INSCY: Indexing Subspace Clusters with In-Process-Removal of Redundancy

Conference Paper · December 2008

DOI: 10.1109/ICDM.2008.46 · Source: DBLP

4 authors, including:

Ira Assent
Aarhus University
97 PUBLICATIONS **2,094** CITATIONS

SEE PROFILE

Thomas Seidl
Ludwig-Maximilians-University of Munich
346 PUBLICATIONS **5,456** CITATIONS

SEE PROFILE

INSCY: Indexing Subspace Clusters with In-Process-Removal of Redundancy

Ira Assent * Ralph Krieger Emmanuel Müller Thomas Seidl
Data management and exploration group
RWTH Aachen University, Germany
{assent, krieger, mueller, seidl}@cs.rwth-aachen.de

Abstract

Subspace clustering aims at detecting clusters in any subspace projection of a high dimensional space. As the number of projections is exponential in the number of dimensions, efficiency is crucial. Moreover, the resulting subspace clusters are often highly redundant, i.e. many clusters are detected multiply in several projections. We propose a novel index for efficient subspace clustering in a novel depth-first processing with in-process-removal of redundant clusters for better pruning. Thorough experiments on real and synthetic data show that INSCY yields substantial efficiency and quality improvements.

1 Introduction

Clustering in high dimensional spaces is obstructed by noise of irrelevant attributes. Subspace clustering thus mines clusters in all subspace projections. As the number of subspace projections is exponential in the dimensionality, efficiency is crucial [1, 5]. The number of resulting subspace clusters is usually exponential as well. Clusters typically show in different subspaces, generating redundant subspace clusters which contain essentially the same information as the "maximal" high dimensional one. To allow users to analyze reasonable result sizes, redundant subspace clusters have to be removed. Existing subspace clustering algorithms proceed breadth-first on the subspace lattice: based on low-dimensional results higher-dimensional candidate sets are generated [1, 5, 7]. Huge numbers of candidates in low dimensional subspaces result in excess candidates, redundant results and poor runtimes. As maximal high dimensional results are mined last, redundant projections can only be cleared out once all subspace clusters have been computed.

We propose INSCY (**IN**dexing **S**ubspace **C**lusters with in-process-removal of redundanc**Y**), a new depth-first approach, i.e. recursive mining in a region of all clusters in all subspace projections, before continuing with the next region. This strategy has two key advantages: First, as the maximal high dimensional projection is evaluated first, immediate pruning of all its redundant low dimensional projections leads to major efficiency gains. Second, indexing of potential subspace cluster regions is possible. INSCY combines in-process redundancy pruning with our novel index structure, the **SCY**-tree, for very efficient subspace clustering. The SCY-tree is a compact representation of potential subspace cluster regions which greatly reduces database scans. Substantial efficiency gains and automatically reduced output size render INSCY fast and concise.

2 Non-redundant Subspace Clusters

Density-based clustering has been shown to successfully detect clusters of arbitrary shape and size even in noisy data [3, 5, 2]. It defines clusters as maximal density-connected sets of dense objects. Density-connections are chains of objects with inclusion in the respective neighborhoods N_ε of the chain. Maximality means that density-connected objects are all assigned to the same cluster. In subspace clustering, the projection to the respective subspace $\mathbf{S}$ extends these notions.

Breadth-first subspace clustering suffers from two major drawbacks: redundancy and inefficiency. Subspace clusters and all of their lower-dimensional projections are computed. These lower dimensional projections typically do not differ much from their higher-dimensional counterparts. Figure 1 illustrates this problem: the $2d$ subspace cluster

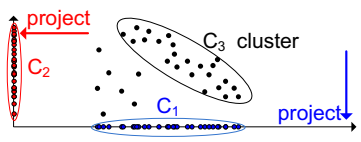


Figure 1. Density-based subspace clustering

* now with Department of Computer Science, Aalborg University, Denmark

C_3 generates redundant $1d$ clusters C_1, C_2. Both contain less information than C_3 as the $2d$ correlation is not visible. Thus, lower dimensional projections are typically considered less informative. As the number of projections is exponential, the number of redundant subspace clusters is overwhelming. Consequently, the output has to be filtered for user benefit. In breadth-first mining, redundant subspace clusters can only be detected once all lower dimensional projections have been processed. Removal cannot be exploited for runtime improvements.

