Computer Science and Information Systems 16(2):469–489 https://doi.org/10.2298/CSIS180601007L

Density-Based Clustering with Constraints

Piotr Lasek[1] and Jarek Gryz[2]

[1] University of Rzeszow, Poland
lasek@ur.edu.pl
[2] York University, Canada
jarek@cse.yorku.ca

Abstract. In this paper we present our *ic-NBC* and *ic-DBSCAN* algorithms for data clustering with constraints. The algorithms are based on density-based clustering algorithms *NBC* and *DBSCAN* but allow users to incorporate background knowledge into the process of clustering by means of instance constraints. The knowledge about anticipated groups can be applied by specifying the so-called *must-link* and *cannot-link* relationships between objects or points. These relationships are then incorporated into the clustering process. In the proposed algorithms this is achieved by properly merging resulting clusters and introducing a new notion of deferred points which are temporarily excluded from clustering and assigned to clusters based on their involvement in *cannot-link* relationships. To examine the algorithms, we have carried out a number of experiments. We used benchmark data sets and tested the efficiency and quality of the results. We have also measured the efficiency of the algorithms against their original versions. The experiments prove that the introduction of instance constraints improves the quality of both algorithms. The efficiency is only insignificantly reduced and is due to extra computation related to the introduced constraints.

Keywords: data mining, data clustering, semi-supervised clustering, clustering with constraints, instance-level constraints

1. Introduction

Clustering is a well-known and often used data mining technique. Its goal is to assign data objects (or points) to different clusters so that objects that are assigned to the same cluster are more similar to each other than to objects assigned to other clusters [10].

Clustering algorithms can operate on different types of data sources such as databases, graphs, text, multimedia, or on any other datasets containing objects that could be described by a set of features or relationships [2]. Performing a clustering task over a dataset can lead to a discovery of unknown yet interesting and useful patterns or trends in the dataset. Since clustering algorithms do not require any external knowledge as input (except certain parameters such as k in the *k-Means* algorithm), the process of clustering, in contrast to classification, is often referred to as an unsupervised learning. However, there has always been a natural need to incorporate already collected knowledge into algorithms to make them better both in terms of efficiency and quality of results. This need led to the construction of a new branch of clustering algorithms based on *constraints*. Constraint-based clustering algorithms utilize the fact, that in many applications, the domain knowledge in the form of, say, labeled objects is already known or could be easily specified

by domain experts. Moreover, in some cases such knowledge can be automatically detected. Initially, researchers focused on algorithms that incorporated pairwise constraints on cluster membership or learned distance metrics. Subsequent research was related to algorithms that used many other kinds of domain knowledge [5].

In [12] and [13] we presented the implementation of two neighborhood-based clustering algorithms *ic-NBC* and *ic-DBSCAN*. These two algorithms combined the well-known *NBC* [20] and *DBSCAN* [8] algorithms with two instance-level constraints, *must-link* and *cannot-link*. In this paper, we build upon our previous work. In particular, in Section 4, we provide a formal background behind the algorithms. The standard concepts used in *ic-NBC* and *ic-DBSCAN* (e.g. k-neighborhood, dense point, direct neighborhood-based density reachability, neighborhood-based density reachability, cluster, noise, nearest cluster, parent cluster, etc.) had to be adjusted to the new context of instance constraints and required new definitions. To improve readability, we have introduced a number of examples and figures illustrating the new concepts. Last but not least, we have added an entirely new section with experimental results to verify both quality as well as efficiency of the algorithms.

The paper is divided into six sections. In Section 2 we give a brief introduction to clustering with constraints and describe the related work in the field of constrained clustering – especially related to density-based clustering. In Section 3, the classic *DBSCAN* and *NBC* algorithms are described. In Section 4 we present our own method. Section 5 contains an experimental evaluation of our algorithms. Conclusions and further research is discussed in Section 6.

2. Constraints

2.1. Instance-level constraints

In clustering algorithms with constraints, background or expert knowledge can be incorporated into algorithms by means of different types of *constraints*. [5]. Several types of constraints have been identified so far, for example, instance constraints describing relations between objects or distance constraints such as inter–cluster δ–constraints and intra–cluster ϵ–constraints [2]. Nevertheless, the hard *instance-level* constraints seem to be most useful since the incorporation of just few constraints of this type can improve clustering accuracy. (We use the Silhouette score to measure clustering quality in our experiments.)

In [16] authors introduced two kinds of *instance-level* constraints, namely: the *must-link* and *cannot-link* constraints. These constraints are simple yet have interesting properties. For example *must-link* constraints are symmetrical, reflexive and transitive: if two points, p_0 and p_1 are in a *must-link* relationship, that is, $c_{=}(p_0, p_1)$ (see Table 1 for notation), then these points should be assigned to the same cluster. On the other hand, if two points r_0 and r_1 are in a *cannot-link* relationship, that is, $c_{\neq}(r_0, r_1)$, then these points must not be assigned to the same cluster.

Consider the following example based on Figure 1. In Figure 1.a we present a sample dataset with two must-link constraints $c_{=}(p_0, p_1)$ and $c_{=}(p_2, p_3)$. Each pair of points should be assigned to the same cluster.In Figure 1.b we present a sample dataset with one *cannot-link* constraint $c_{\neq}(p_0, p_1)$. The dataset should be clustered so that points p_0

Table 1. Notation related to *instance-level* constraints used in the paper and auxiliary variables used in pseudo-code of the algorithm.

Notation	Description
$C(p)$	The cluster to which a point p was assigned. If a point has not been decided yet to which cluster it should be assigned then $C(p)$ returns `UNCLASSIFIED`. If p is a *noise* point, then $C(p) =$ `NOISE`.
$C_{=}$	The set of pairs of points that are in a *must-link* relation.
$c_{=}(p_0, p_1)$	Two points p_0 and p_1 are in a *must-link* relation (must be assigned to the same resulting cluster).
$C_{=}(p)$	The set of points which are in a *must-link* relation with point p.
$C_{\neq}$	The set of pairs of points that are in a *cannot-link* relation.
$c_{\neq}(r_0, r_1)$	Two points r_0 and r_1 are in a *cannot-link* relation (must not be assigned to the same resulting cluster).
$C_{\neq}(r)$	The set of points which are in a *cannot-link* relation with point r.
$ClusterId$	The auxiliary integer variable used for storing currently-created clusters identifier.
$p.ClusterId$	By using such a notation we refer to a $ClusterId$ related to point p.
$p.ndf$	Such a notation is used to refer to a value of the *NDF* factor associated with point p.
R_d, R_t	The auxiliary variables for storing deferred points.
$DPSet$	The variable for storing dense points. It is used for in an iterative process of assigning points to clusters.

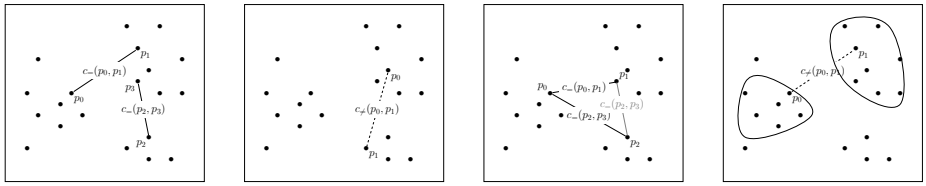
and p_1 will not be assigned to the same cluster.In Figure 1.c and Figure 1.d we illustrate basic features of instance constraints such as transitivity, reflexiveness, symmetry as well as entailment. □

2.2. Related Work

In constrained clustering algorithms, background or expert knowledge can be incorporated into algorithms by means of different types of constraints. Over the years, several methods of using constraints in clustering algorithms have been developed [5]. Constraint-based methods proposed so far employ techniques such as modifying the clustering objective function including a penalty for satisfying specified constraints [6], clustering using conditional distributions in an auxiliary space, enforcing all constraints to be satisfied during clustering process [17] or determining clusters and constraints based on neighborhoods derived from already available labelled examples [1]. In the distance-based methods, the distance measure is designed so that it satisfies given constraints [11,4]. Among algorithms proposed so far, a few represent modifications of density based algorithms, such as *C-DBSCAN* [15], *DBCCOM* [7] or *DBCluC* [18].

