

Flexible Fault Tolerant Subspace Clustering for Data with Missing Values

Stephan Günnemann[•] Emmanuel Müller[◦] Sebastian Raubach[•] Thomas Seidl[•]

[•]RWTH Aachen University, Germany
 {guennemann, raubach, seidl}@cs.rwth-aachen.de

[◦]Karlsruhe Institute of Technology, Germany
 emmanuel.mueller@kit.edu

Abstract—In today’s applications, data analysis tasks are hindered by many attributes per object as well as by faulty data with missing values. Subspace clustering tackles the challenge of many attributes by cluster detection in any subspace projection of the data. However, it poses novel challenges for handling missing values of objects, which are part of multiple subspace clusters in different projections of the data.

In this work, we propose a general fault tolerance definition enhancing subspace clustering models to handle missing values. We introduce a flexible notion of fault tolerance that adapts to the individual characteristics of subspace clusters and ensures a robust parameterization. Allowing missing values in our model increases the computational complexity of subspace clustering. Thus, we prove novel monotonicity properties for an efficient computation of fault tolerant subspace clusters. Experiments on real and synthetic data show that our fault tolerance model yields high quality results even in the presence of many missing values. For repeatability, we provide all datasets and executables on our website¹.

I. INTRODUCTION

In today’s applications tremendous amount of data is gathered. In sensor networks, customer segmentation or gene expression analysis, objects are described by many attributes. As one major data mining task clustering tries to detect groups of similar objects while separating dissimilar ones. However, as applications provide more and more attributes, meaningful clusters are hidden in projections of the data that use only subsets of the given attributes. This challenge for cluster detection is tackled by a recent paradigm: *Subspace Clustering* aims at detecting clusters in any possible subspace projection of the data [13], [8], [11]. Each cluster is associated with an individual set of relevant dimensions. For example, a group of customers shows similarity in sport dimensions, while some of these customers group together with other customers due to common interest in music. Since each object can be grouped in multiple clusters due to locally different subspace projections, novel challenges in the handling of missing values arise (cf. Fig. 1).

Intuitively, missing values should be tolerated in such cases where sufficient information is given about the relevant dimensions and the object groupings. Thus, in Fig. 1 for example the object *o4* should be included in both subspace clusters. This general idea is independent of the underlying subspace cluster definition. Unfortunately, it is not supported by traditional techniques that handle missing values.

	d1	d2	d3	d4	d5
o1	low value	low value	low value		
o2	low value	low value	low value		
o3	low value	low value	low value		
o4	low value	low value	???	high value	high value
o5			high value	high value	high value
o6			high value	high value	high value
o7			high value	high value	high value

Fig. 1. Simplified example of a missing value in multiple subspace clusters

Limitations of statistical pre-processing methods. The easiest way to handle missing values is by eliminating objects containing missing values, the so called complete case analysis [9]. It is known that this principle leads to huge information loss and may distort the distribution of the data. As an alternative, imputation techniques were proposed to fill up the missing values. Single imputation techniques such as imputing by means, hot deck imputation, or imputation based on maximizing the observed-data likelihood [19], fill up missing values with a single value. By using just a single value, we would miss one of the hidden clusters in Fig. 1. Subspace clustering, however, would be able to cope with multiple different values depending on the used subspace projection.

Multiple imputation [18], [1] was first introduced by [17] and e.g. applies Monte Carlo techniques to determine the values used for imputation. The general idea of multiple imputation is to generate several (complete) data sets. The missing values are, e.g., based on simulation or drawn from a learned distribution. The used distribution is identical for any of the generated data sets, but in our scenario missing values should be handled based on the current context, i.e. which subspace is currently considered (cf. Fig. 1). A more severe problem of multiple imputation is that we explicitly have to handle several data sets: for each data set subspace clustering has to be performed and the results have to be joined (pooling step). Thus, we need a technique for combining different subspace clustering results to obtain an overall solution. While for traditional clustering ensemble/consensus techniques are available for this task [21], [5], for subspace clustering there exists no such consensus function to combine multiple clustering results. Furthermore, naive computation of independent subspace clustering results on these complete data sets would be highly inefficient. Thus, current subspace clustering techniques cannot handle

¹<http://www.dme.rwth-aachen.de/OpenSubspace/FTSC>

multiple data sets as generated by multiple imputation.

Overall, all statistical imputation techniques act as pre-processing methods that are used before the actual clustering step. Our approach, however, handles missing values during the clustering process. Though the statistical pre-processing techniques have drawbacks, we include complete case analysis and single imputation, both in combination with existing subspace clustering techniques, as competitors in our experimental section.

Missing value handling in related mining paradigms.

While there exist fault tolerant extensions for the (subspace clustering related) mining paradigms of itemset mining [22], [15] and for the bi-clustering/co-clustering related formal concept analysis [14], [4], they are not directly applicable for subspace clustering. All of these approaches were developed for binary data while for subspace clustering we analyze numerical data. Also sets of similar objects are not adequately considered in previous works. For example in fault tolerant itemset mining only the support of an itemset is determined, not its actual object set as in subspace clustering. Only one subspace clustering publication mentions the challenge of missing values [2], but it does not provide a fault tolerant solution. Simply allowing missing values in the relevant dimensions would introduce pseudo clusters not present in the data, while simply discarding missing values as irrelevant dimensions would lose some meaningful clusters.

Our contribution. As contribution of our fault tolerant subspace clustering model we tackle major open challenges in fault tolerance. Overall, we develop:

- A general fault tolerance model handling missing values in subspace clusters
- A flexible and robust notion of fault tolerance adapting to cluster characteristics
- Novel monotonicity properties enabling efficient candidate pruning

We abstract from concrete subspace clustering definitions and propose a general fault tolerance principle applicable to multiple instantiations. Thus, grid-based subspace clustering approaches as CLIQUE [2] and its enhancements [12], [20], [16], paradigms based on the density-based clustering idea [7], [3], and several other definitions can gain a benefit from our approach. In addition to our general model, we present a concrete instantiation to the well-established grid-based subspace clustering.

Fault tolerance introduces three novel challenges in both modeling of subspace clusters and their computation: First, we have to take care that each object can be part of multiple subspace clusters since in different subspaces the objects can be grouped differently. Second, a meaningful fault tolerance has to consider these varying object and attribute characteristics for each subspace cluster individually. Therefore, we introduce a fault tolerance notion that adapts to the characteristics of subspace clusters. To quantify the amount of acceptable missing values we introduce parameters and

we compare different models of thresholds. Third, the exponential search space of all possible subspaces poses a general challenge for subspace clustering. Allowing missing values introduces an additional search space complexity. Thus, novel pruning techniques for our fault tolerant model are essential to enable an efficient computation. We prove monotonicity properties for our novel thresholds and show especially the robustness of our flexible fault tolerance in the empirical evaluation. Overall, these novel challenges go beyond the ones of traditional subspace clustering. We tackle these challenges by our general fault tolerance model, which achieves high clustering quality even in the presence of many missing values.