Formally, a cluster $(\mathbf{C}, \mathbf{S})$ is *redundant*: $\exists(\mathbf{C}', \mathbf{S}') : \mathbf{C}' \subseteq \mathbf{C} \wedge \mathbf{S}' \supset \mathbf{S}$ and the redundant subspace cluster $\mathbf{C}$ in subspace $\mathbf{S}$ is covered to a degree of redundancy R by a cluster $|\mathbf{C}'| \geq R \cdot |\mathbf{C}|$ in a higher-dimensional subspace $\mathbf{S}' \supset \mathbf{S}$. Parameter R sets the degree of acceptable redundancy to $0 - 100\%$: $R = 0\%$ means no redundancy, i.e. no higher-dimensional cluster exists, whereas $R = 100\%$ corresponds to almost full redundancy, i.e. any cluster is reported if it contains at least one object more than a higher-dimensional cluster. A cluster $(\mathbf{C}, \mathbf{S})$ is *fully redundant*: $\exists(\mathbf{C}', \mathbf{S}') : \mathbf{C}' \subseteq \mathbf{C} \wedge \mathbf{S}' \supset \mathbf{S} \wedge |\mathbf{C}'| = |\mathbf{C}|$. Full redundancy is not considered acceptable, as a projection of the same set of objects is not interesting at all. We follows the definition of density-based subspace clusters in [2], and for simplicity of presentation, use rectangle kernels for density assessment (as e.g. in [3, 5]), yet in principle any kernel can be used.

Definition 1 ***Subspace Cluster.***

A subspace cluster $(\mathbf{C}, \mathbf{S})$ *is a set of objects* $\mathbf{C} \subseteq DB$*, with* $|\mathbf{C}| \geq minSize$ *in subspace* $\mathbf{S}$ *with respect to a density threshold* $\tau_{\mathbf{S}}$*, redundancy factor* R *and* ε*-neighborhood* $\mathcal{N}_{\varepsilon}^{\mathbf{S}}(q) = \{p \in DB \mid dist(p^{\mathbf{S}}, q^{\mathbf{S}}) \leq \varepsilon\}$*, where the projection of object* o *to a subspace* $\mathbf{S}$ *is denoted by* $o^{\mathbf{S}}$ *if:*

- *objects in* $\mathbf{C}$ *are* $\mathbf{S}$***-dense****:*
  $\forall o \in \mathbf{C} : \ |\mathcal{N}_{\varepsilon}^{\mathbf{S}}(o)| \geq \tau_{\mathbf{S}}$
- *objects in* $\mathbf{C}$ *are* $\mathbf{S}$***-connected****:*
  $\forall o, p \in \mathbf{C} : \exists\, q_1, \ldots, q_m \in \mathbf{C} : q_1 = o \ \wedge q_m = p \wedge$
  $\forall i \in \{2, \ldots, m\}\ q_i \in \mathcal{N}_{\varepsilon}^{\mathbf{S}}(q_{i-1})$
- $\mathbf{C}$ *is* ***maximal:***
  $\forall o, p \in \mathbf{DB}$*:* o, p *S-connected* $\Rightarrow (o \in \mathbf{C} \Leftrightarrow p \in \mathbf{C})$
- $(\mathbf{C}, \mathbf{S})$ ***is not redundant:***
  $\neg\exists(\mathbf{C}', \mathbf{S}')$ *subspace cluster with* $\mathbf{C}' \subseteq \mathbf{C} \wedge \mathbf{S}' \supset \mathbf{S} \wedge$
  $|\mathbf{C}'| \geq R \cdot |\mathbf{C}|$

In addition to redundancy removal, the dimensionality-dependent threshold $\tau_{\mathbf{S}}$ takes the expected density of subspaces $|\mathbf{S}|$ into account. Details can be found in [2].

3 Depth-first processing

We have identified three problems of existing breadth-first subspace clustering algorithms: first, large sets of candidates in low-dimensional projections, second, no in-process-removal of redundant subspace clusters, and third, repeated density computations due to lack of index support. To overcome these drawbacks and to reduce runtimes substantially, we propose a depth-first approach.

