An Improved Cop-Kmeans Clustering for Solving Constraint Violation Based on MapReduce Framework

Article *in* Fundamenta Informaticae · January 2013

DOI: 10.3233/FI-2013-883

Yan Yang
Southwest Jiaotong University
58 PUBLICATIONS **236** CITATIONS
SEE PROFILE

Tianrui Li
Southwest Jiaotong University
236 PUBLICATIONS **1,881** CITATIONS
SEE PROFILE

Fundamenta Informaticae 126 (2013) 301–318
DOI 10.3233/FI-2013-883
IOS Press

An Improved Cop-Kmeans Clustering for Solving Constraint Violation Based on MapReduce Framework

Yan Yang*[*] [†], Tonny Rutayisire, Chao Lin, Tianrui Li, Fei Teng
School of Information Science and Technology
Southwest Jiaotong University, Chengdu 610031, P.R. China
yyang@swjtu.edu.cn, rutantonio14@yahoo.com, linchao0916@126.com, {trli,fteng}@swjtu.edu.cn

Abstract. Clustering with pairwise constraints has received much attention in the clustering community recently. Particularly, *must-link* and *cannot-link* constraints between a given pair of instances in the data set are common prior knowledge incorporated in many clustering algorithms today. This approach has been shown to be successful in guiding a number of famous clustering algorithms towards more accurate results. However, recent work has also shown that the incorporation of *must-link* and *cannot-link* constraints makes clustering algorithms too much sensitive to "the assignment order of instances" and therefore results in consequent constraint violation. The major contributions of this paper are two folds. One is to address the issue of constraint violation in Cop-Kmeans by emphasizing a sequenced assignment of *cannot-link* instances after conducting a Breadth-First Search of the *cannot-link* set. The other is to reduce the computational complexity of Cop-Kmeans for massive data sets by adopting a MapReduce Framework. Experimental results show that our approach performs well on massive data sets while may overcome the problem of constraint violation.

Keywords: Semi-supervised clustering, Pairwise constraints, Constraint violation, Cop-Kmeans, MapReduce

[*]This work is supported by the National Science Foundation of China (Nos. 61170111, 61175047, 61134002 and 61202043), the Fundamental Research Funds for the Central Universities (Nos. SWJTU11ZT08 and SWJTU12CX098) and the Research Fund of Traction Power State Key Laboratory, Southwest Jiaotong University (No. 2012TPL-T15)
[†]Address for correspondence: School of Information Science and Technology, Southwest Jiaotong University, Chengdu, 610031, P.R. China

1. Introduction

Clustering is one of the core tasks in data mining, machine learning and pattern recognition. Traditionally, clustering algorithms have been designed to work in a purely unsupervised fashion [1–3]. They are availed with a set of data objects that must be automatically partitioned according to some general measure of similarity: most commonly distance. However, in most real application domains, it's often true that human experts possess specific background information about the domain and/or data set that can be handy in guiding the clustering process towards more accurate and meaningful clustering results. For example, when clustering GPS trace information for auto-mobiles to form traffic lanes [4], it was known that the width of a highway lane is 4 meters and therefore any two data points/cars more than 4 meters apart must be in different clusters/lanes. This form of background knowledge was successful in inspiring the clustering algorithm to produce the required (elongated) clusters. Whereas such specific prior knowledge might be available in most application domains, traditional clustering algorithms cannot take advantage of it. In the past decade, the problem of clustering with constraints has received much attention due to its potential to improve the quality of clustering results.

Existing methods for semi-supervised clustering can be generally divided into two groups. Constraint-based methods aim to guide the clustering process with pairwise constraints [4] or initialize cluster centroids by labeled examples [5]. In distance-based methods, an existing clustering algorithm that employs metric learning techniques to obtain an adaptive distance measure based on the given pairwise constraints [6]. In [7], Saha et al. studied a genetic semi-supervised clustering using point symmetry based distance measure. In [8], Wagstaff et al. introduced two instance-level pairwise constraints, the *must-link* (*ML*) and *cannot-link* (*CL*) that offer ability to incorporate strong background knowledge into clustering algorithms. The *must-link* denoted by $(x, y) \in ML$ indicates that two instances x and y must be in the same cluster, whereas *cannot-link* denoted by $(x, y) \in CL$ indicates that x and y cannot be in the same cluster. K-Means is one of the most famous clustering algorithms in data mining due to its simplicity [9]. Wagstaff et al. modified K-Means to Cop-Kmeans [4] by making use of *ML* and *CL* constraints. Their experimental results confirmed that Cop-Kmeans can remarkably improve the accuracy of clustering results. Recently Sun et al. presented a modified K-Means clustering algorithm based on the clustering problem with balancing constraints [10]. Schmidt et al. introduced a variant of the K-Medoids algorithm taking into account attribute-level constraints which shows how the well-established instance-level constraints *must-link* and *cannot-link* can be adapted to the attribute level [11].

Although clustering with pairwise constraints is generally proven to enhance the conventional K-Means, it often comes with a problem of constraint violation. This brings about failure in hard-constrained clustering algorithms like Cop-Kmeans [4] when an instance has got at least a single *cannot-link* in every cluster. Some work has been directed towards resolving this issue. Instead of assuming hard-constraints, PCKmeans [12] adopted the concept of soft-constraints where every constraint is associated with a cost of violating it. Wagstaff proposeed two variants of Cop-Kmeans for solving this problem [13]: the first is to restart the algorithm with random initial cluster centers whenever constraint violation occurs, while the other is to return the solution of the previous iteration before constraint violation is encountered. In [14], Tan et al. also suggested an improved version of Cop-Kmeans algorithm. On encountering constraint violation, their ICop-Kmeans would backtrack (revisit the assigned instances) and re-arrange some of the instances so that constraint violation could be solved. We find this solution computationally very expensive especially when dealing with massive data sets typically involved in data mining [15], while the solutions in [13] may result into repetitive constraint violations and inaccurate clusters. In our previous work, we discussed solving constraint violation by sequenced *cannot-link* set [16].

With the rapid development of information technology, data volumes processed by typical business applications are very high today which in turn pushes for high computational requirements. To our knowledge though, widely used semi-supervised clustering algorithms are serial and can only run on a single computer. This greatly hinders their capability to handle big data sets. To process such massive data, a highly efficient, parallel approach to clustering needs to be adopted. MapReduce is a parallel programming model for processing big data sets using large numbers of distributed computers (nodes), collectively known as a cluster [17–19]. Recently, several attempts have been made to improve the applicability of K-Means algorithm for massive applications through parallelization [20–24].