II. FAULT TOLERANT SUBSPACE CLUSTERING MODEL

In this section, we present the theoretical basis for tackling the challenges of missing value handling in subspace clustering. The basic principle of our model is to permit a cluster to contain a specific amount of missing values. This way, we want to ensure that all hidden clusters are found, although some of the inner structure of the dataset is obfuscated by missing values. We first present our general model and propose a specific instantiation in Section III.

A. Relaxation of subspace cluster definitions

In general, a subspace cluster $C = (O, S)$ is a set of objects $O \subseteq DB$ together with a set of dimensions $S \subseteq Dim = \{1, \dots, n\}$. In the given database, each object $o \in DB$ corresponds to an n -dimensional feature vector $o = (o_1, \dots, o_n)$ such that $DB \subseteq (\mathbb{R} \cup \{?\})^n$ with "?" as a missing value. Instead of requiring similarity in the n -dimensional space, a subspace cluster (O, S) assumes similarity of objects O only w.r.t. a set of dimensions S . Thus, a subspace cluster definition poses similarity constraints only based on these dimensions. Depending on the underlying cluster model one, e.g., uses the Euclidean Distance in subspace S , i.e. $dist_S(o, p) = \sqrt{\sum_{d \in S} (o_d - p_d)^2}$, or poses constraints at each dimension individually by restricting $|o_d - p_d|$ to a maximal deviation for all $d \in S$. All other dimensions $d' \notin S$ are irrelevant for this cluster and do not pose any constraints to the object set O .

Definition 1: Constrained Attribute Values

For a subspace cluster (O, S) a clustering model poses constraints for each object $o \in O$ in all relevant dimensions $d \in S$: o_d is called a constrained attribute value of (O, S) iff $(o, d) \in O \times S$

A missing value would affect a subspace cluster (O, S) if it occurs in one of the constrained attribute values. In general, subspace cluster definitions require all of these constraints to be fulfilled. Thus, they are not able to handle missing values. Our fault tolerance model allows to relax some of these constraints to allow missing values in the relevant dimensions S of clustered objects O . With our fault tolerance model, any traditional subspace cluster definition

can be relaxed to allow missing values. For example, one can use grid-based [2], [12], [20], [16] or density-based [7], [3] cluster definitions. For each missing value one simply assumes to have given the ideal value instantiation inside the subspace cluster. Thus, only the non-missing values have to fulfill the cluster definition while missing values relax the cluster definition. Unfortunately, this naive relaxation to fault tolerance would allow any amount of missing values. Trivial clusters consisting only of missing values would be detected as valid subspace clusters. Our fault tolerance model enhances this loose relaxation. The key idea is to restrict the number of allowed missing values such that the given data is sufficient to infer the missing parts. Hence, the non-missing values of a subspace cluster (O, S) are sufficient to validate the relevant dimensions S and clustered objects O .

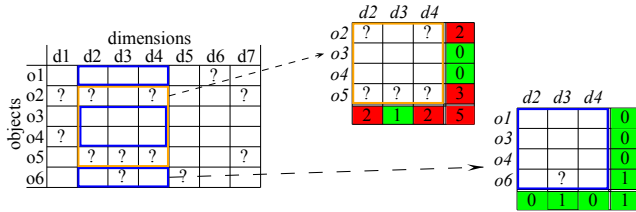


Fig. 2. Database with missing values and two potential subspace clusters

Let us sketch this idea with a toy example: In Fig. 2 a database with missing values is illustrated on the left. On the right, two potential subspace clusters are selected. Both subspace clusters cover the same relevant dimensions but include different objects. The upper cluster is not meaningful since the object 5 has just missing values in the relevant dimensions. Furthermore, the dimensions 2 and 4 have a high amount of missing values for this cluster; the relevance of these dimensions is not well justified. The lower cluster, however, contains just one missing value. Thus, it provides sufficient information. To check this formally, we first introduce a definition to retrieve the amount of missing values per dimension or per object of the cluster.

Definition 2: Missing value set functions

The objects having a missing value in dimension $d \in Dim$ are defined by $O_?(d) = \{o \in DB \mid o_d = ?\}$.

The dimensions of an object $o \in DB$ containing a missing value are defined by $D_?(o) = \{d \in Dim \mid o_d = ?\}$.

These functions return sets based on the full space and the whole database. However, we are only interested in those missing values lying in a potential cluster $C = (O, S)$, i.e. missing values corresponding to the constrained attribute values. Missing values in the irrelevant dimensions $Dim \setminus S$ or contained in the currently not considered objects $DB \setminus O$ do not have any impact on the decision whether (O, S) is a valid cluster or not. Thus, our model considers missing values based on the individual contexts, i.e. which subspace and object set is currently considered. Formally, we

have to intersect the sets introduced in Def. 2 with the (potential) cluster: For the upper cluster in Fig. 2 we get $O_?(3) \cap O = \{5, 6\} \cap \{2, 3, 4, 5\} = \{5\}$ and $D_?(2) \cap S = \{1, 2, 4, 7\} \cap \{2, 3, 4\} = \{2, 4\}$.

B. Novel fault tolerance bounds

The main idea is allowing a cluster to contain a certain amount of missing values, which has a big impact on the clustering result. Allowing no missing values would end up in traditional subspace clustering that misses some clusters due to missing values. Allowing an arbitrary amount of missing values, however, includes pseudo clusters due to the loose model relaxation.

We propose three different characteristics to restrict the allowed number of missing values in a meaningful way:

- (1) Each object of the cluster should not contain too many missing values since otherwise the similarity to the remaining objects is questionable. We denote this property **object tolerance** since it is checked for each object of the cluster.
- (2) Each dimension should not contain too many missing value since otherwise it is questionable whether this dimension is relevant for this cluster. This property is called **dimension tolerance**.
- (3) We restrict the overall number of missing values in our cluster, i.e. by considering the objects and dimensions simultaneously: the **pattern tolerance**.

As first tolerance properties we introduce constant bounds as follows. More enhanced variable bounds are defined in the remainder of this section.

Definition 3: Fault tolerances (constant case)

Given the tolerance thresholds $\varepsilon_o^c, \varepsilon_s^c, \varepsilon_g^c \in \mathbb{N}_0$, a subspace cluster $C = (O, S)$ is

- *object tolerant*, if $\forall o \in O : |S \cap D_?(o)| \leq \varepsilon_o^c$
- *dimension tolerant*, if $\forall d \in S : |O \cap O_?(d)| \leq \varepsilon_s^c$
- *pattern tolerant*, if $\sum_{o \in O} |S \cap D_?(o)| = \sum_{s \in S} |O \cap O_?(s)| \leq \varepsilon_g^c$

By selecting $\varepsilon_o^c=2, \varepsilon_s^c=1, \varepsilon_g^c=4$ in Fig. 2 the upper cluster violates all thresholds while the lower cluster fulfills all of them. If we chose $\varepsilon_g^c = 6$, the pattern tolerance would be fulfilled by the upper one; however, the remaining two tolerances are still violated. Note that none of these properties subsumes one of the others. For example, one might be willing to accept at most 2 missing values per object but at most 10 for the *whole* cluster. If the cluster covers more than 5 objects, some objects must have less than 2 missing values. In this case, the ε_o^c threshold cannot be reached by all objects simultaneously since by the pattern tolerance ε_g^c we take a global view of all objects. Thus, all three thresholds are beneficial.