Breadth-first algorithms are similar to the apriori algorithm originally introduced in frequent itemset mining in that they work their way bottom-up on the subspace lattice. Starting from one dimensional results, all subspace clusters of dimensionality k are mined, before candidates of dimensionality $k + 1$ are generated, and so on.

Typically, the number of low dimensional subspace clusters is huge, as they reflect high dimensional subspace clusters in several projections. As in breadth-first approaches all low dimensional subspaces have to be processed before going into higher-dimensional ones, these approaches have to perform costly density-based clustering on all lower dimensional projections even if they are all redundant. In breadth-first mining, redundant clusters can only be detected once the entire low dimensional projection has been processed. This means that no redundancy-based pruning is possible, as the redundant cluster is already output before its high dimensional representative is processed. Thus, these approaches show not only extremely large result sizes (redundancy) but also high runtimes (no in-process pruning of redundancy) of several days as stated in [5].

As we know, the only type of index support to speed up such density computations in subspace clustering is the usage of inverted files for individual dimensions in [5]. Indexing subspace clusters in breadth-first algorithms would require building index structures for each of the exponential many subspace combinations, which is clearly not advantageous. Thus, depth-first processing is the key to efficient index-based subspace clustering. Our novel index structure for depth-first mining supports access to arbitrary subspaces without mining their lower dimensional projections.

In this work, we show how incorporating redundancy removal into depth-first mining allows for more efficient and accurate subspace clustering. In a recursive fashion on subspace regions, we first identify high dimensional subspace clusters and immediately prune all lower dimensional projections and thus avoid costly density computations on irrelevant low dimensional candidates.

In contrast to breadth-first subspace clustering our algorithm does not need all lower dimensional subspaces for a restriction. Our depth-first processing is particularly advantageous as it allows for pruning redundant clusters during the mining process. In Figure 2 we assume a cluster found in subspace $\{1, 2, 5\}$. During the depth-first processing the shaded subspaces are recursively restricted to $\{1, 2, 5\}$. In contrast to breadth-first approaches $\{1\}, \{2\}, \{5\}, \{1, 2\}, \{1, 5\}$ and $\{2, 5\}$ do not have to be clustered to get to this subspace. If a cluster found in

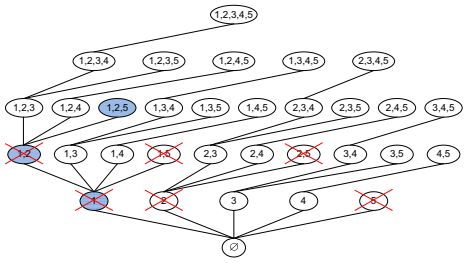


Figure 2. Depth-first in subspace lattice

$\{1, 2, 5\}$ is the only non-redundant result, then we avoid the costly density computations for all of its lower dimensional subspaces. Technically, we step back in the recursion from high dimensional to lower dimensional projections, where all actual high dimensional clusters are available for immediate pruning of redundant regions in the crossed out subspaces in Figure 2.

By processing in depth-first order and using in-process redundancy removal we reduce costly density-based clustering and hence improve the overall efficiency of subspace clustering.

4 Indexing subspace clusters

Our novel index for depth-first mining provides a compact representation of the given data with which we can access arbitrary subspaces without generating all lower dimensional projections. Thus in-process removal of redundant patterns is efficiently supported. To index subspace regions, we use a transformation from the original space to a compact tree structure. Using a single data base scan, the initial SCY-tree is built which represents the entire data base in the full dimensional space. Each leaf node stores the count of objects in a subregion, that is described by the path from the root to this leaf. Special paths indicate neighboring S-connected regions and thus we achieve that each cluster is represented by one SCY-tree by merging SCY-trees representing neighboring S-connected regions. For depth-first mining, SCY-trees can be efficiently restricted to higher-dimensional subspace projections by simply extracting path information. Consequently, pruning of sparse regions and of redundant regions is performed directly on the SCY-trees without costly neighborhood computations. Only for validating high dimensional subspace cluster candidates, access to the original data is required.