This article is an extended version of the paper presented at the RSKT'2011 conference [16]. Its major contributions are as follows. One is to address the issue of constraint violation in Cop-Kmeans by emphasizing a sequenced assignment of *cannot-link* instances after conducting a Breadth-First Search of the *cannot-link* set. The other is to reduce the computational complexities of Cop-Kmeans by adopting a MapReduce Framework. From the view of Breadth-First Search, the extended version discusses how to solve the constraint violation. The parallel CLC-Kmeans algorithm based on MapReduce in Section 3 is a new addition that is effective for big data sets' clustering.

The rest of the paper is organized as follows. Section 2 introduces basic concepts of the Cop-Kmeans algorithm and its problem of constraint violation. Also it gives insight in Breadth-First Search and further illustrates its role in solving constraint violation in CLC-Kmeans. Section 3 presents our parallel CLC-Kmeans algorithm based on the MapReduce framework. Section 4 provides the test results evaluating the performance of the proposed algorithm. Finally we draw our conclusions and future work in Section 5.

2. An Improved Cop-Kmeans Clustering Algorithm

2.1. The Shortcomings of Cop-Kmeans Algorithm

The Cop-Kmeans algorithm with pairwise constraints has been proven to enhance the conventional K-Means. It works as follows: firstly, the transitive closures are implemented to extend *must-link* and *cannot-link* constraints and K initial cluster centers are selected. Secondly, the object x_i is assigned to the nearest cluster centers under the premise of none of extended constraints being violated. Otherwise it will return an empty partition and the algorithm fails if any cluster cannot be found. Finally, every cluster center is updated and the algorithm iterates until convergence.

The Cop-Kmeans is often associated with a problem of constraint violation when an instance has got at least a single *cannot-link* in every cluster. From Figure 1 we find that Cop-Kmeans sometimes don't give a valid result [14]. Object (x_i, x_k) and (x_j, x_k) both belong to the *cannot-link* constraints whereas x_i and x_j have been assigned into cluster C_a and C_b before deciding the label of x_k. When it is the time to assign x_k, there is not any right cluster that satisfies all the *cannot-link* constraints and the algorithm terminates.

The underlying reason for the case of Figure 1 is the sensitivity of Cop-Kmeans. In Figure 1, the number of the clusters K is two, while $(x_i, x_k) \in$ *CL* and $(x_j, x_k) \in$ *CL*. It means that the right class labels of x_i and x_j must be the same. Due to the assignment of x_i and x_j before x_k, however, x_i and x_j are in the different clusters (C_a and C_b) at the moment. Therefore, the improper assignment order lets x_i and x_j separated and indirectly lead to the violation of *cannot-link* constraints for x_k.

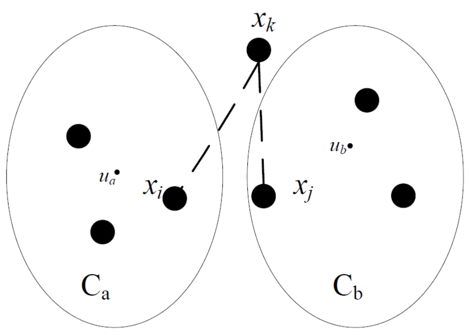

Figure 1. Failure of assignment for x_k

It must be noted that only the *cannot-link* constraints induce the failure of looking for a satisfying solution because the *must-link* constraints are transitive and not influenced by the assignment order. For example, $(x_i, x_k) \in$ *ML* and $(x_j, x_k) \in$ *ML* imply $(x_i, x_j) \in$ *ML* so that x_i and x_j must be in the same cluster before assigning x_k. However, $(x_i, x_k) \in$ *CL* and $(x_j, x_k) \in$ *CL* do not reveal the relationship between x_i and x_j so that the two objects are possible to be placed into the same cluster before assigning x_k. Therefore, a *cannot-link* constraint may be violated when x_k comes.

2.2. Principle Analysis

Generally, Cop-Kmeans changes the clustering problem from *"assigning each instance to the nearest cluster center"* to *"assigning each instance to the nearest* ***feasible*** *cluster center"*. However, when care is not taken while generating pairwise constraints, they may in one way or another contradict each other in such a way that no single clustering satisfies all constraints [24], *i.e.*, *no feasible solution*. For example, there can't be any possible clustering under constraints $\{(x, y) \in$ *ML* , $(x, y) \in$ *CL*$\}$ or $\{(x, z) \in$ *ML*, $(y, z) \in$ *ML*, $(x, y) \in$ *CL*)$\}$ regardless of the value of K (the number of clusters to be formed). Moreover, for even non-directly contradicting constraints such as $\{(x, y) \in$ *CL*, $(x, z) \in$ *CL*, $(y, z) \in$ *CL*$\}$, there is no feasible clustering for $K \leq 2$. Previous work [9, 26] has discussed this problem quite extensively.

Definition 2.1. (Feasibility Problem) [26]: Given a data set X, a collection of constraints C, a lower bound K_l and an upper bound K_u on the number of clusters, does there exists a partition of X into K groups such that $K_l \leq K \leq K_u$ and all the constraints in C are satisfied?

Theorem 2.1. (Brooks's Theorem) [27]: If the maximum node degree of the graph G is Δ and $K \geq \Delta + 1$, then the K-coloring problem is easy/feasible and such a coloring can be obtained using any linear ordering of nodes.

The problem of clustering under CL constraints is often interpreted as graph coloring where Cop-Kmeans fashion of clustering under constraints is viewed as a greedy type of graph coloring that finds a feasible solution while minimizing the algorithm's objective function. Brooks's theorem provides a sufficient condition to determine whether the Cop-Kmeans will always converge or not.

Basically it interprets that if the maximum number of CL constraints involving one instance is less than K, there will always be a feasible solution regardless of the order in which the instances are assigned to the clusters.

Definition 2.2. (Inductiveness of graphs) [26]: Let q be a positive integer. An undirected graph $G(V, E)$ is Q-inductive if the nodes of G can be assigned distinct integer values in such a way that each node has an edge to at most Q nodes with higher assigned values.

Theorem 2.2. [26] Suppose $G(V, E)$ is Q-inductive. G can be colored using at most $Q + 1$ color.

Definition 2.3. [26] A feasibility problem instance is β-easy if a feasible solution can be found by an algorithm β given the ordering of instances in the training set which determines the order they will be assigned to clusters.