If a cluster fulfills all three tolerances, a sufficient amount of information can be used to validate the underlying subspace cluster definition. Thus, assuming an ideal setting for

the missing values to obtain a valid grouping is reasonable. Besides this advantage of Def. 3, the drawback is the constant and thus fixed number of permitted missing values. Though, the subspace clusters hidden in the data can differ w.r.t. their number of objects as well as their number of relevant dimension. Working with constant thresholds, however, forces each cluster to keep conditions regardless of its dimensionality and size. Hence, a 4-dimensional cluster is allowed to have the same amount of missing values as a 16-dimensional one. For example, setting $\varepsilon_o^c = 1$ each object is allowed to cover one dimension with missing values. This is meaningful for the 4-dim. cluster but too restrictive for the 16-dim. one.

Formally, this challenge of flexible fault tolerance can be derived out of the number of constrained attribute values (cf. Def. 1). As motivated in the previous paragraph, each dimension poses additional constraints to the objects in a subspace cluster (O, S) . Thus, a constant fault tolerance (cf. Def. 3) would fail with increasing number of dimensions. Similarly, for increasing cluster size the same drawback of constant tolerance can be observed. Overall, more and more constraints have to be fulfilled due to the underlying subspace cluster definition. Hence, the fault tolerance has to adapt w.r.t. the characteristic properties size $|O|$, dimensionality $|S|$ and number of constrained attribute values $|O \times S|$.

We propose using relative thresholds to solve this problem. The key idea is to limit the ratio of missing values and attribute value constraints. Thus, with increasing number of constraints one allows a proportional increase of missing values to be tolerated. Our flexible fault tolerance adapts to the dimensionality and size of a pattern: the characteristics of the cluster are taken into account. For example, a *relative object tolerance* of $1/4$ allows each object of the current subspace cluster $C = (O, S)$ to cover $1/4 \cdot |S|$ dimensions with missing values. Thus, for the 4-dim. cluster at most one missing value per object is possible and for the 16-dim. cluster at most four missing values per object. Thus, relative thresholds are more suitable than constant thresholds.

Definition 4: Fault tolerances (relative case)

Given the tolerance thresholds $\varepsilon_o, \varepsilon_s, \varepsilon_g \in [0, 1]$, a subspace cluster $C = (O, S)$ is

- *object tolerant*, if $\forall o \in O : |S \cap D_?(o)| \leq \varepsilon_o \cdot |S|$
- *dimension tolerant*, if $\forall d \in S : |O \cap O_?(d)| \leq \varepsilon_s \cdot |O|$
- *pattern tolerant*, if $\sum_{o \in O} |S \cap D_?(o)| \leq \varepsilon_g \cdot |O| \cdot |S|$

In Fig. 3 two clusters covering the same objects but with different dimensions are illustrated. While the left one violates the object tolerance property, the right one fulfills all thresholds since in the additionally included dimension 4 the object 3 does not contain a missing value. In contrast to the constant thresholds that ignore the increased proportion of given values, our relative threshold successfully adapts to the included dimension.

Overall, a valid fault tolerant subspace cluster is formal-

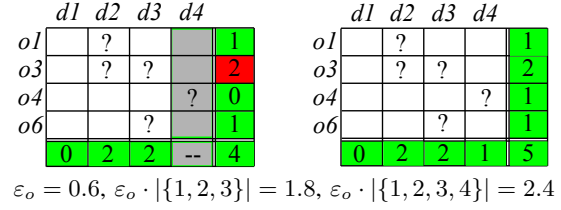


Fig. 3. Relative thresholds: invalid cluster (left) and valid cluster (right) with $\varepsilon_o = \varepsilon_s = 0.6 \varepsilon_g = 0.5$

ized by the following definition.

Definition 5: Valid subspace cluster

A cluster $C = (O, S)$ is called a *valid fault tolerant subspace cluster*, if

- (O, S) fulfills all three tolerance (constant or relative)
- there exists an instantiation of missing values in (O, S) , which fulfills the selected subspace cluster definition (e.g. [2], [12], [7], [3])

Thus, depending on the currently selected cluster and subspace, a missing value can be instantiated differently enabling an object to be part of multiple clusters (cf. Fig. 1). In the following we will focus on the flexible fault tolerance defined by the relative case and only briefly discuss the simpler constant case.

C. Proving monotonicity of fault tolerance

We prove some important monotonicity properties of our model. Most subspace clustering models obey a monotonicity: if (O, S) is a subspace cluster, then in each subspace of S there exists a *superset of objects* such that this set is again a subspace cluster. Formally,

$$(O, S) \text{ is a cluster} \Rightarrow \forall S' \subseteq S : \exists O' \supseteq O : (O', S') \text{ is a cluster} \quad (1)$$

Monotonicity is an essential property for pruning the exponential search space in subspace clustering algorithms. The contrapositive of Eq. (1) is used in a bottom-up processing to prune higher-dimensional subspaces. Unfortunately, as illustrated by the example in Fig. 3, our fault tolerance model does not fulfill a monotonicity property. By considering variable thresholds we defined an enhanced model, however, if (O, S) is a valid subspace cluster this does not imply that (O', S') is one too.

In the following, we derive a monotonicity property for enclosing cluster approximations of our fault tolerant model. These properties can later be used to increase the efficiency of algorithms (e.g. in our instantiation; Sec. III). Let us first compare traditional monotonicity properties with our enclosing approximation: For monotone models, object sets increase in their size with decreasing dimensionality. This aspect is illustrated in Fig. 4 (left) and the property holds for a variety of definitions including [2], [12], [7], [3]. Our fault tolerance model does not obey such a monotonicity.

Thus, we propose an artificial monotonicity based on novel enclosing approximations, which correspond to supersets of the actual clusters (illustrated by rectangles in the figure). Although the clusters do not obey a monotonicity, our enclosing approximations do. In this case, Eq. (1) is not directly valid for the clusters but valid for their enclosing approximations, which are later on refined to generate the actual clusters.

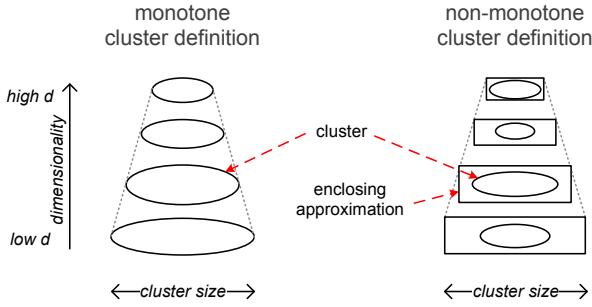


Fig. 4. Monotonicity of clusters and enclosing cluster approximations

In the following, we prove several novel monotonicity properties for our fault tolerant subspace clustering model. To achieve this aim, we first have to develop novel enclosing approximations for fault tolerant subspace clusters, i.e. we have to define supersets of the (truly) clustered objects. Second, we show for these approximations certain monotonicity properties, which can be used for pruning the search space. As a major aspect, we can prove that the monotonicity of Eq. (1) holds for our novel approximations: If there appears no approximation in the subspace S' , then we do not need to analyze any subspace $S \supset S'$. Overall, our fault tolerance model is applicable for several subspace cluster definitions. In our proofs we only assume the underlying subspace cluster model to fulfill Eq. (1). Based on this, we prove that our extension to fault tolerance achieves a monotonicity by enclosing approximations as well.