SCY-tree structure

To efficiently support density-based subspace clustering by our novel indexing structure, we use a compact representation of potentially dense regions. Reconsidering Definition 1, we see that any potential cluster consists of a minimum number of S-dense objects, i.e. objects whose neighborhood count exceeds a certain threshold. We represent these more compactly by mapping regions from the original space to descriptor nodes that record the object count for these regions. Additionally, potential clusters are maximally S-connected, i.e. they consist of chains of dense objects within the ε-neighborhood of one another. To ensure that these connections are adequately reflected in the descriptor node representation, we additionally record information of objects within ε distance of any neighboring region.

Each region can be described as the subspace dimensions it spans and the respective intervals in these dimensions. Figure 3 illustrates the basic idea: we use intervals in each dimension to describe regions in the original space and for each region we store the number of objects. Additionally we ensure S-connectedness by regions of ε-width such that an object has to be present in such a region if two neighboring regions contain a S-connected cluster. In contrast to grid-based techniques like CLIQUE [1] we ensure to find density-based subspace clusters and thus we do not cut clusters apart.

For the given high dimensional data we perform one data base scan and create the initial SCY-tree on which mining operations are performed without excessive data base scans. As depicted in Figure 3 for a 3-dimensional dataset we show one 2-dimensional projection of the data on the left side and the initial SCY-tree $T_{\{\}}$ in the middle. It represents the whole data base, while a restricted SCY-tree $T_{\{(1,1)\}}$ on the right side represents all data objects that are part of region 1 in dimension 1. As we can see, the tree contains a single path, as all three objects in this region are located in interval **2** in dimension two and in interval **2** in dimension three.

The general idea is that one can restrict a region to further dimensions by restricting the corresponding SCY-tree representing this region. How to restrict trees and how to handle density-based clusters across multiple regions via merging of trees is described in the next section.

Definition 2 ***SCY-tree structure***
A SCY-tree T_D represents a region $D = \{(d_1, i_1), \ldots, (d_k, i_k)\}$ in an arbitrary subspace. The SCY-tree consists of nodes, each of them stores:

- *a descriptor (d, i) for dimension d and interval i of the region and its count c of objects*
- *a pointer to the parent node and a list of child pointers*
- *a pointer of a linked list of nodes with the same descriptor*

The main properties of a node are the descriptor and count value. For shorter representation we thus address a node simply by $(d, i) : c$. The SCY-tree nodes are ordered

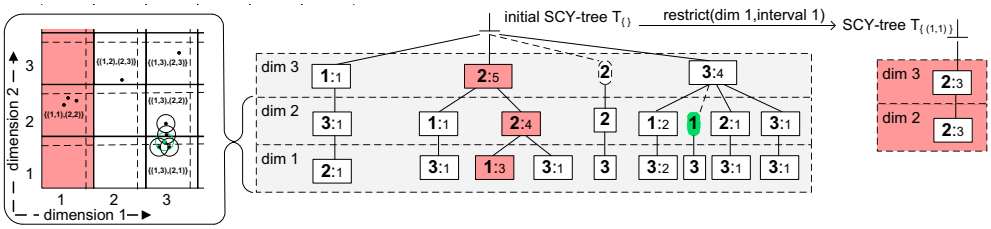


Figure 3. Initial SCY-tree and first recursion step

lexicographically according to their descriptors: first, according to the dimensions and second according to the interval. We omit the dimension in the illustration of each node (cf. Figure 3) as each level of the tree corresponds to one dimension. The special ε-width regions (*S-connectors*) that ensure correct density-based clustering are represented by $(d, i) : -1$ as we are only interested in the presence of an object. Nodes representing S-connectors are depicted without a count value (cf. green node in Figure 3).

By using S-connectors we ensure that each region containing a density-based cluster is represented by one SCY-tree. We therefore set-up the S-connectors at the upper border of each interval. Given an S-connected cluster spreading across two neighboring regions there has to be at least one object in this ε-width region. For the mining this indicates that INSCY has to merge these two neighboring regions to have the whole S-connected cluster represented in one SCY-tree. For multiple regions this merge operation on neighboring SCY-trees can be done iteratively until no further object is contained in any surrounding ε region.