The concept of *inductiveness of graphs* defined above, suggests that despite the Brooks's theorem, there could still be scenarios where there is a feasible solution, even when the maximum number of CL-constraints involving one instance is not necessarily less than K.

However, it is also true and often occurs that, much as there could be a feasible solution, it does not mean it will be easy to find. In [4], Wagstaff et al. showed that even when $K = K^*$ (therefore guaranteeing a feasible solution), Cop-Kmeans occasionally fails. It is important to note that in Cop-Kmeans, unlike the original K-Means, the assignment of instances to clusters is order-sensitive. Therefore, if a poor decision (about assignment order of instances) is made early on, the algorithm may later encounter an instance x_i that has a *cannot-link* constraint to at least one item in each of the K clusters (constraint violation).

2.3. Breadth-First Search

Breadth-First Search (BFS) is a general technique for traversing a graph whose principal is "going broadwise (in breadth) while there is such possibility, otherwise forward". The mechanism is about beginning at a root vertex and inspecting all the neighboring vertices. Then for each of those neighbor vertices in turn, it inspects their neighbor vertices which were unvisited, and so on.

The BFS algorithm uses a FIFO (*i.e.*, First In, First Out) queue (or linked list) data structure to store intermediate results. As it traverses the graph, all child vertices obtained by expanding a vertex are added to a queue. Algorithm 1 illustrates the process of BFS.

Algorithm 1: Breadth-First Search

```
(1) Enqueue the root vertex
(2) Dequeue a vertex and examine it
  (a) If the element sought is found in this vertex, quit the search and
        return a result
  (b) Otherwise enqueue any successors (the direct child vertices) that
        have not yet been visited
(3) If the queue is empty, every vertex on the graph has been examined-
      quit the search and return "not found"
(4) If the queue is not empty, repeat from Step (2)
```

In Figure 2, BFS starts at a given vertex A, which is at level 1 and enqueued. In the first stage, we visit all the vertices that are at the distance of one edge away, *e.g.*, B, C and D. When we visit there, we paint as "visited" the vertices adjacent to the start vertex A - these vertices are marked level 2 and in turn placed into queue. In the second stage, we visit all the new vertices (*e.g.*, E, F) we can reach at the distance of two edges away from the source vertex A. The BFS traversal terminates when every vertex has been visited.

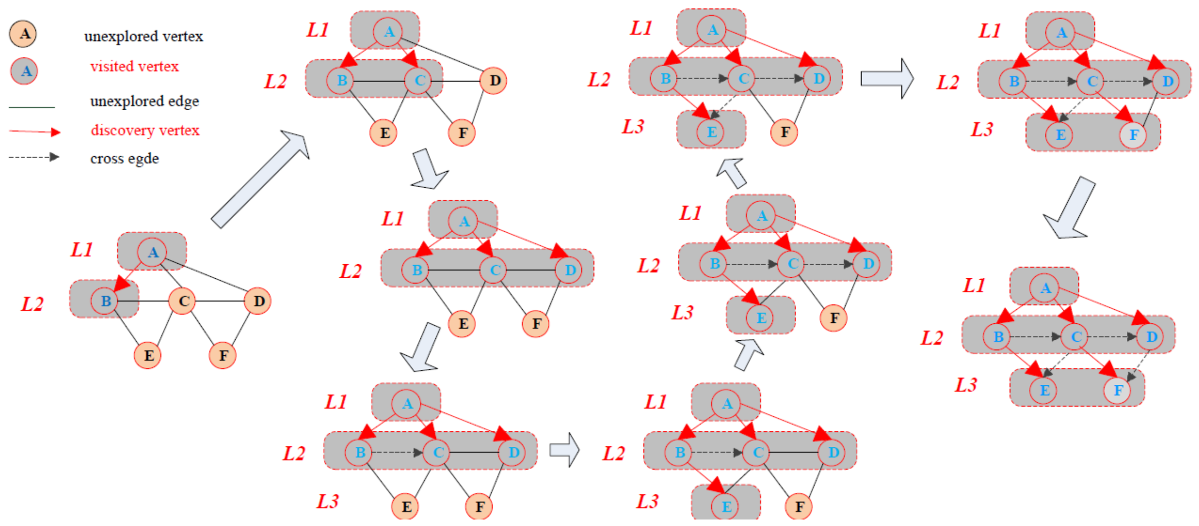

Figure 2. Breadth-First Search

2.4. The Improved CLC-Kmeans Algorithm for Solving Constraint Violation

Given a data set X, a set of *must-link* constraints *ML*, a set of *cannot-link* constraints *CL* and the number of clusters to form K, we proposed an improved Cop-Kmeans algorithm called CLC-Kmeans (Cannot-Link Cop-Kmeans) based on sequencing of the *cannot-link* set [16]. The algorithm returns a partition of instances in X, satisfying all constraints and without failures resulting from constraint violation (*i.e.* never returns an empty partition). Algorithm 2 shows the details of the improved Cop-Kmeans algorithm with major modifications.

The novelty of our algorithm is two folds: (a) For every iteration, instances are assigned to clusters in a purely random order, and this holds unless a *cannot-link* instance is encountered. (b) When a *cannot-link* instance, *e.g.*, *CL* 1 is encountered, the algorithm creates an empty linked list, inserts *CL* 1 at the beginning of the linked list, scans the entire data set for all instances in *CL* with it, and then inserts them in the list after *CL* 1 towards the end of the linked list. For every *cannot-link* instance added to the linked list, scan the data set for all instances in *CL* with it (not already in the linked list) and insert them in the linked list one by one starting from the next free space available in that sequence until all the *CL* instances are in the linked list. Then the *CL* instances are assigned to clusters (as per Cop-Kmeans mechanism), but in the order produced by the linked list.