Let us define our approximation for fault tolerant subspace clusters based on variable thresholds. The definitions for the simpler constant case can be derived respectively. The relative case is more complex since we have to consider the dynamic adaption of the thresholds to the characteristics of the clusters. In higher-dimensional spaces for example, we can permit a larger amount of missing values; thus, the corresponding clusters can potentially contain more objects compared to their lower dimensional counterparts. This aspect has to be considered by a monotone and enclosing cluster approximation.

We derive a so-called mx -approximated subspace cluster. Intuitively, a subspace cluster in subspace S can only be extended by few additional dimensions up to a dimensionality of mx (e.g. $mx = 4$ in Fig. 3). This maximal dimensionality is used to derive our enclosing approximation.

Definition 6: Cluster approximation (relative case)

Each fault tolerant subspace cluster $C = (O, S)$

(w.r.t. $\varepsilon_o, \varepsilon_s, \varepsilon_g$) with dimensionality $|S| \leq mx$ is mx -approximated by a maximal fault tolerant subspace cluster $A = (O_A, S)$ determined by the tolerance thresholds $\varepsilon_o = \min\{\varepsilon_o \cdot \frac{mx}{|S|}, 1\}, \varepsilon_s = \varepsilon_g = 1$.

Theorem 1: The approximation of Def. 6 is enclosing
That is, if $A = (O_A, S)$ is an mx -approximation of the cluster $C = (O, S)$, then $O_A \supseteq O$.

Proof: The approximation is enclosing since all thresholds are relaxed. The dimension and pattern tolerance are set to the maximal values. The former object tolerance ε_o is also replaced by a larger value since for all $|S| \leq mx$ it holds: $\min\{\varepsilon_o \cdot \frac{mx}{|S|}, 1\} \geq \min\{\varepsilon_o \cdot \frac{mx}{mx}, 1\} \geq \varepsilon_o$ \square

The important aspect to be considered is the dependency between the dimensionality of the clusters and the mx -approximation. By selecting mx , the approximations are only correct for clusters up to this maximal dimensionality. Consequently, for a single cluster different approximations based on different mx values are possible. The smaller mx , the tighter is the approximation; however, with a small mx we account only for few subspace clusters since $|S| \leq mx$ is required. By fixing mx , the introduced cluster approximation shows also monotone behavior w.r.t. the object sets.

Lemma 1: Monotonicity of objects

Given an mx -approximation $A = (O_A, S)$ with $|S| \leq mx$, for each $S' \subseteq S$ there exists $O_B \supseteq O_A$ such that $B = (O_B, S')$ is again an mx -approximation.

Proof: For each mx -approximation (O_A, S) in the subspace S we get the object tolerance condition $|S \cap D_{\gamma}(o)| \leq \min\{\varepsilon_o \cdot \frac{mx}{|S|}, 1\} \cdot |S| \Leftrightarrow |S \cap D_{\gamma}(o)| \leq \min\{\varepsilon_o \cdot mx, |S|\}$. Since always $|S \cap D_{\gamma}(o)| \leq |S|$ this can be replaced by $|S \cap D_{\gamma}(o)| \leq \varepsilon_o \cdot mx$. Thus, we get on the right hand side the constant term $\varepsilon_o \cdot mx$ and $O_A \subseteq O_B$ can be proven by contradiction: Assume there exists an object $o \in O_A$ with $o \notin O_B$. o cannot violate the object tolerance in O_B , since $\forall o \in O_A : |S \cap D_{\gamma}(o)| \leq \varepsilon_o \cdot mx \stackrel{S' \subseteq S}{\Rightarrow} |S' \cap D_{\gamma}(o)| \leq \varepsilon_o \cdot mx \Rightarrow$ object tolerance fulfilled. The dimension and pattern tolerance are always fulfilled for a cluster approximation. Consequently, only if o is not a valid element according to the cluster definition we can get $o \notin O_B$. However, we assume Eq. 1 to be true for the cluster definition as proven for e.g. [2], [12], [7], [3]; thus, there exists an O_B such that (O_B, S') fulfills the cluster definition. $\Rightarrow o \in O_B$ and hence $O_A \subseteq O_B$. \square

Besides the monotonicity of the object sets, we prove monotonicity properties of the cluster sizes of the potential clusters represented by the approximations. More precisely, we present *upper bounds* for the cluster sizes (of clusters represented by the approximation) and these bounds are monotone decreasing with increasing dimensionality of the clusters/approximations. Since a cluster (O, S) is a subset of an approximation (O_A, S) , the cardinality $|O_A|$ is of course an upper bound for the size of O ; we develop tighter bounds $b((O_A, S))$. In other words, if we want to generate an actual

cluster (O, S) based on the approximation (O_A, S) we can ensure that $|O| \leq b((O_A, S))$ in addition to $O \subseteq O_A$. For the first bound we assume $\varepsilon_s < \frac{1}{|S|}$ (otherwise the naive bound $|O_A|$ still could be used for pruning).

For the definition of these bounds let us first provide some basic notions: Let $obj(O, S, x) \subseteq O$ be the set of objects with exactly x missing values in the dimensions S . Formally, $obj(O, S, x) = \{o \in O \mid |S \cap D_?(o)| = x\}$. And $greedy(O, S, y)$ the largest subset of O with in total y (or less) missing values (break ties arbitrarily). Formally, $greedy(O, S, y) = \text{argmax}_{M \subseteq O} \{ |M| \mid \sum_{i=0}^{|S|} |obj(M, S, i)| \cdot i \leq y \}$. This set can efficiently be determined by a greedy approach, selecting the objects in increasing order of their missing values until y is exceeded.

Lemma 2: Monotonicity of cluster sizes (ε_s)

Given an m -approximation $A = (O_A, S)$, the size of each valid cluster represented by the approximation is bounded by $b_s((O_A, S)) = |obj(O_A, S, 0)| \cdot \frac{1}{1 - |S| \cdot \varepsilon_s}$. And for each $S' \subseteq S$ the corresponding m -approximation $B = (O_B, S')$ with $O_B \supseteq O_A$ fulfills: $b_s((O_B, S')) \geq b_s((O_A, S))$.

Proof: In each dimension $\varepsilon_s \cdot |O|$ missing values are permitted; thus given the clustered objects O , at most $|S| \cdot \varepsilon_s \cdot |O|$ objects containing missing values can be selected because otherwise at least one dimension exceeds the threshold. By selecting objects without missing values we increase the potential for selecting objects with missing values. Thus the size of the clustered objects is in the best case given by $|O| = |obj(O_A, S, 0)| + |S| \cdot \varepsilon_s \cdot |O|$. Reformulating, such that $|O|$ appears just on one side of the equation, yields the desired upper bound.