5 INSCY algorithm

INSCY mines subspace clusters directly on SCY-tree paths that correspond to regions and their S-connected regions.

Algorithm 1: INSCY(scy-tree, result)

```
1 foreach descriptor in scy-tree do
2     restricted-tree := restrict(scy-tree, descriptor);
3     restricted-tree := mergeWithNeighbors(restricted-tree);
4     pruneRecursion(restricted-tree); //prune sparse regions
5     INSCY(restricted-tree,result); //depth-first via recursion
6     pruneRedundancy(restricted-tree); //in-process-removal
7     result := DBClustering(restricted-tree) ∪ result;
```

1. Restricting SCY-trees: searching different subspaces

In one recursive INSCY call the for-loop restricts the current SCY-tree for each descriptor (d, i) in lexicographical order of descriptors. Restriction means that only objects in dimension d in interval i contribute, consequently the tree decreases by at least level d. This step is similar in spirit to conditional FP-growth steps in association rule mining that builds conditional subtrees for frequency counts [4]. Note that any descriptor, not only those at leaf level, are picked for restriction. This is crucial for detecting arbitrary subspace clusters in any possible combination of dimensions. INSCY can thus efficiently obtain any possible subspace by extracting the relevant path information into restricted trees. For efficient restriction, all nodes labeled with the same descriptor are accessed via a linked list.

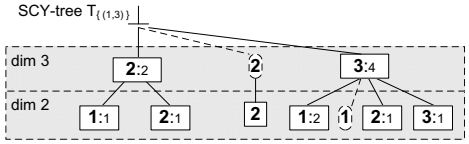


Figure 4. SCY-tree for dimension 1 interval 3

Example. The initial SCY-tree in Figure 3 is first restricted to $(1, 1)$ following lexicographical order, i.e. in dimension 1, interval 1 (red region). For each node in dimension 1 (at leaf level) labeled **1** (only the third leaf node from the left with count 3), the paths from this node to the root are copied into the restricted SCY-tree $T_{\{(1,1)\}}$, depicted at the right, labeled with the count of that node.

2. Pruning recursive calls: removing sparse regions

If any SCY-tree count does not exceed the parameter $minSize$, further restriction, i.e. projection, can only lead to lower count values that do not exceed $minSize$ either. Consequently, they can be safely pruned.

Example. $T_{\{(1,1)\}}$ has no S-connector in dimension 1, thus the region cannot be grown. The count of 3 is below our example $minSize$ value of 4, thus the tree is pruned.

3. Merging SCY-trees: growing S-connected regions

Potentially S-connected restricted trees are easily detected from special paths without count values. Merging of S-connected restricted SCY-trees means simply inserting all paths of one tree into the other one, aggregating the count values on common paths, possibly inserting new nodes.

Example. Restricting the SCY-tree $T_{\{(1,3)\}}$ (Fig. 4) in dimension 2 we get the two SCY-trees for $(2, 1)$ and

$(2,2)$ shown in Figure 5. Neither exceeds $minSize$, but the S-connector node 1 in dimension 2 in the parent tree $T_{\{(1,3)\}}$ indicates a connection in this dimension from $(2,1)$ to the lexicographical neighbor $(2,2)$. The two SCY-trees are merged to the SCY-tree $T_{\{(1,3)\times(2,1-2)\}}$ representing both intervals in dimension 2, depicted in Figure 5.

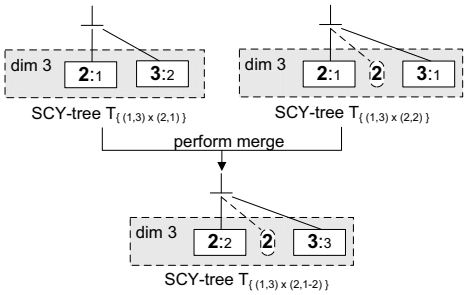


Figure 5. Merge of two SCY-trees

4. Clustering: mining actual subspace clusters

In the final step, the density-based clustering on the restricted SCY-tree, the actual data is accessed to identify the actual neighborhoods and to check all conditions of subspace clusters (Definition 1). This access, however, is required only for those few regions that cannot be pruned.