Algorithm 2: CLC-Kmeans Algorithm

```
Input: data set $X$, number of clusters $K$, must-link constraints ML,
cannot-link constraints CL, maximum iteration time $T$
Output: clusters $C_1, ..., C_K$
(1) Initialize $K$ cluster centers $C_{t1}, ..., C_{tK}$
(2) For every randomly selected object $x_i$ in $X$
    (a) If $x_i$ is a CL-instance, then Process-list ($x_i$, CL) is called
    (b) Else assign it to the nearest cluster $C_j$ such that
        Violate-constraints ($x_i, C_j$, ML, CL) returns false. If no such
        cluster is found, return $\{\}$
(3) Update every cluster center $C_{t1}, ..., C_{tK}$ by calculating the
    mean of all objects that have been assigned to it
(4) Repeat step (2) and (3) until convergence or the iteration time
    comes to $T$
(5) Return clusters $C_1, ..., C_K$

Process-list $(x_i, CL)$
(1) Create an empty linked list $L$
(2) Insert $x_i$ into $L$
(3) For every newly inserted instance $x_{li}$ i.e., beginning with $x_i$
    (a) Scan the data set $X$ for all instances in CL with $x_{li}$ except
        those already in $L$
    (b) Insert them in $L$, occupying the next free spaces towards the end
(4)For every instance $L_i$ in $L$ (left to right), assign it to the nearest
    cluster $C_j$ such that Violate-constraints ($x_i, C_j$, ML, CL) returns
    false. If no such cluster is found, return $\{\}$

Violate-constraints (object $x_i$, cluster $C$, must-link constraints ML,
cannot-link constraints CL)
(1) If $x_c$ is already in cluster C and $(x_i, x_c)$ are in CL, return true
(2) If $x_m$ is already assigned to another cluster other than $C$ and
    $(x_i, x_m)$ in ML, return true
(3) Otherwise, return false
```

Note that instances in the linked list will be sequenced in such a way that each of them would have at most $K - 1$ instances in *CL* with it to be assigned before it, implying that it would be left with at least one possible cluster for assignment. Also note that this algorithm will create a differently ordered list for every iteration depending on the first *CL*-instance to be encountered.

Our algorithm is greatly inspired by the work in [26], where the authors highlighted the importance of the ordering of instances in the data set in most iterative clustering algorithm of which Cop-Kmeans is among. CLC-Kmeans therefore capitalizes on re-arranging the *CL* instances before assignment, in such a way that will prevent constraint violation to happen while assigning instances to clusters.

To further illustrate the operation of our algorithm, we consider a set of *CL*-constraints; $\{(y_1, y_2) \in$ *CL*, $(y_1, y_3) \in$ *CL*, $(y_2, y_3) \in$ *CL*, $(y_2, y_5) \in$ *CL*, $(y_3, y_5) \in$ *CL*$\}$. In this set the maximum number of *CL*-Constraints involving a single instance, Δ, is 3 and therefore, according to Brooks's Theorem, this clustering problem would have a feasible solution when $K \geq \Delta + 1$, *i.e.*, $K \geq 4$ regardless of the ordering. However, Definitions 2, 3 and Theorem 2 show that this clustering problem could be interpreted as a q-inductive graph $(q = 2)$ which can be colored/clustered using as few as $q + 1$ clusters instead of $K \geq \Delta + 1$, depending on the ordering.

Figure 3 depicts possible assignment orderings of instances with Cop-Kmeans in (a) and CLC-Kmeans in (b). In Cop-Kmeans, since the assignment of instance does not follow a pre-defined sequence, it is possible to follow y_1, y_5, y_3, y_2 such as in (a). In this way, if y_1, y_5 and y_3 each is assigned to a different cluster, then y_2 will not have a feasible cluster and this will cause a failure in Cop-Kmeans. However, using CLC-Kmeans by BFS and beginning with any of the instances, the set is sequenced such that at any point of assignment, the instance will have at most two instances in *CL* with it, already assigned, thereby at least remaining with one feasible cluster. One of such sequences is y_1, y_3, y_2, y_5 in (b).

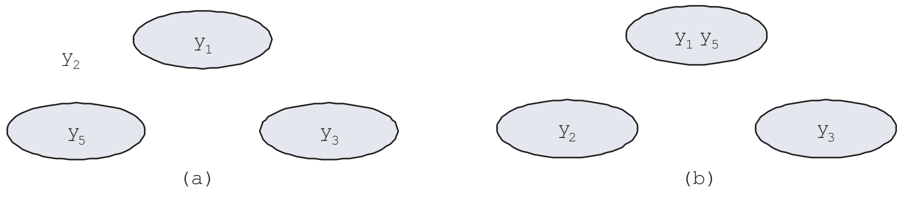

Figure 3. Possible assignment orderings with Cop-Kmeans and CLC-Kmeans algorithms

3. A Parallel CLC-Kmeans Algorithm Based on MapReduce

In this section, we present the proposed parallel CLC-Kmeans algorithm, but prior to that, we introduce the underlying mechanism of the core approach used in this algorithm namely: MapReduce.

3.1. The MapReduce Framework

MapReduce is a programming model introduced by Google to support distributed computing of large data sets on clusters of computers. It has been used in a wide range of applications including: distributed pattern-based searching, distributed sort, web link-graph reversal, term-vector per host, web access log stats, and document clustering. Even the programmers with no experience on distributed systems can easily utilize the resources of a large distributed system, by simply specifying “Map” and “Reduce” functions. The Map function processes the input in the form of key/value pairs to generate intermediate key/value pairs, and the Reduce function processes all intermediate values associated with the same intermediate key generated by the Map function. These distributed Map and Reduction functions can be processed in parallel in MapReduce framework automatically. MapReduce data flow with two reduce tasks is illustrated in Figure 4 [28].

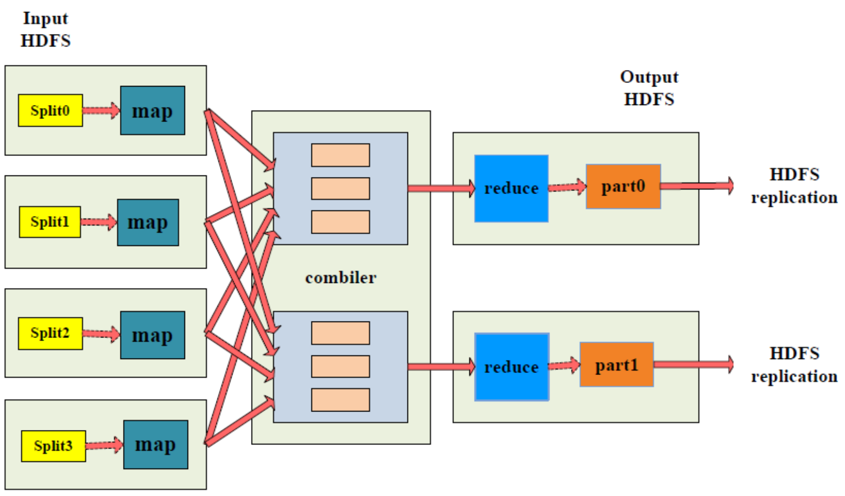

Figure 4. The MapReduce framework

As shown in Figure 4, input data is stored in a distributed file system, such as HDFS. The Map functions are distributed across multiple machines by automatically partitioning the input data into a set of several splits. These splits can be processed in parallel on different machines. Map functions produce intermediate key/value pairs buffered in memory. These intermediate key/value pairs are then categorized in the combiner so that all occurrences of the same key are grouped together, which are used as the input of Reduce. Every unique key and the corresponding set of intermediate values are parsed to individual Reduce function. The output of the Reduce function is appended to a final output file, and will be written back to HDFS.