W.l.o.g. $S' \cup \{x\} = S$; the approximation O_A contains at most $\varepsilon_s \cdot |obj(O_A, S, 0)| \cdot \frac{1}{1 - |S| \cdot \varepsilon_s} := k$ objects with missing values in dimension x ; for the best case we have to assume that these k objects have no missing values in the other dimensions, thus they are included in $obj(O_B, S', 0)$. Since obviously also $obj(O_B, S', 0) \supseteq obj(O_A, S, 0)$ holds, we get $|obj(O_B, S', 0)| \cdot \frac{1}{1 - |S'| \cdot \varepsilon_s} \geq (|obj(O_A, S, 0)| + k) \cdot \frac{1}{1 - (|S|-1) \cdot \varepsilon_s} = |obj(O_A, S, 0)| \cdot \frac{1}{1 - |S| \cdot \varepsilon_s} \cdot \frac{2 - |S| \cdot \varepsilon_s}{1 + \varepsilon_s - |S| \cdot \varepsilon_s} \geq |obj(O_A, S, 0)| \cdot \frac{1}{1 - |S| \cdot \varepsilon_s}$ \square

Lemma 3: Monotonicity of cluster sizes (ε_g)

Given an m -approximation $A = (O_A, S)$ with $|S| \leq m$, the size of each valid cluster represented by the approximation is bounded by $b_g((O_A, S)) = |greedy(O_A, S, \varepsilon_g \cdot |O_A| \cdot m)|$. And for each $S' \subseteq S$ the m -approximation $B = (O_B, S')$ with $O_B \supseteq O_A$ fulfills: $b_g((O_B, S')) \geq b_g((O_A, S))$.

Proof: In the best case O_A itself is a valid cluster; thus, the pattern tolerance would permit at most $\varepsilon_g \cdot |O_A| \cdot |S| \leq \varepsilon_g \cdot |O_A| \cdot m$ missing values \Rightarrow the size of the greedy set is an upper bound.

Since $O_B \supseteq O_A$ and $\varepsilon_g \cdot |O_B| \cdot m \geq \varepsilon_g \cdot |O_A| \cdot m$ the

monotonicity holds obviously. \square

Overall, also the relative thresholds enable to generate monotone behaving cluster approximations and we can prove important bounds for the cluster sizes. In the following, these properties are used for our instantiation of the model but are of course not restricted to this instantiation. They enable in general an efficient processing for the constant threshold case as well as for the relative threshold case.

III. INSTANTIATION AND FTSC ALGORITHM

In this section, we use the general model presented in Sec. II to extend traditional subspace clustering. To focus on the handling of missing values, we decided to keep the cluster definition as simple as possible. Further extensions to more complex clustering models are intended for future work. Our method is called FTSC (fault tolerant subspace clustering).

A. Subspace cluster definition

For our cluster definition we extend the widely used grid-based subspace clustering paradigm, e.g. used by CLIQUE [2] or SCHISM [20]. The idea is to discretize the search space by partitioning the data space into non-overlapping rectangular grid-cells. Cells are obtained by partitioning every dimension into a given number of equal-length intervals and intersecting intervals from each dimension. A grid-cell is dense and hence a cluster, if it contains more than a given number of objects. Assuming the database to be normalized between 0.0 – 1.0 and using g intervals per dimension, a cluster $C = (O, S)$ is defined by

$$\forall o, p \in O : \forall d \in S : [o_d \cdot g] = [p_d \cdot g], \text{ i.e. } O \text{ contains the objects of a } S\text{-dimensional cell, and } |O| \geq \text{minPts,} \\ \text{i.e. the grid cell is dense}$$

To extend this approach to fault-tolerance, we additionally manage objects containing missing values. Thus, each dimension d gets an additional interval containing those objects having no value in d . Besides the objects within the same cell we have to consider the missing value-intervals per dimension to get the cluster approximations. Each approximation (O_A, S) is the intersection of those sets for each dimension $d \in S$, which are obtained by the union of the one-dimensional value interval with the missing value interval. After this, the objects with too many missing values can be removed (cf. Def. 6). Thus, a cluster approximation for a cluster at cell $(2, 1)$ consists of the objects located in the cells $(2, 1)$, $(2, ?)$, $(?, 1)$ and $(?, ?)$. Assuming e.g. at most one missing value is allowed we can immediately reject the cell $(?, ?)$. Furthermore, we use the redundancy model introduced in [3] to exclude redundant clusters and to avoid an overwhelming result size.

B. Algorithmic processing

The general processing of FTSC is depicted in Algorithm 1. For generating the clusters we use a depth-first search through the subspace lattice (starting with low-dimensional subspaces going down to higher dimensional ones; line 11-14,20-22). Advantageous in using a depth-first search is the elimination of low-dimensional redundant clusters [3]: While descending the search tree we only generate the cluster approximations (line 5). The actual clusters are generated during the recursive ascension of the search tree; however, only if the previously generated approximation is not already redundant to the identified clusters (line 15,16). Thus, in many cases we only have to generate the cluster approximations and not the clusters while traversing the subspaces.

For an efficient generation of cluster approximations, we use an enhanced representation of the corresponding object sets O_A : we perform a partitioning of this set according to the amount of missing values per object. An approximation (O_A, S) is represented by

$$(O_A, S) = ([O_A^0, O_A^1, \dots, O_A^x], S)$$

where $O_A^i \subseteq O_A$ denotes the objects having exactly i missing values in S . Please note that x is upper bounded by Def. 6, i.e. objects with too many missing values are immediately rejected. The advantage of this representation is the efficient calculation of approximations with increasing dimensionality as needed for a depth-first processing. Given the objects I of an one-dimensional interval for the dimension d and the objects $I_?$ that exhibit a missing value in the same dimension (line 3-4), then the extension of the cluster approximation $([O_A^0, O_A^1, \dots, O_A^x], S)$ to the higher-dimensional subspace $S' = S \cup \{d\}$ can be obtained by

- $O_B^0 = (O_A^0 \cap I)$
- $O_B^i = (O_A^i \cap I) \cup (O_A^{i-1} \cap I_?), 0 < i < x$
- $O_B^{x+1} = (O_A^x \cap I_?)$

generating $([O_B^0, \dots, O_B^{x+1}], S')$. This efficient intersecting is possible due to the monotonicity of the object sets as proven in Lem. 1 and utilized in line 5. The new approximation (O_B, S') has one additional set in contrast to the old one since the number of missing values for each object can increase at most by one. However, the set with the highest number of missing values is potentially rejected based on Def. 6 (line 6).

Pruning of cluster approximations. To avoid analyzing all exponential many subspaces, our method uses the monotonicity properties introduced in Sec. II-C for pruning the search space. First, we exploit the monotonicity of the object sets itself since the approximations are obtained by intersecting with previously obtained ones (subset property; line 5).