C-DBSCAN [15] is an example of a density-based algorithm using *instance-level* constraints where constraints are used to dictate whether some points may appear in the same cluster or not. In the first step, the algorithm partitions the dataset into subspaces using the *KD-Tree* [3] and then enforces *instance-level* constraints within each tree leaf producing so-called *local clusters*. Next, under *cannot-link* constraints, adjacent local clusters are merged enforcing *must-link* constraints. Finally, adjacent clusters are merged hierarchically enforcing remaining *cannot-link* constraints.

DBCluC [18] which was also based on the *DBSCAN* [8] employs an obstacle modelling approach for density-based clustering of large two–dimensional datasets. By means of the modelling it is also capable of detecting clusters of arbitrary shape and is not sensitive to the order of points in a dataset, constraints and noise. The efficiency of clustering



(a) **(b)** **(c)** **(d)**

Fig. 1. An illustration of (a) *must-link* constraints connecting points p_0 and p_1 as well as p_2 and p_3. Points that are connected by must-link constraint have to be assigned to the same cluster; (b) *cannot-link* constraint connecting points p_1 and p_2. In spite of the fact that points may be located relatively closely, if there is a *cannot-link* relation between them, they cannot by assigned to the same cluster; (c) transitive, reflexive and symmetrical features of *must-link* constraints. p_0 and p_1 as well as p_0 and p_2 are connected by *must-link* constraints, thus p_1 and p_2 are also connected by a *must-link* constraint; (d) entailed *cannot-link* constraints. All points from clusters s_0 and s_1 cannot be assigned to the same cluster because of $c_{\neq}(p_0, p_1)$ constraint

is leveraged by a reduction of polygons modelling the obstacles – the algorithm simply removes unnecessary edges from the polygons making the clustering faster in terms of number of constraints to be analysed. Nevertheless, the mechanism of obstacle reduction requires a complex preprocessing to be done before clustering.

The *DBCCOM* algorithm [7] pre-processes an input dataset by modeling the presence of physical obstacles - similarly to *DBCluC*. It also detects clusters of arbitrary shapes and size and is also considered to be insensitive to noise as well as an order of points in a dataset. The algorithm comprises of three steps: first, it reduces the obstacles by employing the edge reduction method, then performs the clustering and finally applies hierarchical clustering on formed clusters. The obstacles in the algorithm are represented as simple polygons and however the algorithm uses a more efficient polygon edge reduction algorithm than *DBCluC*. The results reported by the authors algorithm confirm that it can perform polygon reduction even faster than *DBCCOM* and can produce a hierarchy of clusters.

3. Density-based clustering

Density-based clustering algorithms use density functions to identify clusters. Clusters are dense regions separated by regions of empty space or low density called noise or outliers. Clusters generated in this way can be of arbitrary shape. In this section we describe two density-based algorithms: *DBSCAN* and *NBC*.

3.1. DBSCAN

The *DBSCAN* algorithm [8] is a well known density-based clustering algorithm. The algorithm takes three input parameters: D – the set of data points, ϵ – the radius of the

neighborhood, $MinPts$ – the minimal number of points within ϵ-neighborhood. Each point in D has an attribute called $ClusterId$ which stores the cluster's identifier and initially is equal to UNCLASSIFIED. The key definitions related to the *DBSCAN* algorithm shown below will be used in the sequel. Again, the general notation is given in Table 1.

Definition 1 (ϵ–neighborhood, or $\epsilon NN(p)$ of point p). *ϵ–neighborhood of point p is the set of all points q in dataset D that are distant from p by no more than ϵ; that is,*

$$\epsilon NN(p) = \{q \in D | dist(p,q) \leq \epsilon\},$$

where $dist$ is a distance function.

Clusters in *DBSCAN* are associated with core points which can be considered as seeds of clusters.

Definition 2 (core point). *p is a* core point *with respect to ϵ if its ϵ-neighborhood contains at least $MinPts$ point; that is, $|\epsilon NN(p)| \geq MinPts$.*

The point p_2 in Figure 3a is a core point as its ϵ–neighborhood contains 6 points (we assume $MinPts = 6$ in this case).

Definition 3 (directly density-reachable points). *Point p is directly density reachable from point q with respect to ϵ and $MinPts$ if the following two conditions are satisfied:*
a) $p \in \epsilon NN(q)$
b) *q is a core point.*

Figure 3a illustrates the concept of direct reachability.

Definition 4 (density-reachable points). *Point p is density-reachable from a point q with respect to ϵ and $MinPts$ if there is a sequence of points $p_1, ..., p_n$ such that $p_1 = q$, $p_n = p$ and p_{i+1} and is directly density-reachable from p_i, $i = 1 \ldots n-1$.*

Figure 3b illustrates the concept of reachability.

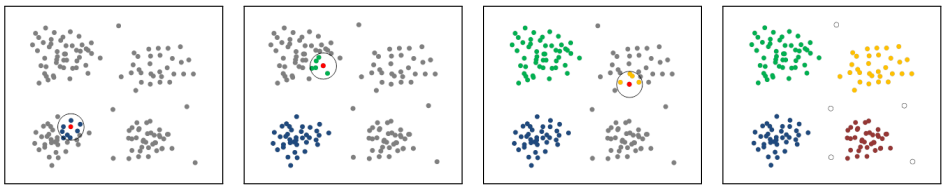
Definition 5 (cluster). *A cluster is a non-empty set of points in D which are density-reachable from the same core point.*

Although Definition 5 is formulated somewhat differently than the definition provided in [8], the resulting clusters are identical in both cases.

Points that are not in dense areas are not associated with any clusters and are considered noise.

Definition 6 (noise). *Noise is the set of all points in D that are not density-reachable from any core point.*

DBSCAN proceeds as follows. Firstly, the algorithm generates a label for the first cluster to be found. Next, the points in D are read. The value of the $ClusterId$ attribute of the first point read is equal to UNCLASSIFIED. While the algorithm analyzes point



(a) **(b)** **(c)** **(d)**

Fig. 2. Illustration of a sample execution of the *DBSCAN* algorithm. The neighborhood of the first core point is assigned to a cluster (a). Subsequent assignment of density-reachable points forms the first cluster; initial seeds are determined for the second cluster (b). The second cluster reaches its maximum size; the initial seeds are determined for the third cluster (c). The third cluster reaches its maximum size; the initial seeds are determined for the fourth cluster. Finally, *DBSCAN* labels noise points represented here by empty dots (d).

after point, it may happen that the $ClusterId$ attributes of some points may change before these points are actually analyzed. Such a case may occur when a point is density-reachable from a core point examined earlier. Such density-reachable points will be assigned to the cluster of a core point and will not be analyzed later. If a currently analyzed point p has not been classified yet (the value of its $ClusterId$ attribute is equal to `UNCLASSIFIED`), then the *ExpandCluster* function is called for this point. If p is a core point, then all points in $C(p)$ are assigned by the *ExpandCluster* function to the cluster with a label equal to the currently created cluster's label. Next, a new cluster label is generated by *DBSCAN*. Otherwise, if p is not a core point, the attribute $ClusterId$ of point p is set to `NOISE`, which means that point p will be tentatively treated as noise. After analyzing all points in D, each point's attribute $ClusterId$ stores a respective cluster label or its value is equal to `NOISE`. An illustration of a sample execution of *DBSCAN* has been ploted in Figure 2.

3.2. Neighborhood-based clustering

The Neighborhood-Based Clustering (*NBC*) [20] algorithm also belongs to the class of density based clustering algorithms. The characteristic feature of *NBC* compared to *DBSCAN* is its ability to measure relative local densities. Hence, it is capable of discovering clusters of different local densities and of arbitrary shape. The algorithm has two parameters: the set of points D and the number k which is used to describe density of a point.

The key definitions related to the *NBC* algorithm are presented below; k–neighborhood and k^+–neighborhood, defined below, are parameters used to describe dense neighborhoods.