3.2. An Improved CLC-Kmeans Algorithm Based on MapReduce

The underlying idea behind the original Cop-Kmeans algorithm is to assign each instance in the data set to the nearest feasible cluster center. We can see that the algorithm involves typically three steps: calculating the shortest distance between each instance and the cluster centers, assigning instance while avoiding constraint violation and updating cluster centers. From the serial point of view, all these steps are handled by a uniprocessor and all the concerned data is kept in its local memory. However, in order to incorporate these tasks in MapReduce framework, some crucial modifications must be made.

Map Function. Ideally, since the task of distance computations is so bulky and would be independent of each other, it is reasonable to execute it in parallel by the Map function. The challenge about this is that the next task of assigning the instance to the cluster center with which it has the shortest distance takes into consideration constraint violation. This would mean dependence on assignment of instances in other Mappers which could have a *ML* or *CL* with the instance in question. To overcome this issue and simplify, we generate constraints from the partial data set allocated to each Mapper. This would ensure that distance computations and considerations of *ML* and *CL* are purely independent of other Mappers. This Map function outputs intermediate data to be used in the Reduce function.

Algorithm 3 shows the Map function of our proposed parallel CLC-Kmeans [24].

Algorithm 3: Map Function

```
Input: partial data set $X$ (Key, Value), number of global initial centers $K$
Output: (Key, Value) pairs where the Value is the instance information,
while the Key represents its cluster label
(1) Generate ML and CL constraints from the partial data set $X$
(2) For every randomly selected instance $x_i$ in $X$
   (a) If $x_i$ is a CL-instance, create an empty linked list $L$, then
      Breadth-First Search($x_i$, CL) is called (see Algorithm 1)
   (b) Else assign it to the nearest cluster $C_j$ such that
      Violate-constraints($x_i, C_j$, ML, CL) (see Algorithm 2) returns false.
      If no such cluster is found, return $\{\}$
(3) Take index of closest cluster as Key
(4) Take the instance information as Value
(5) Return (Key, Value) pair
```

Reduce Function. The Reduce Function gets its input from the Map Function of each Mapper (host). As shown in Algorithm 3 above, in every Mapper, the Map function outputs a list of instances with each labeled. Therefore it follows that, in each Mapper, all instances labeled with same current cluster are sent to a single Reducer. In the Reduce function, the new cluster center can be easily computed by averaging all the instances in the Reducer. Algorithm 4 details the Reduce Function of the parallel CLC-Kmeans [24].

As shown in Algorithm 4, the output of each Reducer is consequently the cluster label (Key) and its new center coordinates (Value). The new centers are then fed back to the Mappers for the next iteration and this goes on until convergence.

3.3. The Analysis of Computational Complexity

Given a set of objects $(x_1, x_2, , x_N)$, where each object is a d-dimensional real vector, the parallel K-Means clustering aims to partition the N objects into K clusters $(K \leq N)$. So there are K centers totally. *ML* and *CL* represent the number of *must-link* and *cannot-link*, respectively. Maximum iteration time is T.

Algorithm 4: Reduce Function (Key, L)

```
Input: Key is the index of a particular cluster, $L$ is the list of instances
assigned to the cluster from different Mappers
Output: (Key, Value) where Key is the label of the cluster, Value is the
information representing the new center
(1) Compute the sum of values for each dimension of the instances
      assigned to the particular cluster
(2) Initialize the counter to record the number of instances assigned
      to the cluster
(3) Compute the average of values of the instances assigned
      to the cluster to get coordinates of the new center
(4) Take coordinates of new cluster center as Value
(5) Return (Key, Value) pair
```

For the classical K-Means clustering, there are 3 stages in each round. Firstly, the distance calculation between objects and centers spends the most execution time $O(NK)$. Secondly, assigning an object to the closest center spends $O(K)$. Thirdly, updating the new centers spends $O(N)$. Assume the K-Means algorithm repeats T times ($T > 1$) on average to reach convergence. Therefore, the iteration of three stages spends $O(TNK) + O(TK) + O(TN)$. Considering $K << N$, the time complexity of K-Means clustering algorithm is $O(TNK)$.

Assume the MapReduce cluster contains P nodes that can process in parallel. For our proposed parallel CLC-Kmeans algorithm with MapReduce, the first and second stages are realized in Map phase, while the third stage is realized in Reduce stage. The objects in data set can be categorized into three classes, normal objects, objects with *must-link* and objects with *cannot-link*. For each normal object, time complexity of Map phase is $O((N - M - C) * K/P) + O(K)$. An object with a *must-link* can directly find its center, so it costs nothing in Map phase. Besides, an object with a *cannot-link* only need calculate the distance with other $(K - 1)$ centers, so the corresponding time complexity of Map phase is $O(C * (K - 1)/P) + O(K - 1)$. The time complexity of Reduce phase is still $O(N)$. If *ML* and *CL* is negligible, the time complexity of parallel CLC-Kmeans clustering is $O(TNK/P)$. Obviously, the time cost decreases linearly when the number of parallel nodes increases. If the supervision information is abundant, the computational complexity of T MapReduce interations is $O(T(NK - MK + C)/P)$.

4. Experimental Results and Analysis

4.1. Data Sets

To evaluate the performance of the improved CLC-Kmeans, we compare it with original Cop-Kmeans with respect to their proportions of failure (constraint violation) and F-measures. For this purpose, we use four well-known numerical data sets from UCI Machine Learning Repository [29] namely: Iris

(150, 4, 3), Wine (178, 13, 3), Balance-scale (200, 4, 3) and Sonar (208, 60, 2). The figures in the brackets indicate for each data set; the number of instances, number of attributes and number of classes respectively.

In the parallel experiments, we evaluate the efficiency of the proposed parallel CLC-Kmeans based on MapReduce framework. We use data sets Covertype, Shuttle and MinibooNE from UCI [29] described in Table 1. Also we use larger data set USCensus1990 (823MB) which owns 68 categorical attributes. Many of the less useful attributes in the original data set have been dropped, the few continuous variables have been discretized and the few discrete variables that have a large number of possible values have been collapsed to have fewer possible values. For comparison purposes, we divided the data set into 4 groups of files, approximately having 200MB, 400MB, 600MB and 800MB in that order.

