of clusters , agent set , preferences where and )

1. Let and be the minimum and maximum number for the cardinality of each cluster as specified in the informal description.
2. Let and be the lower and higher bound of **k** interval:
3. Let be the initial cluster centers.
4. For each agent in , assign it to the closest and lowest in terms of preference constraints violation ( such that no course constraint is violated (). If **no** such cluster **exist**, increment by one and check whether is not greater than . If so, then **break** iteration and go to step 3. If it is greater, then **fail** (return {}).

***Objective Function****:*

The functionranges from0 (same course progress level) to 100 (maximum course progress level).

The function ranges from 0 (no individual preference violated) to 1 (all individual preferences with strength 1.0 violated).

The function ranges from 0 (no course constraint violated) to 2 (both course constraints violated).

1. Update each cluster center by **averaging** all of the points that have been assigned to it.
2. Iterate between **step 4** and **step 5** until convergence.
3. Return the partition .

1. Let .
2. For each connection :

**If**

**If** and , then and .

**Else** **if** and , then and .

**Else** **if** ,

Increment by 1.

**If** and , then increment by 1.

**Else** **if** and , then increment by 1.

1. Return

1. Let
2. Check cardinality constraint:

**If** , then increment by 1.

1. In order to consider *missing previous lesson* constraint, let be a function which maps an agent to a value which represents whether he/she participated in the previous lesson:

**If** agent missed the previous lesson and , then increment by 1.

1. Return

1. Let function be a function that maps an agent into its Course Progress Level.
2. Return where is the cluster center of cluster C.

The function examines each constraint that applies to agent . is the maximum (absolute) strength of violated constraints. counts the number of constraints at strength and tracks how many of those are violated. The function then returns times the fraction of maximum strength constraints that are violated. Constraints at a lower strength may or may not also be violated, but the maximum strength violations take precedence. The function ranges from 0 (no individual preference violated) to 1 (all individual preferences with strength 1.0 violated).

The function calculates the number of course constraints that would be violated if agent was assigned to cluster . counts the number of violations. It is incremented by 1 each time a course constraint is violated. The function then returns the . The function ranges from 0 (no course constraint violated) to 2 (both course constraints violated).

The function given an agent and a cluster, calculates the distance between cluster center and agent course progress level. It ranges from0 (same course progress level) to 100 (maximum course progress level).

*Objective function:*

The minimization part of the objective function consists of similarity constraint (dist) and individual preferences (penalty for violation). The *Similarity* is described as a course constraint in the informal description of LCC scenario. Unlike other course constraints (e.g cardinality), it is not treated as a hard constraint. However, the objective function is designed in a way that the similarity constraint and individual preferences affect the result equally. To do so, distance value is divided by 100 to bring it to the same scale as the individual preferences violation penalty value.

**SCOP-KMeans** Runtime:

Let (Individual Preferences Evaluation) be a function that generates a set which consists of relationships with an additional *strength* factor, , that indicates how reliable the constraint is. The relationships that agent generates are in between agent and all other agents. Higher strength values indicate a stronger constraint. A constraint relationship  
 corresponds to a must-link, corresponds to a cannot-link constraint. A constraint  
 corresponds to “don’t care”.

where .

Each agent independently evaluates the relationship between itself and all other agents to decide the strength of the constraint based on its individual preferences.

For example, agent is free to generate if it highly wants to be clustered with . As well as, it can generate if it highly prefers to be in a different cluster than the cluster of .

Once all agents submit their individual constraints, there can be at most 2 relationship constraints which include same agents, one from each agent perspective. For example, (example.1) would be generated by and (example.2) would be generated by based on their individual preferences. Therefore, dedicated agent needs to combine double relationships such that there are only unique relationship constraints in terms of the constrained agents. To do so, dedicated agent can take the lower (0) or higher (0.8) value out of the two relationships in example.1 and example.2. However, to make the preferences of both agents are considered equally (fairly) in the final (unique) constraint relationship, dedicated agent calculates the average of two strength values. For *example.1* and *example.2*, . Consequently, dedicated agent generates and removes other relationships which contain .

As a result of refinement/recalculation process of constraint relationships, dedicated agent would have a set which consists of unique .