Definition 7 (k–neighborhood, or $kNN(p)$). *k–neighborhood of point p is a set of k ($k > 0$) points satisfying the following conditions:*

$$|kNN(p)| = k, \text{ and}$$

$$\forall_{o' \in D \setminus kNN(p)} \forall_{o \in kNN(p)} \; dist(p, o' \geq dist(p, o).$$

Definition 8 (k^+–neighborhood, or $k^+NN(p)$). *k^+–neighborhood of point p is equal to $\epsilon' NN(p)$ where:*

$$\epsilon' = max(\{dist(p, v)|v \in kNN(p)\}).$$

k^+–neighborhood is similar to $\epsilon NN(p)$ (see Def. 3.1). However, ϵ is not a parameter given a priori to the algorithm, but a property of dense neighborhoods relative to a given data set.

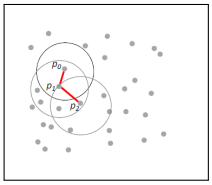
Definition 9 (punctured k^+–neighborhood). *Punctured k^+–neighborhood of point p $k^+NN(p^-)$ is equal to $k^+NN(p) \setminus \{p\}$; that is:*

$$k^+NN(p^-) = k^+NN(p) \setminus \{p\}.$$

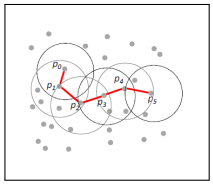
The concept of k^+–neighborhood of p is illustrated in Figure 4a.

Definition 10 (reversed punctured k^+–neighborhood of a point p). *Reversed punctured k^+–neighborhood of a point p $Rk^+NN(p)$ is the set of all points $q \neq p$ in dataset D such that $p \in k^+NN(q^-)$; that is:*

$$Rk^+NN(p) = \{q \in D|p \in k^+NN(q^-)\}.$$

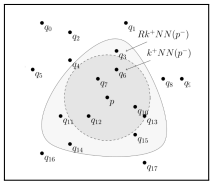


(a) p_0 is directly density-reachable from core point p_1; p_0 is density-reachable from p_2 ($MinPts = 6$).

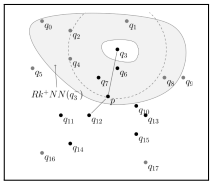


(b) Both p_0 and p_5 are density-reachable from core point p_2, so p_0 and p_5 belong to $C(p_2)$ ($MinPts = 6$).

Fig. 3. Illustration of some of the concepts of *DBSCAN*



(a) q_6 is directly neighborhood-based density reachable from p because $q_6 \in k^+NN(p^-)$.



(b) Point q_{12} is neighborhood-based density reachable from point q_3.

Fig. 4. Illustration of some of the concepts of *NBC* algorithm.

Definition 11 (neighborhood-based density factor of a point – $NDF(p)$). *Neighborhood-based density factor of a point p is defined as*

$$NDF(p) = |Rk^+NN(p)|/|k^+NN(p^-)|.$$

Points having the value of the value of NDF factor equal to or greater than 1, are considered dense.

Definition 12 (dense point). *Point p is called a* local dense point *if its $NDF(p)$ is greater than* 1.

Definition 13 (directly neighborhood–based density reachable). *A point p is directly neighborhood–based density reachable from point q if $p \in k^+NN(q^-)$ and q is a dense (core) point.*

Point q_6 in Figure 4a is directly neighborhood–based density reachable from point p.

Definition 14 (neighborhood-based density reachable). *A point p is neighbor-hood-based density reachable from r if p is directly neighborhood-based density reachable from q and r is directly neighborhood-based density reachable from q.*

Point q_{12} in Figure 4b is directly neighborhood–based density reachable from point q_3.

Definition 15 (cluster). *A cluster is a maximal non-empty subset of D such that for two points p and q in the cluster, p and q are neighborhood-based density-reachable from a local core point with respect to k, and if p belongs to cluster C and q is neighborhood-based density connected with p with respect to k, then q belongs to C.*

Definition 16 (noise). *Noise is the set of all points in D that do not belong to any cluster. In other words, noise is the set of all points in D that are not neighborhood-based density-reachable from any local core point.*

In order to find clusters, NBC starts with calculating values of NDF factors for each point p_i in a database D, $i = 0, 1, \ldots, |D|$. Next, for each p_i, a value of NDF is checked. If it is greater than or equal to 1, then p_i is assigned to the currently created cluster c (identified by the value of $ClusterId$). Next, the temporary variable $DPSet$ for storing references to points, is cleared and each point, say q, belonging to $k^+NN(p_i^-)$ is assigned to c. If $q.ndf$ is greater than or equal to 1, then q is also added to $DPSet$. Otherwise, q is omitted and a next point from $k^+NN(p_i^-)$ is analyzed. Further, for each point from $DPSet$, say r, $k^+NN(r^-)$ is computed. All unclassified points belonging to $k^+NN(r^-)$ are assigned to c and points having values of NDF greater than or equal to 1 are added to $DPSet$. Next, r is removed from $DPSet$. When $DPSet$ is empty, $ClusterId$ is incremented and a next point from D, namely p_{i+1}, is analyzed. Finally, if there are no more points in D having values of NDF factor greater than or equal to 1, then all unclassified points in D are marked as $NOISE$.

4. Clustering with Constraints

In this section we present two density-based algorithms with constraints based on *DBSCAN* and *NBC*. The main modification in both algorithms is the introduction of the `DEFERRED` points. The deferred points are in ϵ–neighborhood (for *DBSCAN* algorithm) or k^+–punctured neighborhood (for the *NBC* algorithm) of points involved in *cannot-link* relationship. The original algorithms are then first executed without the deferred points after which the points are assigned to appropriate clusters to satisfy their *cannot-link* constraints.

The *must-link* constraints are handled in a simple way. In the original algorithms, the construction of clusters originates from the core points. These points are kept in appropriate lists which are then updated in subsequent iterations of the algorithm. If a given core point p is involved in a *must-link* relationship with another core point r, then r is added

to the cluster originating in p. In this way, the algorithm can connect two remote regions via a bridge defined by the pair of points in a *must-link* relationship.

Our interpretation of the instance constraints is slightly different from most of the existing approaches which stop execution of the clustering algorithms upon the discovery of conflicting constraints. We believe that instance constraints do not necessarily have to be fully satisfied. *ic-NBC* and *ic-DBSCAN* use techniques similar to *DBCluC* [18] where the concept of so-called *obstacle points* was introduced. Obstacle points are ignored during the process of clustering. In our algorithms, we treat *cannot-link* constraints (along with their *nearest neighbors*) as points which constitute similar *obstacles*, but we do not ignore them completely during clustering

Thus, in the process of clustering, if a conflicting constraints exists, the algorithm does not have to be stopped, and conflicting points are labeled as `NOISE`.

4.1. ic-DBSCAN

In this subsection we offer a modified version of *DBSCAN* with constraints. First we introduce a definition of *deferred* points (Definition 17) and then present modified definitions of *cluster* and *noise* - Definition 18 and Definition 21, respectively.