Based on the monotonicity of the cluster sizes we can further increase the efficiency of FTSC. Since the cluster sizes are bounded by two different equations, we can reject

Algorithm 1 FTSC algorithm

function: *traverseSubspaces*(\dots)

input: previous approximation (O_A, S) , current dimension d , maximal reachable dimensionality mx

```

1:  $S' \leftarrow S \cup \{d\}$  // current subspace
2: for each( interval  $i$  of dimension  $d$  )
3:    $I \leftarrow \text{getObjects}(i,d)$  // objects of this interval
4:    $I_? \leftarrow \text{getObjects}(?,d)$  // objects with miss. vals. in  $d$ 
5:    $(O_B, S') \leftarrow \text{newApprox}((O_A, S), I, I_?)$  // Lemma 1
6:   prune objects with too many miss. vals. // Definition 6
7:   if  $|O_B| < \text{minPts}$  ) continue; // next interval
8:    $\text{bound}_s \leftarrow \text{calc. bound of cluster size}$  // Lemma 2
9:    $\text{bound}_g \leftarrow \text{calc. bound of cluster size}$  // Lemma 3
10:  if  $\text{bound}_s < \text{minPts} \vee \text{bound}_g < \text{minPts}$  ) continue;
    // next interval
11:  for( dimension  $d'$  from  $d + 1$  to  $|Dim|$  ) // depth-first
12:    // maximal dimensionality of reachable clusters
13:     $mx' \leftarrow |S'| + |Dim| - d' + 1$ 
14:    traverseSubspaces $((O_B, S'), d', mx')$ 
15:  if  $(O_B, S')$  is not redundant to  $C \in \text{Result}$  ) // cf. [3]
16:     $(O, S') \leftarrow \text{generateActualCluster}((O_B, S'))$ 
17:    if  $|O| < \text{minPts}$  ) continue; // next interval
18:    if  $(O, S')$  is not redundant to  $C \in \text{Result}$  ) // cf. [3]
19:       $\text{Result} \leftarrow \text{Result} \cup \{(O, S')\}$ 

```

function: *main*()

```

20: for( dimension  $d$  from 1 to  $|Dim|$  )
21:    $mx' \leftarrow |Dim| - d + 1$ 
22:   traverseSubspaces $((DB, \emptyset), d, mx')$ 

```

an approximation if one of these bounds is smaller than minPts ; we cannot get a valid cluster anymore. Because the bounds are monotone decreasing with increasing dimensionality, all approximations in higher-dimensional subspaces can also not yield valid clusters; we prune the whole subtree (line 8-10).

When using relative thresholds, the following improvement can be realized: At the beginning of the depth-first search we have to assume that valid clusters in the full-dimensional space exist, i.e. we potentially have to traverse the tree down to the subspace Dim . Since the mx -approximations are linked to the dimensionality of the clusters, we have to start with $mx = |Dim|$ to get a correct approximation. However, if the highest dimensional subspace was already processed or the subtree was pruned beforehand, we can refine the maximal realizable dimensionality. Thus, the value of mx can be decreased and we get tighter approximations of the clusters, which lead to better estimations for the sizes and hence more effective pruning (line 13,21). Overall, we use several pruning techniques to increase the efficiency of our FTSC algorithm.

Summary. We presented an instantiation of our fault tolerant model in grid-based subspace clustering; called FTSC. We used DFS along with powerful pruning techniques to search for cluster candidates. In the following, we compare FTSC to traditional subspace clustering that does not handle missing values.

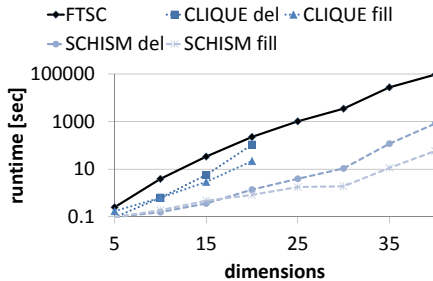


Fig. 5. Runtime vs. dimensions

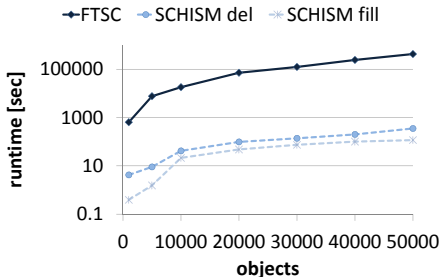


Fig. 6. Runtime vs. database size

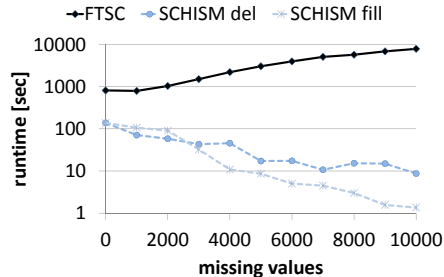


Fig. 7. Runtime vs. missing values

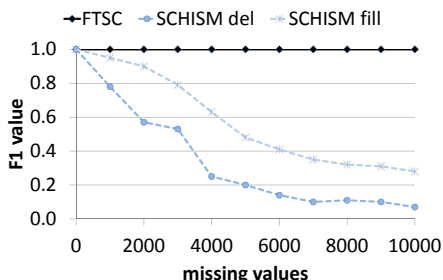


Fig. 8. Missing values distributed randomly

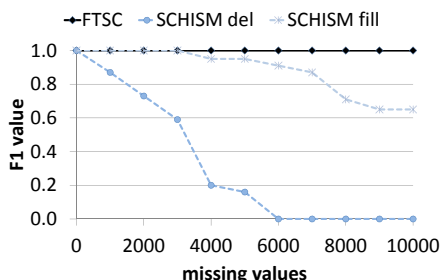


Fig. 9. Miss. val. distributed outside clusters

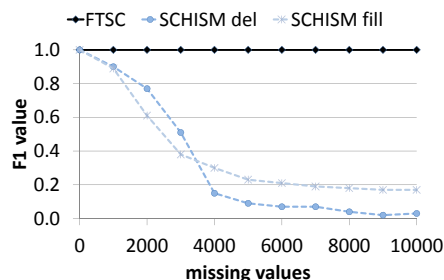


Fig. 10. Miss. val. distributed inside clusters

IV. EXPERIMENTS

Setup. As there are no competing subspace clustering approaches that handle missing values, we compare FTSC with methods working on (complete) data, cleaned by statistical pre-processing techniques. In one case we use complete case analysis and in the second case mean value imputation. To ensure a fair comparison, we apply the grid-based clustering methods CLIQUE and SCHISM on these data since FTSC is also grid-based. The number of grid cells and the density parameters are identically chosen for each method. We generate synthetic data with in general 5000 objects, 30 dimensions and 10 clusters by the method used in our evaluation study [11]. Each cluster was hidden in 50-60% of the overall dimensions. As real world data we use data of the UCI archive [6] (pendigits, glass, breast cancer, vowel) and features extracted from sequence data (shape). We measure clustering quality via the F1 value, similar used in [10], [11]. F1 measures the harmonic mean of recall (are all objects and dimensions detected?) and precision (how accurate is the cluster detected?). For real world data class labels are used as indicators for the natural grouping of objects. All experiments were conducted on Opteron 2.3GHz CPUs and Java6 64 bit.