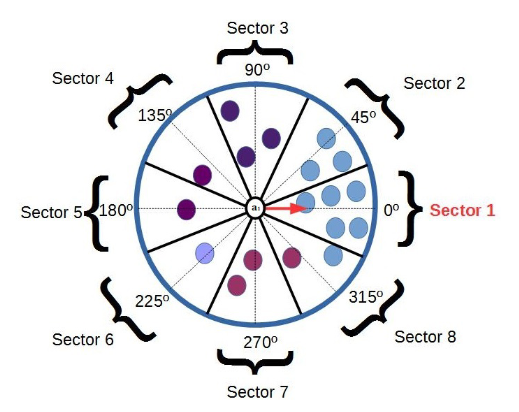
Decentralized Clustering in Multi Agent Systems

Techniques from two papers:

1. A Method for Decentralized Clustering in Large Multi-Agent Systems (Ogston, Steen, Brazier, & Overeinder, 2003)
2. A Multi-Agent-Based algorithm for data clustering (Adamatti & Emmendorfer, 2020)

The first technique “…*agents are each given a two-dimensional point and seek to group themselves based on the* ***Euclidean distance*** *between their* ***points****. Initially agents are randomly assigned a small number (5) of neighbor agents. These neighbors are an agent’s only view of the system as a whole. Based on these local views, agents form clusters with the closest points they come across. Agents within a cluster coordinate, combining their local views to allow each member to search a broader range of neighbors for better matches. Clusters are limited in size by a user-defined parameter. Once clusters have grown to this size they spilt when better matches are found by their members, allowing stronger new clusters to form.*”. Preferences and personal information are not simple numbers or points which can be used by agents to calculate the **Euclidean distance.**

The second technique “…*Each agent* ***a*** *senses its own environment, within a limited vision scope of radius* ***α*** *around its own current coordinates. This resulting circumference is then divided into* ***s*** *sectors equally distributed. At each step, the agent evaluates all s sectors around it, searching for the sector with the* ***highest number******of agents****. Then the agent changes its cluster label, assuming the most frequent label in the agents inside the chosen sector, and finally moves towards the direction given by that sector. The movement step is set to the smallest distance among all original data points in D.*” The technique is based on the amount of agents (density). Personal information and preferences cannot be illustrated in this technique.



**Fig**. Illustration of an agent vision radius.

LCC\_AVC algorithm

1. Each agent shares its personal info with Dedicated agent
2. Dedicated agent collects and broadcasts the complete personal info
3. Each agent constructs the info that there are n clusters for n agents and a\_i is in C\_i.
4. For each agent:
   1. It evaluates other clusters and selects the one it prefers most
   2. It informs other agents about its selection and other agents update their knowledge about the cluster membership

Each agent already has the complete personal info of others.

t=0

Repeat

t+=1

For each a in A:

Agent a evaluates the clusters and selects the one it prefers most for iteration t-1 (meaning that agent a must have info from all agents for the previous iteration)

Agent a broadcasts the cluster name for iteration t (or agent sends to a dedicated agent to prevent the broadcasting)

All agents update the cluster membership knowledge once they receive the complete info for iteration t (or dedicated agent collects the info for iteration t from all agents, updates the cluster membership and broadcasts)

Until no more changes in the cluster membership

Distributed clustering:

The idea is to prevent having a central dedicated agent as far as I know.

In AVC, each agent evaluates the sectors to select the most populated one at each step until there is no modification.

In LCC, each agent evaluates the existing clusters to select the most suitable one at each step until convergence.

ACS time complexity

ACS solution quality

Gain rate for ACS

Gain rate indicates the extent to which an algorithm has improved the solution compared to that of the singleton coalitions.

ADC (Qiao & Brown, 2019)

ACS (Unknown, 2021)

**References**

Adamatti, D., & Emmendorfer, L. (2020). A Multi-Agent-Based algorithm for data clustering. *Progress in Artificial Intelligence*.

Likas, A., & Tzortzis, G. (2008). The Global Kernel k-Means Clustering Algorithm. *IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence).* Hong Kong.

Ogston, E., Steen, M. v., Brazier, F., & Overeinder, B. (2003). A Method for Decentralized Clustering in Large Multi-Agent Systems. *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*.

Okabe, M., & Yamada, S. (2018). Clustering Using Boosted Constrained k-Means Algorithm. *Frontiers in Robotics and AI*.

Qiao, C., & Brown, K. (2019). Asynchronous Distributed Clustering Algorithm for Wireless Sensor Networks. *the 2019 4th International Conference*.

Unknown, U. (2021). Code-based Algorithm for Coalition Structure Generation. *Unknown*.

Wagstaff, K. L. (2002). Intelligent Clustering with Instance-Level Constraints. *PhD thesis,Department of Computer Science, Cornell University*.

Wagstaff, K., Claire, C., Seth, R., & Stefan, S. (2001). Constrained K-means Clustering with Background Knowledge. *Proceedings of the Eighteenth International Conference on Machine Learning*, 577-584.