Definition 17 (deferred point). *A point p is called* deferred *if it is in a* cannot-link *relationship with any other point or it belongs to a ϵ–neighborhood $\epsilon NN(q)$, where q is any point in a* cannot-link *relationship ($q \in C_{\neq}$). In the latter case we call q a parent point.*

Definition 18 (cluster). *A cluster is a maximal non-empty subset of D such that:*
- *for two* non-deferred *points p and q in the cluster, p and q are neighborhood-based density-reachable from a local core point with respect to k, and if p belongs to cluster C and q is also neighborhood-based density connected with p with respect to k, then q belongs to C;*
- *a* deferred *point p is assigned to a cluster C if the 1st-nearest punctured neighbour of p belongs to C ($1 - NN(p^{-}) \in C$), otherwise, p is considered as a noise point.*

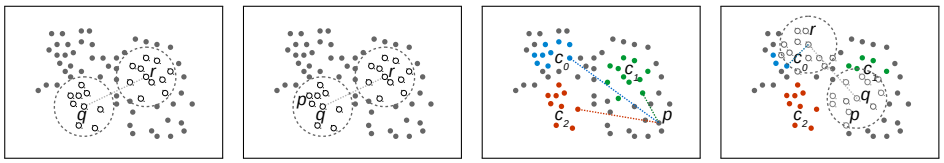
Definition 19 (nearest cluster). *A nearest cluster of a given point p is a cluster C to which p belongs.*

Definition 20 (parent cluster). *A parent cluster of a given point p (g_p) is a cluster C to which a parent point of p belongs.*

Definition 21 (noise). *The noise is the set of all points in D such that each of them is:*
- *not density-reachable from any core point or*
- *is a* deferred *point that has two or more neighbours at the same distance from it and thus can not be unambiguously assigned to a cluster.*

In other words, noise *is the set of all points in D that are not neighborhood-based density-reachable from any local* core point *and* deferred *points points that could not be assigned to any cluster.*

In the first phase, *ic-DBSCAN* algorithm (Figure 6a) omits all points which are involved in any *cannot-link* relationship and marks them as `DEFERRED`. Then, it adds those points to an auxiliary list called R_d which will be later used in the main loop of the algorithm using the *AssignDeferredPoints* function.



(a) Deferred points marked with empty circles.

(b) q is a parent point of point p.

(c) c_1 is a nearest cluster of point p.

(d) c_0 is a parent cluster of point p.

Fig. 5. An illustration of definitions of deferred points (a), parent point (b), nearest and parent cluster (c,d).

Then the algorithm iterates through all `UNCLASSIFIED` points from D except those which were added to R_d. For all of those points it calls the *ExpandCluster* function (Figure 6c) and passes all necessary parameters. The main modifications of the *ExpandCluster* function (compared to the classic *DBSCAN* algorithm) is in how the *must-link* constraints are processed. When a *must-link* point is processed and it is a *core point* or belongs to a neighbourhood of a point which is a *core point*, then it is assigned to $seeds$ or $curSeeds$ lists (containing *seed points*) depending on which part of the *ExpandCluster* function is currently executed. (The $seeds$ and $curSeeds$ are lists containing of points that belong to ϵ–neighborhoods of currently processed point in the *ExpandCluster* function and the number of points in the neighborhood is greater or equal to $MinPts$.)

The last part of the algorithm is to process the set of `DEFERRED` points. This is done by means of the *AssignDeferredPoints* function (Figure 6b). For each point q from R_d (a list of points which were marked as `DEFERRED` in the main algorithm method) the function determines what would be the parent cluster g_p of q. Next, it finds a point $p_{\neq}$ involved in *cannot-link* relationship and similarly determines its parent cluster $g_{p_{\neq}}$. Then, if those two parent clusters are the same ($g_p = g_{p_{\neq}}$) the `DEFERRED` point q cannot be assigned to the nearest cluster g_p and is labeled as `NOISE`. Otherwise, if two parent clusters are different q is assigned to g_p.

4.2. ic-NBC

In this subsection we offer our new neighborhood-based constrained clustering algorithm called *ic-NBC*. The algorithm is based on the *NBC* algorithm [20] but uses both *must-link* and *cannot-link* constraints for incorporating knowledge into the algorithm.

Below we present the definition of deferred point as well as modified definitions of cluster and noise - Definition 23 and Definition 24, respectively.

The *ic-NBC* algorithm employs the same definitions as *NBC* which are used in a process of clustering to assign points to appropriate clusters or mark them as noise. In *NBC* three types of points can be distinguished: *unclassified*, *classified* and *noise* points. In *ic-NBC*, we also employ a concept of `DEFERRED` points although defined slightly different than before.

Definition 22 (deferred point). *A point p is deferred if it is involved in a* cannot-link *relationship with any other point or it belongs to a k^+–punctured neighborhood $k^+NN(q^-)$, where q is any point involved in a* cannot-link *relationship.*

```
Algorihtm ic-DBSCAN(D, k, C=, C≠)
1.  R_d = ∅
2.  label all points in D as UNCLASSIFIED;
3.  ClusterId = label of a first cluster;
4.  for each point q involved in any constraint from C≠ do
5.      label q and points in ϵNN(q) as DEFERRED;
6.  endfor;
7.  add all DEFERRED points to R_d;
8.  foreach point p in set D \ R_d do
9.      if (p.ClusterId = UNCLASSIFIED) then
10.         if ExpandCluster(D, p, ClusterId, ϵ, MinPts) then
11.             ClusterId = NextId(ClusterId);
12.         endif;
13.     endif;
14. endfor;
15. AssignDefferedPoints(D, p, ClId, MinPts, ϵ, C=, C≠);
```

(a) The *ic-DBSCAN* algorithm.

```
Function AssignDeferredPoints(D, R_d, C≠)
1.  for each point q ∈ R_d do
2.      p ← GetParent(q);
3.      g_p ← NearestCluster(p);
4.      p≠ ← C≠(p);
5.      g_p≠ = NearestCluster(p≠);
6.      if g_p = g_p≠ then
7.          mark q as NOISE;
8.      else if
9.          assign point q to g_p;
10.     end if;
11.     remove q from R_d;
12. end for;
```

(b) Assigning deferred points to clusters.

```
Function ExpandCluster(D, p, ClId, MinPts, ϵ, C=, C≠)
1.  seeds = Neighborhood(D, p, ϵ);
2.  if |seeds| < MinPts then
3.      p.ClusterId = NOISE;
4.      return FALSE;
5.  else do
6.      for each point q in seeds do
7.          q.ClusterId = ClId;
8.          add C=(q) to seeds;
9.      endfor
10.     delete p from seeds;
11.     while |seeds| > 0 do
12.         curPoint = first point in seeds;
13.         curSeeds = Neighborhood(D, curPoint , ϵ);
14.         if |curSeeds| ≥ MinPts then
15.             for each point q in curSeeds do
16.                 add C=(q) to seeds;
17.                 if q.ClusterId = UNCLASSIFIED then
18.                     q.ClusterId = ClId;
19.                     append q to seeds;
20.                 else if q.ClusterId = NOISE then
21.                     q.ClusterId = ClId;
22.                 end if;
23.             end for;
24.         end if;
25.         delete curPoint from seeds;
26.     end while;
27. end else;
```

(c) The *ExpandCluster* function.

Fig. 6. The pseudo-code of the *ic-DBSCAN* algorithm using instance constraints.

Definition 23 (cluster). *A cluster is a maximal non-empty subset of D such that:*

- *for two* non-deferred *points p and q in the cluster, p and q are neighborhood-based density-reachable from a local core point with respect to k, and if p belongs to cluster C and q is also neighborhood-based density connected with p with respect to k, then q belongs to C;*
- *a* deferred *point p is assigned to a cluster C if the nearest neighbour of p which is not in* cannot-link *relationship with p belongs to C, otherwise p is considered as a noise point.*

Definition 24 (noise). *Noise is the set of all points in D that:*

- *have not been assigned to any cluster or*
- *each of them is a deferred point p whose $1^+NN(p^-)$ neighborhood contains points assigned to different clusters and thus can not be unambiguously assigned to a particular cluster.*

In other words, noise is the set of all points in D that are not neighborhood-based density-reachable from any local core point and deffered *points points which could be assigned to two or more clusters.*

ic-NBC (Figure 7a) can be divided into two main steps. In the first step the algorithm works very similarly to *NBC*. It calculates NDF factors and performs clustering. The main difference between *ic-NBC* and *NBC* is that the former:

```
Algorithm C-NBC(D, k, C=, C≠)
1.  R_d ← ∅
2.  ClusterId = 0;
3.  label all points in D as UNCLASSIFIED;
4.  CalcNDF(D, k);
5.  for each point q involved in any constraint from C= or C≠ do
6.      label q and points in k+NN(q−) as DEFERRED
7.      add q to R_d;
8.  endfor
9.  ClusterId = 0;
10. for each unclassified point p in D such that p.ndf ≥ 1 do
11.     p.ClusterId = ClusterId;
12.     clear DPSet;
13.     for each point q ∈ k+NN(p−) \R_d do
14.         q.ClusterId = ClusterId;
15.         if (q.ndf ≥ 1) then add q to DPset; endif
16.         add all points r from C=(q) such that
17.             r.ndf ≥ 1 to DPset;
18.     endfor
19.     while (DPSet ≠ ∅) do
20.         s = first point in DPset;
21.         for each unclassified point t in k+NN(s−) \R_d do
22.             t.ClusterId = ClusterId;
23.             if (t.ndf ≥ 1) then add t to DPset; endif
24.             add all points u from C=(t) such that
25.                 u.ndf ≥ 1 to DPset;
26.         endfor
27.         remove s from DPset;
28.     endwhile
29. ClusterId + +;
30. endfor
31. label unclassified points in D as NOISE;
32. AssignDeferredPointsToClusters(D, R_d, k, C≠);
```