Scalability. In our work we focus on the quality of FTSC; nonetheless, we briefly analyze the efficiency of the approach. In Fig. 5 the dimensionality of the data, containing 5% missing values, is varied. All methods show an increase in their runtime. Not surprisingly, the runtime of FTSC is the highest since it has to analyze the complex missing values while the others simply work on pre-processed data. However, as we will analyze in the last experiment, our pruning methods effectively lower the runtimes and lead to

an applicability of fault tolerance in practical applications. An important result is that CLIQUE is not applicable for high dimensionalities; the number of generated clusters is too large. Thus, in the remaining experiments we focus on FTSC and SCHISM. A similar trend can be observed in Fig. 6 where the number of objects is varied. However, for this setting the increase in runtime is only moderate.

Most important is the runtime behavior with increasing number of missing values (Fig. 7). The amount of missing values spread inside the dataset defines how much of the original inner structure is still present. The more missing values are spread, the less certain information is given. Therefore, we randomly add missing values to the data. FTSC shows a slight increase in runtime since with increasing number of missing values the cluster approximations get larger and pruning is more difficult. Overall, the runtime of FTSC is higher than SCHISM; however, this effort is worthwhile since it is accompanied by a high clustering quality as analyzed in the following experiments.

Clustering quality. In Fig. 8 the same data as in Fig. 7 is used. As shown, FTSC gets nearly perfect quality even with a high amount of missing values in the data. Our novel method based on relative thresholds countervails the errors in the data. For the setting with 0 missing values, the competing approaches get the same qualities. However, the quality of the complete cases analysis (*del*) drops heavily due to the high information loss. By filling up the missing values (*fill*), this effect can be slightly weakened; however, the errors that come along with the pre-processing eventually get too large, leading to poor quality. Overall, our FTSC method is the only choice for a flexible handling of missing values.

While in Fig. 8 the missing values are randomly dis-

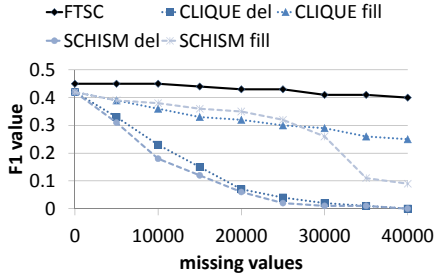


Fig. 11. Quality on pendigits

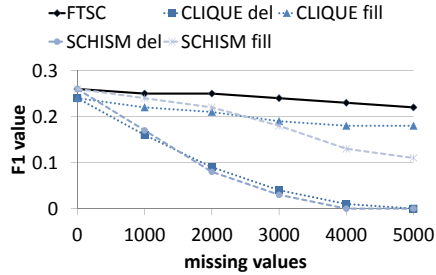


Fig. 12. Quality on vowel

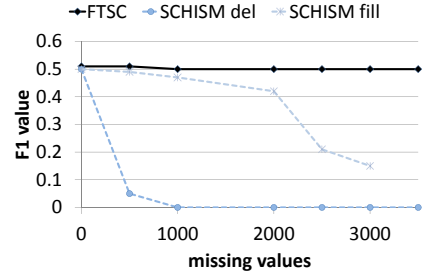


Fig. 13. Quality on breast

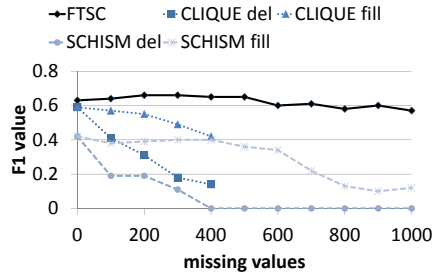


Fig. 14. Quality on shape

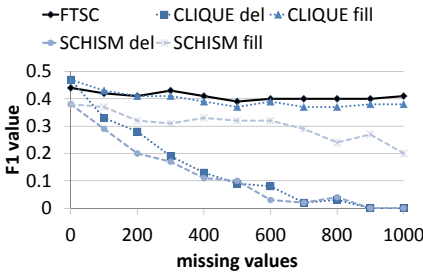


Fig. 15. Quality on glass

<i>data set</i> (# miss. val)	<i>pendigits</i> (10000)	<i>vowel</i> (3000)	<i>breast</i> (2000)	<i>shape</i> (400)	<i>glass</i> (400)
FTSC	3340	187	203	1544	141
SCHISM del	313	1	109	3	47
SCHISM fill	797	140	156	16	15
CLIQUE del	593	94	-	405	1063
CLIQUE fill	1562	141	-	46	875

Fig. 16. Runtime [msec] for real world data

tributed within the data, in Fig. 9 we add them only to the dimensions of objects not covered by the hidden clusters. Thus, we do not obfuscate the hidden clusters with missing values. Again, our FTSC gets the highest quality while the delete approach drops heavily. Under this setting SCHISM-fill can keep up with FTSC until 4000 missing values. Since the missing values are distributed outside the clusters, they can be recovered; however, with a huge amount of missing values artificial clusters appear due to the pre-processing.

This observation is confirmed in Fig. 10 where the data is generated the other way round: the missing values are distributed within the objects and dimensions of the hidden clusters. As expected, both SCHISM methods cannot handle this scenario; the clusters cannot be detected even with few missing values. Our FTSC is able to detect the hidden clusters even with large numbers of missing values.

Real world data. In the following experiments depicted in Fig. 11-15 we analyze the clustering quality for real world data. We increase the number of randomly distributed missing values to analyze the methods' robustness to faults in the data. For all datasets the following observations become apparent: By adding no missing values, the qualities of all approaches are nearly identical. The small differences can be explained by slightly different clustering definitions. Overall, our FTSC achieves the highest clustering qualities and shows robust behavior with increasing number of missing values. Even for a huge amount of missing values the quality is high and only for some datasets a small decrease can be observed. The methods based on pre-processing show a stronger decrease of their clustering qualities. Especially, the deletion methods are consistently worse than the methods based on filling up missing values by mean value imputation. In Fig. 16 the runtimes of the methods for an exemplary

number of missing values is depicted. The efficiency of all approaches is high. Summarized, our FTSC shows good runtimes on real world data and gets highest clustering qualities even if the data is polluted by a huge amount of missing values.

Model analysis. At last, we analyze some characteristics of our approach. First we highlight the difference by using constant or relative thresholds in Fig. 17. On the left we vary the relative threshold ε_o and determine the corresponding clustering qualities. As shown we identify all hidden clusters within the data and this can be realized by a wide range of parameter settings. The parameter is very robust and only for the extreme cases (near 0 or 1) the quality drops. On the other hand, by using constant thresholds (right figure, ε_o^c is the absolute number of allowed missing values) we cannot get a perfect clustering. Moreover, a good clustering is only obtained for a single parameter setting. This experiment confirms that using constant thresholds is not meaningful for fault tolerant clustering approaches; instead our flexible fault tolerance should be used. Relative thresholds are very robust and are easily interpretable since they directly correspond to the fraction of missing values the user is willing to accept.