Table 1. A description of data sets

Data set	Attributes	Classes	Instances
Covertype	54	7	581012
Shuttle	9	5	58000
MinibooNE	50	2	130065

4.2. Performance Measure

The results of any clustering algorithm should be evaluated using an informative quality measure that reflects the "goodness" of the resulting clusters. There are three types of evaluation, called external quality measure, internal quality measure or relative quality measure, depending on whether the data sets have prior knowledge or not [30]. An external evaluation criterion called the F-measure is used here.

The F-measure combines the ideas of precision and recall from the information retrieval literature. The precision and recall of a cluster j with respect to a class i are defined as:

$$precision(i,j) = n_{ij}/n_j \tag{1}$$

$$recall(i,j) = n_{ij}/n_i \tag{2}$$

where n_{ij} is the number of class i in cluster j, n_j is the number of members of cluster j and n_i is the number of members of class i.

The F-measure of cluster j and class i is then given by

$$F(i,j) = \frac{2 \times precision(i,j) \times recall(i,j)}{precision(i,j) + recall(i,j)}. \tag{3}$$

With respect to class i we consider the cluster with the highest F-measure to be the cluster j that maps to class i, and that F-measure becomes the score for class i. The overall F-measure for the clustering result is computed by taking the weighted average of all values for the F-measure as given by

$$F = \sum_{i=1}^{k} \frac{n_i}{k} max\{F(i,j)\} \tag{4}$$

where k is the total number of clusters.

To validate the MapReduce framework, we use Speedup, Sizeup and Scaleup measure [22, 31]. Speedup of a parallel system is defined by the following formula:

$$Speedup(p) = T_1/T_p \tag{5}$$

where p is the number of nodes, T_1 is the execution time on one node and T_p is the execution time on p nodes. To measure the Speedup of a parallel system, we keep the data set constant while increasing the number of nodes in the system. In the experiment, we compare the Speedup performances produced when data set of varying sizes are given as inputs.

On the other hand, the Sizeup metric is defined by:

$$Sizeup(D, p) = T_{Sp}/T_{S1} \tag{6}$$

where D is the size of the data set, p is the multiplying factor of the data set, T_{Sp} is the execution time for $p * D$, T_{S1} is the execution time for D. To measure the Sizeup of a system, we keep the number of nodes in the system constant while growing the size of the data set by p. For our experiment, we compare the Sizeup performances produced by fixing the number of nodes in the system to 1, 2, 3, and 4.

At last, the Scaleup metric is:

$$Scaleup(D, p) = T_1/T_{Dp} \tag{7}$$

where T_{Dp} is the execution time for $p * D$ on p nodes. To measure the Scaleup, we grow both the number of nodes and the size of the data set. Scaleup is defined as the ability of a p-times larger system to perform a p-times larger data set in the same run-time as the original system.

4.3. Results and Analysis

To measure the usefulness of our approach in solving “constraint violation”, we compare both Cop-Kmeans and CLC-Kmeans basing on percentage of failure. Note that both algorithms are designed in such a way that they will fail whenever constraint violation arises. In this set of experiments, for a given number of constraints, each of the two algorithms is run 300 times on a data set. With increasing number of constraints (as inputs), we generate and report the corresponding percentage of failure (see Figure 5).

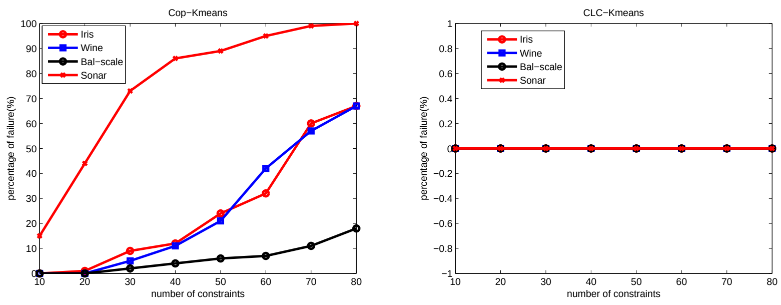

Figure 5. Percentage of Failure in Cop-Kmeans and CLC-Kmeans Algorithms

As shown in Figure 5, we can see an increasing trend of failure in Cop-Kmeans as the number of constraints increases. The worst scenario of this being depicted by the sonar data set *K=2* with 100%

failure when only 80 constraints are given as input. On the other hand, CLC-Kmeans shows no single case of failure in all the runs. In our experiments, it is observed that much as the constraints are generated randomly, the number of *CL* constraints tends to be greater than that of *ML* constraints. Therefore, as the number of constraints becomes large, the number of *CL* constraints becomes much greater than the *ML* constraints. Hence the chances of any instance being part of K or more than K *CL* constraints become high. The situation is also worsened by the fact that since the overall number of *ML* constraints is small, many instances with the same extrinsic label don't get the chance to combine and form a single connected component. Since Cop-Kmeans lacks the assignment mechanism which avoids constraint violation, it is more likely to suffer greater chances of failure.

The second set of experiments is aimed at showing whether or not the modifications we made have got any negative effects on Cop-Kmeans algorithm in terms of clustering accuracy. For this purpose, we choose F-measure to evaluate and then compare the clustering accuracy of both algorithms (see Figure 6). The results at each point on the curve are obtained by averaging the F-measures over the 300 runs for a given number of constraints. Note that in both algorithms, initial cluster centers and all pairwise constraints are generated randomly from the data sets.