(a) The *ic-NBC* algorithm.

```
Function AssignDefferedPointsToClusters(D, R_d, C≠)
Input:
    D - the input dataset (not clustered)
    R_d - the set of points marked as deferred
    k - the parameter of the C-NBC algorithm
    C≠ - the set of cannot-link constraints
Output:
    The clustered set with clusters satisfying cannot-link and
    must-link constraints.
1.  R_t ← R_d;
2.  do begin
3.      R_t ← R_d; // a temporary set for storing deferred points
4.                 // assigned to any cluster
5.      foreach point p in R_t do begin
6.          foreach point q in k+NN(p−) do
7.              if (q.ndf ≥ 1 and q.ClusterId > 0 and q ∈ R_d)
8.                  if (CanBeAssigned(p, q.ClusterId)) and
9.                      // checking if p can be assigned
10.                     // to a cluster identified by q.clusterId
11.                     (CanBeAssigned(
12.                         p.nearestCannotLinkPoint,
13.                         q.ClusterId)) then
14.                             p.ClusterId = q.ClusterId;
15.                             add p to R_a;
16.                     break;
17.                 endif
18.             endif
19.         endfor
20.         remove R_a from R_t;
21.     endfor
22. while (R_a ≠ ∅)
```

(b) The *AssignDefferedPointsToClustesrs* function.

Fig. 7. The pseudo-code of the *ic-NBC* algorithm.

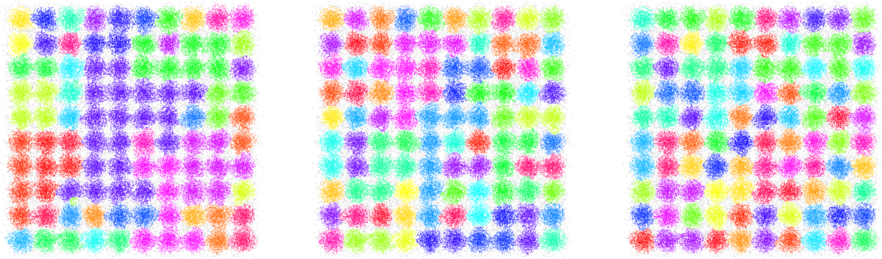
- determines which points will be considered as DEFERRED;
- excludes these points from all calculations (except to compute the values of NDF factors); and
- merges areas of clustered dataset according to *must-link* constraints.

The *ic-NBC* algorithm starts with the *CalcNDF* function. After calculating the NDF factors for each point from D, the deferred points are determined by scanning pairs of *cannot-link* connected points. These points are added to an auxiliary set R_d.

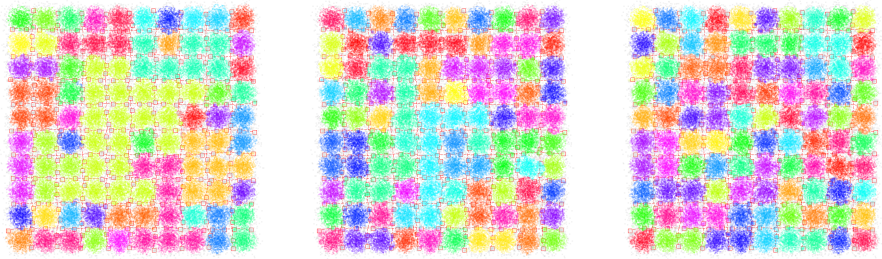
Then, the clustering process is performed in the following way: for each point p which was not marked as DEFERRED, it is checked if $p.ndf$ is less than 1. If $p.ndf < 1$, then p is omitted and a next from $DPSet$ is checked. If $p.ndf \geq 1$, then p as a *dense point* is assigned to the currently-created cluster identified by the current value of $ClusterId$.

Next, the temporary variable $DPSet$ is cleared and each non-deferred point, say q, belonging to $k^+NN(p^-) \setminus R_d$ is assigned to the currently-created cluster identified by the current value of the $ClusterId$ variable. Additionally, if $q.ndf \geq 1$, then it is assigned to $DPSet$ as well as all dense points which are in a must-link relation with q.

Next, for each unclassified point from $DPSet$, say s, its punctured k^+–neighborhood is determined. Each point, say t, which belongs to this neighborhood and has not been labeled as deferred yet is assigned to the currently created cluster and if its value of NDF is equal to or greater than 1, is added to $DPSet$. Moreover, all dense points which are in a must-link relation with t are added to $DPSet$ as well. Later, s is removed from $DPSet$



(a) k = 75, s = 0.063 **(b)** k = 100, s = 0.305 **(c)** k = 125, s = 0.434



(d) k = 75, s = 0.068 **(e)** k = 100, s = 0.303 **(f)** k = 125, s = 0.424

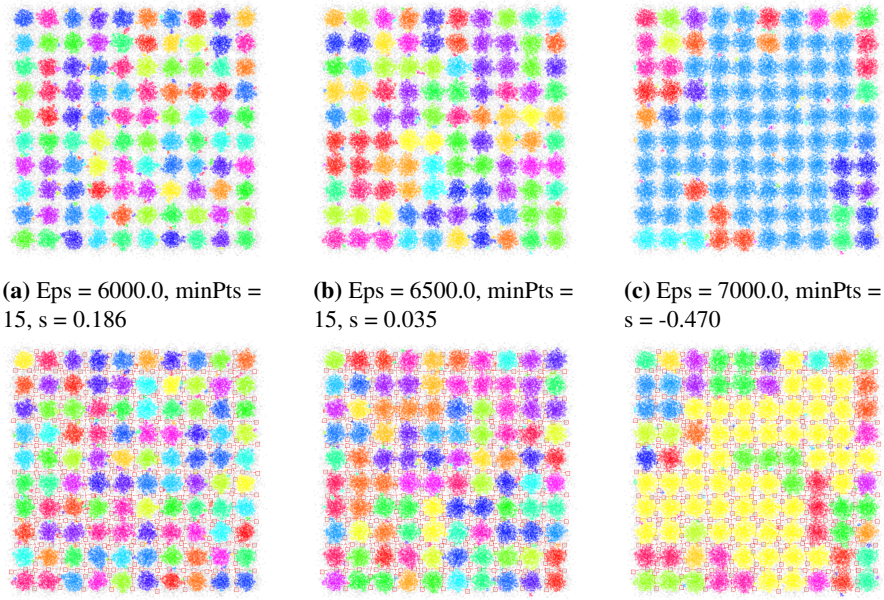
Fig. 8. Results of clustering for the Birch1 dataset using *NBC* (a-c), and *ic-NBC* (d-f). Colors represent mined clusters. k is a parameter if the algorithm. s is a value of the Silhouette factor computed for the given clustering result. Red dashed lines denote *cannot-link* constraints.

and next point from $DPSet$ is processed. When $DPSet$ is emptied, then $ClusterId$ is incremented. After all points from D are processed, unclassified points are marked as noise by setting the values of their $ClusterId$ attribute to `NOISE`. However, in order to process the deferred points, the *AssignDeferredPointsToCluster* function is invoked. The function performs so that for each deferred point p it finds the nearest point q assigned to any cluster and checks whether it is possible (in accordance with cannot-link constraints) to assign p to the same cluster as q. Additionally, the function checks if the assignment of p to a specific cluster will not violate previous assignments of deferred points.