In Fig. 18 we analyze the efficiency gain of our pruning methods on the pendigits data. We proved in Sec. II-C different monotonicities based on the three types of tolerances/thresholds. These properties are used in the FTSC algorithm for pruning the search space. In Fig. 18 these pruning techniques are turned on or off. Consequently, the highest runtime is obtained by using no pruning, also setting up the baseline runtime (100% runtime; individually for the relative and constant case). Using just a single pruning method already yields a decrease of the runtime. For this dataset the pruning based on ε_s performs best but

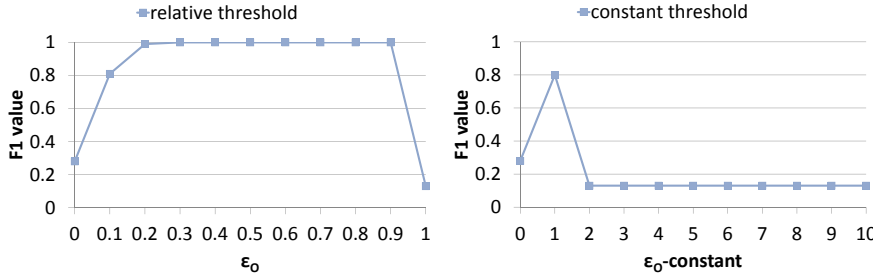


Fig. 17. Relative vs. constant thresholds (clustering quality)

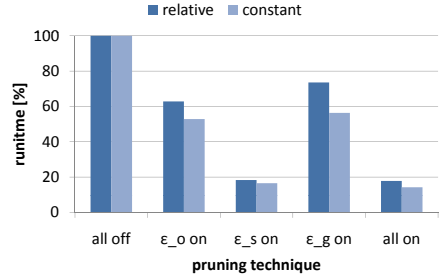


Fig. 18. Effectivity of pruning methods

for other data different results were obtained. However, by using all techniques simultaneously (rightmost bars), we always achieve the lowest runtime. On average, we achieve a runtime improvement of about 60-90% for all datasets. By using all the information derived in Sec. II-C, our FTSC can efficiently determine a subspace clustering result even in the complex scenario of missing values.

V. CONCLUSION AND FUTURE WORK

In this paper, we proposed a method for handling missing values in the context of subspace clustering. We introduced a general fault tolerance model to handle missing values that can be applied to multiple subspace cluster definitions. Our model handles missing values based on the currently considered subspace and set of objects, thus enabling objects to be part of multiple subspace clusters. With our variable thresholds we realize a flexible model that adapts to the individual cluster characteristics and that ensures a robust parameterization. The increased complexity of fault tolerant subspace clustering demands for effective pruning methods. Thus, we proved important monotonicity properties of our general model. As an instantiation of our general model we implemented a fault tolerant grid-based method. In the experimental evaluation, this approach yields high quality results on real and synthetic data even in the presence of many missing values.

As our future work we will address extensions of our fault tolerance model to density-based and other subspace clustering paradigms. Furthermore, the development of ensemble techniques for subspace clustering is a promising step and enables us to apply the existing multiple imputation techniques to handle missing values.

Acknowledgment. This work has been supported by the UMIC Research Centre, RWTH Aachen University, Germany.

REFERENCES

- [1] M. Aerts, G. Claeskens, N. Hens, and G. Molenberghs, "Local multiple imputation," *Biometrika*, vol. 89, no. 2, pp. 375–388, 2002.
- [2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, "Automatic subspace clustering of high dimensional data," *DMKD Journal*, vol. 11, pp. 5–33, 2005.
- [3] I. Assent, R. Krieger, E. Müller, and T. Seidl, "INSCY: Indexing subspace clusters with in-process-removal of redundancy," in *ICDM*, 2008, pp. 719–724.
- [4] J. Besson, C. Robardet, and J.-F. Boulicaut, "Mining a new fault-tolerant pattern type as an alternative to formal concept discovery," in *ICCS*, 2006, pp. 144–157.
- [5] S. Bickel and T. Scheffer, "Multi-view clustering," in *ICDM*, 2004, pp. 19–26.
- [6] A. Frank and A. Asuncion, "UCI machine learning repository," 2010. [Online]. Available: <http://archive.ics.uci.edu/ml>
- [7] K. Kailing, H.-P. Kriegel, and P. Kröger, "Density-connected subspace clustering for high-dimensional data," in *SDM*, 2004, pp. 246–257.
- [8] H.-P. Kriegel, P. Kröger, and A. Zimek, "Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering," *TKDD*, vol. 3, no. 1, pp. 1–58, 2009.
- [9] R. Little and D. Rubin, *Statistical analysis with missing data*. Wiley-Interscience, 2002.
- [10] G. Moise, J. Sander, and M. Ester, "P3C: A robust projected clustering algorithm," in *ICDM*, 2006, pp. 414–425.
- [11] E. Müller, S. Günemann, I. Assent, and T. Seidl, "Evaluating clustering in subspace projections of high dimensional data," *PVLDB*, vol. 2, no. 1, pp. 1270–1281, 2009.
- [12] H. Nagesh, S. Goil, and A. Choudhary, "Adaptive grids for clustering massive data sets," in *SDM*, 2001.
- [13] L. Parsons, E. Haque, and H. Liu, "Subspace clustering for high dimensional data: a review," *SIGKDD Explorations*, vol. 6, no. 1, pp. 90–105, 2004.
- [14] R. G. Pensa and J.-F. Boulicaut, "Towards fault-tolerant formal concept analysis," in *AI*IA*, 2005, pp. 212–223.
- [15] A. K. Poernomo and V. Gopalkrishnan, "Towards efficient mining of proportional fault-tolerant frequent itemsets," in *KDD*, 2009, pp. 697–706.
- [16] C. Procopiuc, M. Jones, P. Agarwal, and T. Murali, "A monte carlo algorithm for fast projective clustering," in *SIGMOD*, 2002, pp. 418–427.
- [17] D. Rubin and I. ebrary, *Multiple imputation for nonresponse in surveys*. Wiley Online Library, 1987, vol. 519.
- [18] J. Schafer, *Analysis of incomplete multivariate data*. Chapman & Hall/CRC, 1997, vol. 72.
- [19] J. Schafer and J. Graham, "Missing data: Our view of the state of the art." *Psychological methods*, vol. 7, no. 2, pp. 147–177, 2002.
- [20] K. Sequeira and M. Zaki, "SCHISM: A new approach for interesting subspace mining," in *ICDM*, 2004, pp. 186–193.
- [21] A. Strehl and J. Ghosh, "Cluster ensembles — a knowledge reuse framework for combining multiple partitions," *Journal of Machine Learning Research*, vol. 3, pp. 583–617, 2002.
- [22] C. Yang, U. Fayyad, and P. S. Bradley, "Efficient discovery of error-tolerant frequent itemsets in high dimensions," in *KDD*, 2001, pp. 194–203.