Example. Figure 5: $T_{\{(1,3)\times(2,1-2)\}}$ is merged in dimension 3 according to S-connector node 2, yielding $T_{\{(1,3)\times(2,1-2)\times(3,2-3)\}}$ with a count of $5 \geq minSize$. There is no further dimension that we could restrict the tree in. Since our result set is still empty, no redundancy pruning is possible, and we cluster the region $(1,3) \times (2,1-2) \times (3,2-3)$. In our example, we assume detection of a cluster.

5. Redundancy pruning: in-process removal

In-process-removal of redundant clusters is checked on the SCY-tree (with respect to the redundancy parameter R) if there is already a cluster in the result set covering this region. As INSCY proceeds in a depth-first manner, the recursion stops if no higher-dimensional clusters can be found (i.e. no further restriction is possible, or $minSize$ is no longer exceeded). INSCY then steps back to analyze lower-dimensional projections, checking only SCY-trees containing potentially non-redundant clusters. As the maximal high dimensional subspace clusters is the first one to be included in the result set, costly mining steps of low dimensional projections are avoided.

Example. Stepping back in the recursion, process $T_{\{(1,3)\times(2,1-2)\}}$. The redundancy check shows that SCY-tree $T_{\{(1,3)\times(2,1-2)\}}$ has a count of only 5. Thus no DBClustering is performed, because this is exactly the size of the subspace cluster found in the 3-dimensional subspace $(1,3)(2,1-2)(3,2-3)$.

6. Arbitrary restrictions: detecting all subspace clusters

The INSCY algorithm mines arbitrary subspaces, i.e. any possible combination of dimensions. Any dimension not currently under consideration, be it at leaf level or above, is simply disregarded during restriction of SCY-trees. More precisely, as the path information is extracted, in-between dimensions do not require any special treatment. Consequently, SCY-trees provide the means for efficient mining of any subspace.

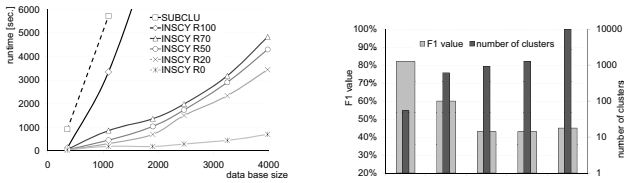
Example. Another step back in the recursion leads to $T_{\{(1,3)\}}$ shown in Figure 4. Here, dimension 2 has already been processed, and it does not need to be restricted any more. Next, restriction in dimension 3 is mined. From Figure 4 we can see that the resulting SCY-tree $T_{\{(1,3)\times(3,2-3)\}}$ would have a count of 6, as there is a count of 2 in cell 2 and a count of 4 in cell 3 with an S-connector path between them. Redundancy pruning is possible again, as with a redundancy parameter of 80% the inequation $(5 \geq 80\% \cdot 6)$ is true and thus a count of 6 is considered too close to the already detected subspace cluster of size 5 for mining or inclusion in the result set. In the last step back to the initial SCY-tree, the INSCY algorithm detects a non-redundant cluster. Skipping dimension 1 and performing restriction and merge for dimension 2 yields $T_{\{(2,1-2)\}}$ with a count of 8. A further recursive call of INSCY on dimension 3 performs the second DBClustering call on $T_{\{(2,1-2)\times(3,2-3)\}}$ because this time $(5 \geq 80\% \cdot 8)$ does not hold for redundancy pruning. Thus, actual subspace clustering is performed.

Correct detection of non-redundant subspace clusters

First, all non-redundant clusters are in the listing, which is straightforward as any SCY-tree that is pruned with respect to redundancy is compared against subspace clusters that have already been stored in the result listing. As no subspace cluster is later removed from the listing, all non-redundant clusters are included. Second, no redundant cluster is in the output. This is ensured as INSCY mines regions in a depth-first manner, i.e. each individual region is clustered in all subspace projections. Due to the recursive nature of the algorithm, higher-dimensional subspace clusters are mined first, before stepping back to lower-dimensional subspace clusters in the same region which can then be pruned.