5. Experiments

In this section we present results of the experiments we performed to test the quality and efficiency of the proposed methods. We divided the experiments into two parts. First we focused on *quality* of clustering, then on the *efficiency*.

Datasets. For the experiments we used three standard two dimensional clustering benchmarking datasets (Birch) [19] with 100 000 points and 100 clusters. We examined three different versions of the Birch dataset containing clusters in regular grid structure (Birch1 - Figures 8-9), clusters at a sine curve (Birch2 - Figures 10-11) and random sized clusters in random locations (Birch3 - Figures 12-13).



(a) Eps = 6000.0, minPts = 15, s = 0.186

(b) Eps = 6500.0, minPts = 15, s = 0.035

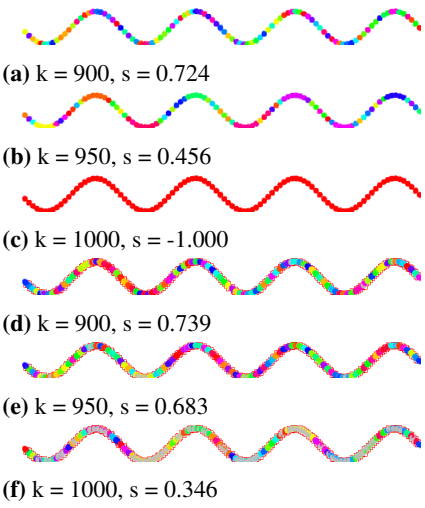
(c) Eps = 7000.0, minPts = 15, s = -0.470

(d) Eps = 6000.0, minPts = 15, s = 0.192

(e) Eps = 6500.0, minPts = 15, s = 0.077

(f) Eps = 7000.0, minPts = 15, s = -0.369

Fig. 9. Results of clustering for the Birch1 dataset using *DBSCAN* (a-c), and *ic-DBSCAN* (d-f). Colors represent mined clusters. Eps and $minPts$ are parameters of the algorithm. s is a value of the Silhouette factor computed for the given clustering result. Red dashed lines denote *cannot-link* constraints.



(a) k = 900, s = 0.724

(b) k = 950, s = 0.456

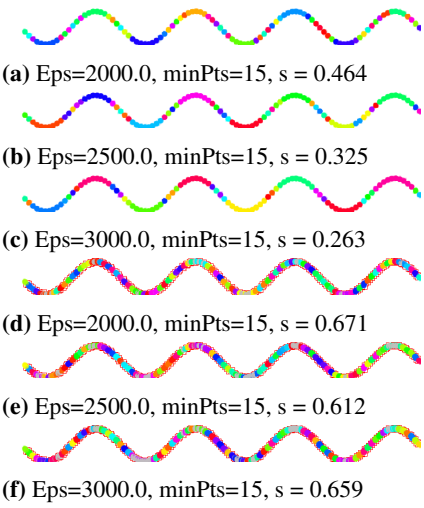
(c) k = 1000, s = -1.000

(d) k = 900, s = 0.739

(e) k = 950, s = 0.683

(f) k = 1000, s = 0.346

Fig. 10. Results of clustering for the Birch2 dataset using *NBC* (a-c), and *ic-NBC* (d-f). Colors represent mined clusters. k is a parameter if the algorithm. s is a value of the Silhouette factor computed for the given clustering result. Red dashed lines denote *cannot-link* constraints.



(a) Eps=2000.0, minPts=15, s = 0.464

(b) Eps=2500.0, minPts=15, s = 0.325

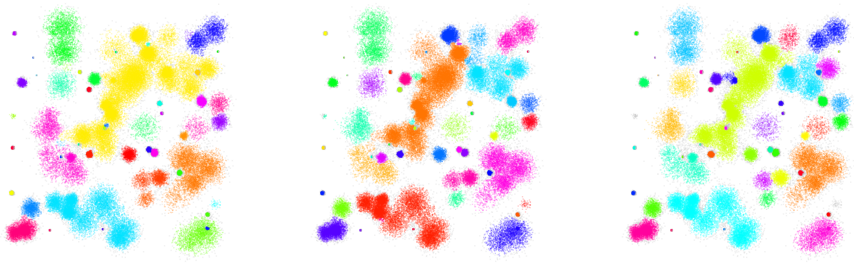
(c) Eps=3000.0, minPts=15, s = 0.263

(d) Eps=2000.0, minPts=15, s = 0.671

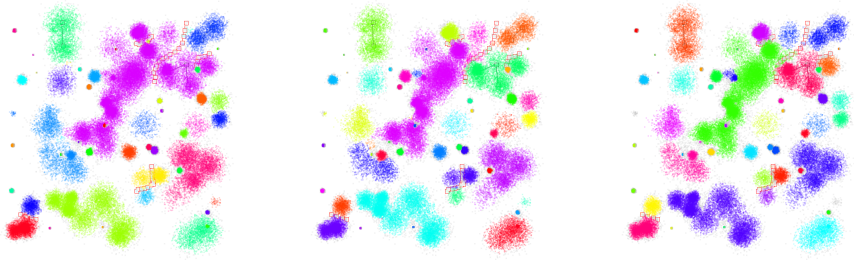
(e) Eps=2500.0, minPts=15, s = 0.612

(f) Eps=3000.0, minPts=15, s = 0.659

Fig. 11. Results of clustering for the Birch2 dataset using *DBSCAN* (a-c), and *ic-DBSCAN* (d-f). Colors represent mined clusters. Eps and $minPts$ are parameters of the algorithm. s is a value of the Silhouette factor computed for the given clustering result. Red dashed lines denote *cannot-link* constraints.

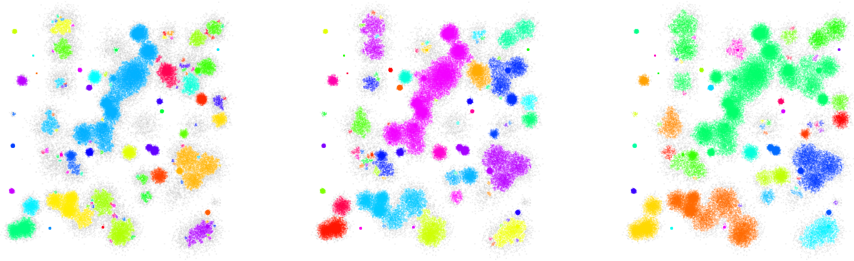


(a) k = 30, s = 0.107 **(b)** k = 50, s = 0.224 **(c)** k = 70, s = 0.282



(d) k = 30, s = 0.113 **(e)** k = 50, s = 0.240 **(f)** k = 70, s = 0.288

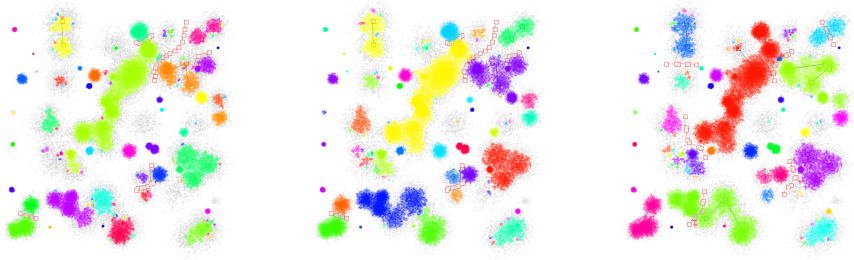
Fig. 12. Results of clustering for the Birch3 dataset using *NBC* (a-c), and *ic-NBC* (d-f). Colors represent mined clusters. k is a parameter if the algorithm. s is a value of the Silhouette factor computed for the given clustering result. Red dashed lines denote *cannot-link* constraints. Black solid lines are *must-link* constraints. In the experiment 12 *must-link* and 43 *cannot-link* constraints were used.