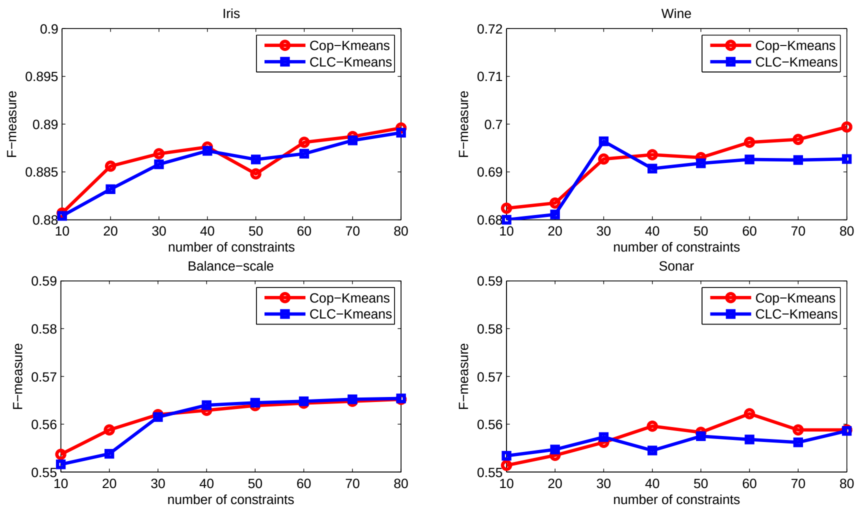

Figure 6. Average F-measures of Cop-Kmeans and CLC-Kmeans Algorithms

Figure 6 shows comparisons of Cop-Kmeans and CLC-Kmeans based on average F-measure. Ideally, since CLC-Kmeans algorithm puts much emphasis on sequencing *CL* instances in a way that would avoid constraint violation, it is reasonable to assume that this is achieved at some expense of clustering accuracy. However from our results, it can be observed that the F-measure of both algorithms is not much different, in all the data sets except in sonar. It can also be observed that (for Cop-Kmeans) there is no F-measure result obtained for sonar data set when the number of constraints is 80, since the algorithm fails for all runs.

Table 2 shows the runtime of both ICop-Kmeans and CLC-Kmeans for 500 times. Because ICop-Kmeans algorithm [14] was also proposed to solve constraint violation, it is important to note that it accomplishes the task at a much greater computational time than CLC-Kmeans, as shown in Table 2 for the three common data sets used.

Table 2. Runtime comparisons of ICop-Kmeans & CLC-Kmeans (seconds)

No. of Constraints	Algorithm	Iris	Wine	Sonar
20	ICOP-Kmeans	54	55	59
	CLC-Kmeans	13	17	28
40	ICOP-Kmeans	115	120	116
	CLC-Kmeans	21	30	33
60	ICOP-Kmeans	211	219	219
	CLC-Kmeans	30	51	67
80	ICOP-Kmeans	361	358	325
	CLC-Kmeans	54	73	74

In the third part of the experiments, we compute the running time on different nodes vary from 1 to 4 with 3 large data sets (Table 1). And then we evaluate the efficiency of the proposed parallel CLC-Kmeans with respect to Speedup, Sizeup and Scaleup on USCensus1990 data set. Note that the idea behind this algorithm is to intelligently distribute the computational workload across a cluster of computer nodes. In this set of experiment, we don't evaluate accuracy but rather efficiency of the parallel CLC-Kmeans in processing massive data set. The experiments are run on Hadoop MapReduce platform of four nodes, each having 2.13 GHz of processing power and 2GB of memory.

The running time of parallel CLC-Kmeans algorithm on Covertype, Shuttle and MinibooNE data sets are shown in Figure 7. It is observed that the running time is decreasing as the number of nodes increasing.

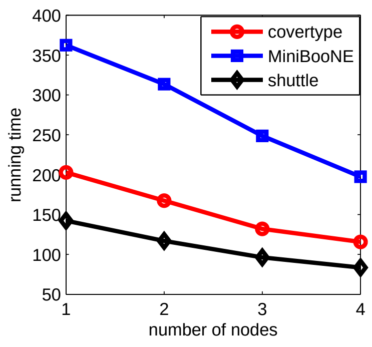

Figure 7. The running time of parallel CLC-Kmeans algorithm for three large data sets

Figure 8 reports the Speedup, Sizeup and Scaleup measures for our proposed parallel CLC-Kmeans Algorithm. As shown in the Figure 8 (a), for each data set, we increase the number of nodes in the system while reporting corresponding Speedup for that data set. It can be noted that the results show a good Speedup performance for our proposed parallel CLC-Kmeans. Furthermore, it can be noted that as the size of the data set increases, also Speedup performance increases; an indication that indeed our parallel algorithm can efficiently handle massive data sets. Sizeup evaluation also shows good performance of the parallel CLC-Kmeans. From the results (see Figure 8 (b)), we can observe that as we increase the number of nodes in the system, we get better results for Sizeup. As shown in the Figure 8 (c), we have performed Scaleup experiments where we have increased the size of the data sets in direct proportion to the number of nodes in the system. The data sets size of 200MB, 400MB, 600MB and 800MB are executed on 1, 2, 3 and 4 nodes, respectively. Clearly, the CLC-Kmeans algorithm handles larger data sets very well when more nodes are available.

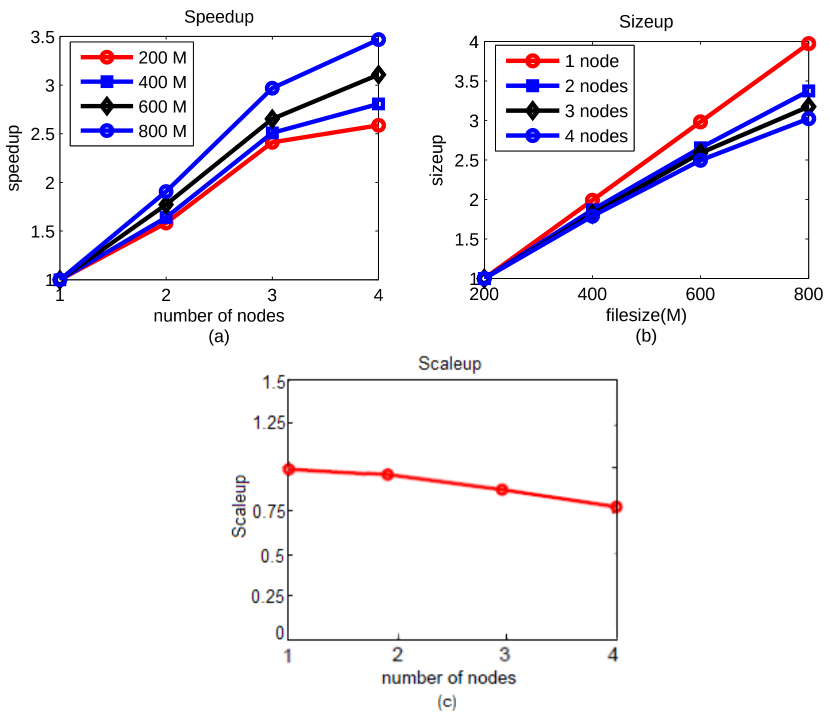

Figure 8. Speedup, Sizeup and Scaleup evaluations of parallel CLC-Kmeans algorithm

5. Conclusions

In this paper, we proposed a parallel CLC-Kmeans Algorithm based on MapReduce framework. This Algorithm adopted two crucial mechanisms: Breadth-First Search and MapReduce. Inspired by the sensitivity to assignment order of instances, BFS capitalizes on sequencing CL-instances in a way that will not allow constraint violation to happen. Results from our experiments confirmed that this improvement enhanced the accuracy of Cop-Kmeans without a single case of failure. Parallelizing the algorithm on a MapReduce framework also gave positive results in term of speeding up, sizing up and scaling up the computational processes. Hence our proposed algorithm confirmed efficient applicability to massive data sets typical to today's business applications.