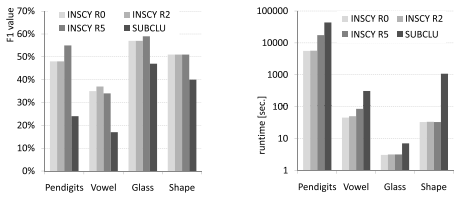
6 Experiments

INSCY is compared to the most recent non-approxive density-based algorithm SUBCLU [5] that uses inverted



(a) Runtime (b) Quality

Figure 6. Evaluation of Redundancy.



(a) Quality (b) Runtime

Figure 7. Evaluation on real world data.

files on individual dimensions as the only possible indexing. Synthetic density-based subspace clusters are generated as in [5, 2] to generate density-based clusters in arbitrary subspaces with additional overlapping subspace clusters, just as in most real world data sets. Quality is measured via $F1$ measure [9, 7], the harmonic mean of recall ("are all clusters detected?") and precision ("are the clusters accurately detected?"). F1-value of a clustering is the average of all F1-values, where the most frequent label is assigned to any detected subspace cluster.

Figure 6(a) shows that runtime scalability depends largely on the degree of redundancy. No redundancy, i.e. $R = 0\%$ shows runtimes far lower than all other variants. Even as little as $R = 20\%$ redundancy in the result set leads to up to four times slower runtimes. Towards full redundancy of $R = 100\%$ runtimes are only slightly better than SUBCLU. Figure 6(b) illustrates the respective F1 quality measurements and the result size on 1900 objects. As we can see from the dark gray bars, the number of subspace clusters increases exponentially (logarithmic scale), meaning overwhelming output size of competing algorithms. The light gray bars are F1 measurements. Clearly, removing redundancy leads to far better F1 values. Redundant subspace clusters thus obscure information in maximal dimensionality subspace clusters. The fully redundant SUBCLU algorithm did not finish after more than 10 days, hence we were not able to evaluate the F1 value for SUBCLU.

On real world data sets (pendigits, vowel, glass [8] and shapes [6]) $F1$ values are given in Figure 7(a). Removing redundancy is important for all four real world data sets as well: noisy clusters are removed from the result set, and precision improves (e.g. $R = 0\%$ results in a precision of 30% for vowel while $R = 5\%$ yields a precision of 24%). Removing all redundant subspace clusters might miss a few cluster objects resulting in slightly lower recall. As illustrated by Figure 7(a) allowing a very small amount of redundant clusters is a good compromise. The effect for runtime behavior is presented in Figure 7(b). The results demonstrate that INSCY outperforms SUBCLU, especially for maximal redundancy pruning ($R = 0\%$).

7 Conclusion

INSCY (indexing subspace clusters with in-process-removal of redundancy) overcomes two major drawbacks of existing subspace clustering approaches: highly redundant results and poor runtimes. We analyzed how in-process removal of redundancy in a depth-first approach permits powerful pruning. Our SCY-tree is the first indexing structure for compact representation of potential subspace cluster regions to avoid repeated database scans. Thorough experiments demonstrate that INSCY clearly outperforms existing subspace clustering algorithms.

Acknowledgments: This research was funded in part by the cluster of excellence on Ultra-high speed Mobile Information and Communication (UMIC) of the DFG (German Research Foundation grant EXC 89).

References

[1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD*, pages 94–105, 1998.

[2] I. Assent, R. Krieger, E. Müller, and T. Seidl. DUSC: Dimensionality unbiased subspace clustering. In *ICDM*, pages 409–414, 2007.

[3] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases. In *KDD*, pages 226–231, 1996.

[4] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD*, pages 1–12, 2000.

[5] K. Kailing, H.-P. Kriegel, and P. Kröger. Density-connected subspace clustering for high-dimensional data. In *SDM*, pages 246–257, 2004.

[6] E. Keogh, L. Wei, X. Xi, S.-H. Lee, and M. Vlachos. LB_Keogh supports exact indexing of shapes under rotation invariance with arbitrary representations and distance measures. In *VLDB*, pages 882–893, 2006.

[7] G. Moise, J. Sander, and M. Ester. P3C: A robust projected clustering algorithm. In *ICDM*, pages 414–425, 2006.

[8] D. Newman, S. Hettich, C. Blake, and C. Merz. UCI repository of MLDBs, 1998.

[9] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, USA, 2005.