(a) Eps = 5000.0, minPts = 15, s = 0.128

(b) Eps = 6000.0, minPts = 15, s = 0.146

(c) Eps = 7000.0, minPts = 15, s = 0.031

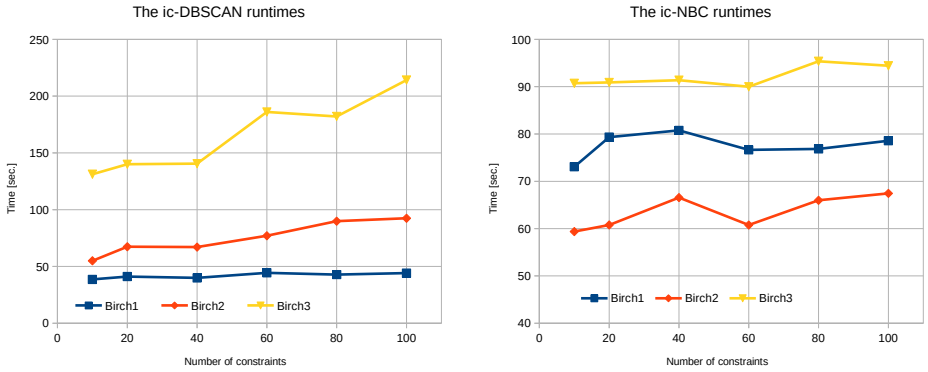


(d) Eps = 5000.0, minPts = 15, s = 0.154

(e) Eps = 6000.0, minPts = 15, s = 0.194

(f) Eps = 7000.0, minPts = 15, s = 0.171

Fig. 13. Results of clustering for the Birch3 dataset using *DBSCAN* (a-c), and *ic-DBSCAN* (d-f). Colors represent mined clusters. Eps and $minPts$ are parameters of the algorithm. s is a value of the Silhouette factor computed for the given clustering result. Red dashed lines denote *cannot-link* constraints. Black solid lines are *must-link* constraints. In the experiment 12 *must-link* and 43 *cannot-link* constraints were used.



(a) *ic-DBSCAN*.

(b) *ic-NBC*.

Fig. 14. Charts with the results of experiments for testing efficiency of *ic-DBSCAN* (a) and *ic-NBC* (b) for datasets Birch1, Birch2, and Birch3.

Table 2. Results of exeriments. Def - number of deferred points; Count - number of discovered clusters; Silh. - a value of the Silhouette score; Ind. - time of index building; Clust. - time of clustering; Tot. = Ind. + Clust. Times are given in milliseconds.

Data	Alg.	Param.	Ind.	Clus.	Def.	Cnt.	Tot.	Silh.
birch1	NBC	75	10145	63372		47	73517	0.063
birch1	ic-NBC	75	9006	67731	6537	49	76737	0.068
birch1	NBC	100	8980	61405		72	70385	0.305
birch1	ic-NBC	100	10399	74341	9877	73	84740	0.303
birch1	NBC	125	8934	67587		89	76521	0.434
birch1	ic-NBC	125	8791	80985	13542	89	89776	0.424
birch1	DBSCAN	6000.0, 15	9892	29946		223	39838	0.186
birch1	ic-DBSCAN	6000.0, 15	8879	34089	4518	216	42968	0.192
birch1	DBSCAN	6500.0, 15	9702	30969		152	40671	0.035
birch1	ic-DBSCAN	6500.0, 15	8885	34726	5386	146	43611	0.077
birch1	DBSCAN	7000.0, 15	9323	31151		73	40474	-0.470
birch1	ic-DBSCAN	7000.0, 15	9648	37155	6778	74	46803	-0.369
birch2	NBC	900	8955	108149		99	117104	0.724
birch2	ic-NBC	900	9260	187205	77742	100	196465	0.739
birch2	NBC	950	9011	118442		64	127453	0.456
birch2	ic-NBC	950	10281	210245	88899	92	220526	0.683
birch2	NBC	1000	9446	120449		1	129895	-1.000
birch2	ic-NBC	1000	9402	205044	87771	43	214446	0.346
birch2	DBSCAN	2000.0, 15	9370	46691		63	56061	0.464
birch2	ic-DBSCAN	2000.0, 15	9026	68285	26811	89	77311	0.671
birch2	DBSCAN	2500.0, 15	8807	51681		46	60488	0.325
birch2	ic-DBSCAN	2500.0, 15	9470	98568	55157	82	108038	0.612
birch2	DBSCAN	3000.0, 15	9243	56087		39	65330	0.263
birch2	ic-DBSCAN	3000.0, 15	8991	144539	101349	85	153530	0.659
birch3	NBC	30	9093	71470		52	80563	0.111
birch3	ic-NBC	30	8848	72752	213	48	81600	0.090
birch3	NBC	50	8712	78941		54	87653	0.234
birch3	ic-NBC	50	9984	79111	450	50	89095	0.149
birch3	NBC	70	8871	84654		50	93525	0.282
birch3	ic-NBC	70	10469	101585	622	45	112054	0.199
birch3	DBSCAN	5000.0, 15	9132	61933		118	71065	0.133
birch3	ic-DBSCAN	5000.0, 15	8482	63644	247	111	72126	-0.014
birch3	DBSCAN	6000.0, 15	9136	62968		80	72104	0.153
birch3	ic-DBSCAN	6000.0, 15	8644	72144	316	78	80788	0.020
birch3	DBSCAN	7000.0, 15	9293	67478		57	76771	0.031
birch3	ic-DBSCAN	7000.0, 15	8464	72594	515	57	81058	0.069

(a) Results of experiments designed to examine quality of proposed constrained algorithms compared to the original versions of the algorithms using the Silhouette score.

Data	Alg.	Param.	Cons.	Cnt.	Ind.	Clus.	Def.	Tot.
birch1	ic-NBC	100	10	62	8728	63990	359	73077
birch1	ic-NBC	100	20	55	9915	68614	803	79332
birch1	ic-NBC	100	40	39	9089	70127	1544	80760
birch1	ic-NBC	100	60	27	8896	65691	2054	76641
birch1	ic-NBC	100	80	15	9184	65275	2396	76855
birch1	ic-NBC	100	100	10	9746	66166	2661	78573
birch1	ic-DBSCAN	5000.0, 15	10	337	9319	28958	309	38586
birch1	ic-DBSCAN	5000.0, 15	20	337	9520	30872	712	41104
birch1	ic-DBSCAN	5000.0, 15	40	333	8739	29988	1192	39919
birch1	ic-DBSCAN	5000.0, 15	60	335	10501	31883	2002	44386
birch1	ic-DBSCAN	5000.0, 15	80	332	8738	31548	2533	42819
birch1	ic-DBSCAN	5000.0, 15	100	328	8960	31973	3216	44149
birch2	ic-NBC	50	10	92	9041	50207	117	59365
birch2	ic-NBC	50	20	81	8966	51558	237	60761
birch2	ic-NBC	50	40	65	10092	55825	633	66550
birch2	ic-NBC	50	60	49	8889	51154	708	60751
birch2	ic-NBC	50	80	32	9199	55838	942	65979
birch2	ic-NBC	50	100	22	9600	56700	1135	67435
birch2	ic-DBSCAN	1000.0, 15	10	94	9652	41497	3836	54985
birch2	ic-DBSCAN	1000.0, 15	20	82	11627	49121	6593	67341
birch2	ic-DBSCAN	1000.0, 15	40	68	9514	44713	12791	67018
birch2	ic-DBSCAN	1000.0, 15	60	53	9078	49770	18159	77007
birch2	ic-DBSCAN	1000.0, 15	80	36	9360	56373	24097	89830
birch2	ic-DBSCAN	1000.0, 15	100	39	9150	56351	26937	92438
birch3	ic-NBC	50	10	45	9050	81492	171	90713
birch3	ic-NBC	50	20	36	9494	80983	421	90898
birch3	ic-NBC	50	40	29	9165	81561	643	91369
birch3	ic-NBC	50	60	22	8499	80386	1097	89982
birch3	ic-NBC	50	80	15	10239	83934	1196	95369
birch3	ic-NBC	50	100	16	9206	83578	1627	94411
birch3	ic-DBSCAN	6000.0, 15	10	72	8711	90199	32349	131259
birch3	ic-DBSCAN	6000.0, 15	20	67	8621	90319	41090	140030
birch3	ic-DBSCAN	6000.0, 15	40	65	8433	91516	40586	140535
birch3	ic-DBSCAN	6000.0, 15	60	61	8815	109920	67327	186062
birch3	ic-DBSCAN	6000.0, 15	80	62	8760	107802	65545	182107
birch3	ic-DBSCAN	6000.0, 15	100	58	9075	124772	80327	214174

(b) Results of experiments testing the efficiency of the proposed constrained density based algorithms with respect to the number of constraints and values of the algorithms' parameters.