References

[1] Davidson, I., Basu, S.:A survey of clustering with instance level constraints, *ACM Transactions on Knowledge Discovery on Data*, 2007, 1-41.

[2] Wang G., Wang Y.: 3DM: Domain-oriented Data-driven Data Mining, *Fundamenta Informaticae*, **90**, 2009, 395-426.

[3] Yu J., Yang M., Hao P.: A Novel Multimodal Probability Model for Cluster Analysis, *Fundamenta Informaticae*, **111**, 2011, 81-90.

[4] Wagstaff, K., Cardie, C., Rogers, S., Schroedl, S.: Constrained K-means clustering with background knowledge, *Proc. International Conference on Machine Learning*, 2001, 577-584.

[5] Basu S., Banerjee A., Mooney R.J.: Semi-Supervised clustering by seeding, *Proc. 19th International Conference on Machine Learning*, 2002, 19-26.

[6] Xing E.P., Ng A.Y., Jordan M.I., et al.: Distance metric learning, with application to clustering with side-information, *Advances in Neural Information Processing Systems*, 2003, 505-512.

[7] Saha. S., Bandyopadhyay S.: Semi-GAPS: A Semi-supervised Clustering Method, *Fundamenta Informaticae*, **96**, 2009, 195C209.

[8] Wagstaff, K., Cardie, C.: Clustering with instance level constraints, *Proc. International Conference on Machine Learning*, 2000, 1103-1110.

[9] Davidson, I., Ravi, S.S.: Clustering with constraints: Feasibility issues and the k-means algorithm, *Proc. SIAM International Conference on Data Mining*, 2005, 138-149.

[10] Sun Y., Liu M., Wu C.: A Modified K-means Algorithm for Clustering Problem with Balancing Constraints, *Proc. 3rd International Conference on Measuring Technology and Mechatronics Automation*, 2011, 127-130.

[11] Schmidt J., Brandle E. M., Kramer S.: Clustering with Attribute-Level Constraints, *Proc. 11th IEEE International Conference on Data Mining*, 2011, 1206-1211.

[12] Basu, S., Banerjee, A., Mooney, R.J.: Active semi-supervision for pairwise constrained clustering, *Proc. SIAM International Conference on Data Mining*, 2004, 333-344.

[13] Wagstaff, K.: *Intelligent clustering with instance-level constraints*, Cornell University, 2002.

[14] Tan, W., Yang, Y., Li, T.: An improved COP-KMeans algorithm for solving constraint violation, *Proc. International FLINS Conference on Foundations and Applications of Computational Intelligence*, 2010, 690-696.

[15] Anthony, K., Han, J., Raymond, T.: Constraint-Based Clustering in Large Databases, *Proc. International Conference on Database Theory*, 2001, 405-419.

[16] Rutayisire T., Yang Y., Lin C., Zhang J.: A Modified Cop-Kmeans Algorithm Based on Sequenced Cannot-Link Set, *Proc. 6th International Conference on Rough Sets and Knowledge Technology (RSKT2011)*, LNAI 6954, Springer, 2011, 217-225.

[17] Dean J., Ghemawat S.: MapReduce: Simplified Data Processing on Large Clusters, *Communications of the ACM*, **51**(1), 2008, 107-113.

[18] Xuan, W.: *Clustering in the Cloud: Clustering Algorithm Adoption to Hadoop Map/Reduce Framework*, Technical Reports-Computer Science 19, 2010.

[19] Yang Y., Chen Z.: Parallelized Computing of Attribute Core Based on Rough Set Theory and MapReduce, *Proc. 7th International Conference on Rough Sets and Knowledge Technology (RSKT2012)*, LNAI 7414, Springer, 2012, 167-172.

[20] Malay, K.: Clustering Large Databases in Distributed Environment, *Proc. International Advanced Computing Conference*, 2009, 351-358.

[21] Zhang, Y., Xiong, Z., Mao, J.: The Study of Parallel K-Means Algorithm, *Proc. World Congress on Intelligent Control and Automation*, 2006, 5868-5871.

[22] Zhao, W., Ma, H., He, Q.: Parallel K-Means Clustering Based on MapReduce, *Proc. CloudCom*, LNCS5931, 2009, 674-679.

[23] Li H., Wu G., et al.: K-Means Clustering with Bagging and MapReduce, *Proc. 44th Hawaii International Conference on System Sciences*, 2011.

[24] Lin C., Yang Y., Rutayisire T.: A Parallel Cop-Kmeans Clustering Algorithm Based on MapReduce Framework, *Proc. 6th International Conference on Intelligent Systems & Knowledge Engineering*, 2011, 93-102.

[25] Haichao, H., Yong, C., Ruilian, Z.: A semi-supervised clustering algorithm based on must-link set, *Proc. International conference on Advanced Data Mining & Applications*, 2008, 492-499.

[26] Davidson, I., Ravi, S.S.: Identifying and Generating easy sets of constraints for clustering, *Proc. American Association for Artificial Intelligence*, 2006, 336- 341.

[27] West, D.B.: *Introduction to Graph Theory, Prentice Hall*, Inc., Englewood Cliffs, NJ, 2001.

[28] Wbite T.: Hadoop: *The Definitive Guide, O'Reilly Media*, Inc., Second Edition, 2011.

[29] Machine Learning Repository, http://archive.ics.uci.edu/ml/datasets.html

[30] Yang Y., Kamel M.: An Aggregated Clustering Approach Using Multi-Ant Colonies Algorithms. *Pattern Recognition*, **39**(7), 2006, 1278-1289.

[31] Xu, X., Jager, J., Kriegel, H.P.: A Fast Parallel Clustering Algorithm for Large Spatial Databases. *Data Mining and Knowledge Discovery*, **3**, 1999, 263-290.