Implementation. Both implementations of the algorithms employ the same index structure – the *R-Tree* [9]. We implemented them in Java and performed the experiments on MacBook Pro 2.8GHz eight-core Intel Core i7, 16GB RAM. The source code can be found under the following link: `http://github.com/piotrlasek/clustering`

Quality. To examine how quality of clustering could be improved by means of instance constraints, we used the Silhouette score [14], a method of interpretation and validation of consistency within clusters. The *Silhouette score* for a point i is given by the following formula $s(i) = \frac{(b(i)-a(i))}{max\{a(i),b(i)\}}$, where $a(i)$ is the avarage dissimilarity of i with all other points within the same cluster, $b(i)$ is the lowest average dissimilarity of i to any other cluster to which i does not belong. The silhouette value measures *cohesion* and *separation* that means of how similar an object is to its own cluster compared to other cluster. The values of the silhouette score can range from -1 to $+1$, where a higher value indicates that the object was correctly assigned to its cluster. We report the mean Silhouette value over all objects in a dataset. Times and values of the Silhouette score are reported in Table 2a and Figures 8-14.

The introduction of instance constraints improves the quality of both *DBSCAN* as well as *NBC*; in most cases, the improvement is substantial. However, the clustering quality rises much more for *DBSCAN* than for *NBC*. *NBC* is designed - contrary to *DBSCAN* - to discover clusters with varying local densities (thanks to how the NDF factor was defined). In other words, *DBSCAN* mines clusters based on a global notion of density, *NBC* determines clusters using density calculated locally. For this reason, we do not see as much improvement in employing constraints in *NBC* compared to *DBSCAN*.

Efficiency. In the second part of the experiments we focused on time efficiency of clustering with respect to the number of constraints as well as values of algorithms' parameters (Table 2b, Figures 14a-b).

When performing experiments using *ic-NBC* we were changing the number of *must-link* and *cannot-link* constraints from 10 to 100. Since the additional operations must have been be performed in order to take the constraints into account, this was obvious that constrained versions of the algorithms had to be less effective than the original ones. However, as plotted in Figures 14a-b, the algorithms' execution times are almost constant with respect to the number of constraints used.

6. Conclusions

In this paper we have presented two clustering algorithms with constraints, *ic-NBC* and *ic-DBSCAN*, which were designed to let users introduce instance constraints for specifying background knowledge about the anticipated groups. In our approach we treat must-link constraints as more important than *cannot-link* constraints. Thus, we try to satisfy all must-link constraints (assuming, of course, that all of them are valid) before incorporating any cannot-link constraints. When processing *cannot-link* constraints, points which are contradictory (in terms of satisfying both *must-link* and *cannot-link* constraints) are marked as noise.

We have performed a number of experiments to test the quality of clustering and the efficiency of our algorithms by comparing them to their original versions. The results of the experiments clearly show that constraints are useful in clustering.

The experiments proved that the introduction of instance constraints improved the quality of clustering in both cases. At the same time, due to additional computations needed to process constraints, the performance of the algorithms was reduced but only marginally. The experiments also showed that the number of constraints does not have a critical impact on the algorithms performance.

In this work we have focused on of incorporating *instance-level* constraints into clustering algorithms by modifying the algorithms. Nevertheless, there are other ways of incorporating constraints into the process of clustering. For example, the constraints can be used to modify a distance matrix so that it reflects *must-link* and *cannot-link* relationships. Such a matrix can then be used as an input to the original algorithm without constraints. We believe that this is a promising area of research and we plan to explore it in future.

References

1. Basu, S., Banerjee, A., Mooney, R.: Semi-supervised clustering by seeding. In: In Proceedings of 19th International Conference on Machine Learning (ICML-2002. Citeseer (2002)
2. Basu, S., Davidson, I., Wagstaff, K.: Constrained clustering: Advances in algorithms, theory, and applications. CRC Press (2008)
3. Bentley, J.L.: Multidimensional binary search trees used for associative searching pp. 509–517 (1975)
4. Chang, H., Yeung, D.Y.: Locally linear metric adaptation for semi-supervised clustering. In: Proceedings of the twenty-first international conference on Machine learning, p. 20. ACM (2004)
5. Davidson, I., Basu, S.: A survey of clustering with instance level constraints. ACM Transactions on Knowledge Discovery from Data **1**, 1–41 (2007)
6. Davidson, I., Ravi, S.: Clustering with constraints: Feasibility issues and the k-means algorithm. In: SDM, vol. 5, pp. 201–211. SIAM (2005)
7. Duhan, N., Sharma, A.: Dbccom: Density based clustering with constraints and obstacle modeling. In: Contemporary Computing, pp. 212–228. Springer (2011)
8. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Kdd, vol. 96, pp. 226–231 (1996)
9. Guttman, A.: R-trees: a dynamic index structure for spatial searching, vol. 14. ACM (1984)
10. Han, J., Kamber, M.: Data Mining, Southeast Asia Edition: Concepts and Techniques. Morgan kaufmann (2006)
11. Hertz, T., Bar-Hillel, A., Weinshall, D.: Boosting margin based distance functions for clustering. In: Proceedings of the twenty-first international conference on Machine learning, p. 50. ACM (2004)
12. Lasek, P.: C-nbc: Neighborhood-based clustering with constraints. In: Proceedings of the 23th International Workshop on Concurrency, Specification and Programming, vol. 1269, pp. 113–120 (2014)
13. Lasek, P.: Instance-level constraints in density-based clustering. In: Proceedings of the 24th International Workshop on Concurrency, Specification and Programming, pp. 11–18 (2015)
14. Rousseeuw, P.J.: Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. Journal of Computational and Applied Mathematics **20**(Supplement C), 53 – 65 (1987). DOI https://doi.org/10.1016/0377-0427(87)90125-7. URL `http://www.sciencedirect.com/science/article/pii/0377042787901257`
15. Ruiz, C., Spiliopoulou, M., Menasalvas, E.: C-dbscan: Density-based clustering with constraints. In: Rough Sets, Fuzzy Sets, Data Mining and Granular Computing, pp. 216–223. Springer (2007)

16. Wagstaff, K., Cardie, C.: Clustering with instance-level constraints. AAAI/IAAI **1097** (2000)
17. Wagstaff, K., Cardie, C., Rogers, S., Schrödl, S., et al.: Constrained k-means clustering with background knowledge. In: ICML, vol. 1, pp. 577–584 (2001)
18. Zaïane, O.R., Lee, C.H.: Clustering spatial data when facing physical constraints. In: Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on, pp. 737–740. IEEE (2002)
19. Zhang, T., Ramakrishnan, R., Livny, M.: Birch: A new data clustering algorithm and its applications. Data Mining and Knowledge Discovery **1**(2), 141–182 (1997)
20. Zhou, S., Zhao, Y., Guan, J., Huang, J.: A neighborhood-based clustering algorithm. In: Advances in Knowledge Discovery and Data Mining, pp. 361–371. Springer (2005)

Piotr Lasek received his M.Sc. degree in computer science from the Warsaw University of Technology in 2004 and the Ph.D. degree from the same university in 2012. Currently, is an Assistant Professor at the Rzeszow University, Poland. His main areas of research include interactive data mining and visualization.

Jarek Gryz is a Professor at the Department of Computer Science and Engineering at York University, Canada. He received his Ph.D. in Computer Science from the University of Maryland, USA, in 1997. His main areas of research involve database systems, data mining, query optimization via data mining, preference queries, query sampling.

Received: June 1, 2018; Accepted: February 20, 2019.