A Monte Carlo Algorithm for Fast Projective Clustering [*]

Cecilia M. Procopiuc
AT&T Research Laboratory
Florham Park, NJ 07932
magda@research.att.com

Michael Jones[†]
Mitsubishi Electric Research Laboratory
Cambridge, MA 02139
mjones@merl.com

Pankaj K. Agarwal
Department of Computer Science
Duke University
Durham, NC 27708
pankaj@cs.duke.edu

T. M. Murali [†]
Bioinformatics Program
Boston University
Boston, MA 02215
murali@bu.edu

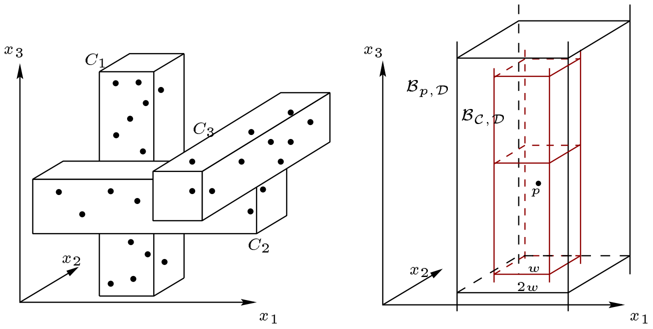
ABSTRACT

We propose a mathematical formulation for the notion of optimal projective cluster, starting from natural requirements on the density of points in subspaces. This allows us to develop a Monte Carlo algorithm for iteratively computing projective clusters. We prove that the computed clusters are good with high probability. We implemented a modified version of the algorithm, using heuristics to speed up computation. Our extensive experiments show that our method is significantly more accurate than previous approaches. In particular, we use our techniques to build a classifier for detecting rotated human faces in cluttered images.

1. PROJECTIVE CLUSTERING

Clustering is a widely used technique for data mining, indexing, and classification. Many practical methods proposed in the last few years, such as CLARANS [11], BIRCH [15], DBSCAN [5, 6], and CURE [7], are "full-dimensional," in the sense that they give equal importance to all the dimensions when computing the distance between two points. While such approaches have proven successful for low-dimensional datasets, their accuracy and/or efficiency decrease significantly in higher dimensional spaces (see [9] for an excellent analysis and discussion). The reason for this performance deterioration is the so-called dimensionality curse. Recent research shows that for moderate-to-high dimensional spaces (tens or hundreds of dimensions), a full-dimensional distance is often irrelevant, as the farthest neighbor of a point is expected to be almost as close as its nearest neighbor [8].

Methods such as Principal Component Analysis (PCA) reduce the dimensionality of the data by projecting all points on a subspace so that the information loss is minimized. A standard clustering method is then used in this subspace. However, PCA does not handle well those situations when different subsets of the points lie on different lower-dimensional subspaces. For the example in Figure 1(a), any attempt to reduce the dimensionality of all the points results in significant information loss, and a full-dimensional clustering technique like k-means is unlikely to discover the three patterns in the data.



x_3
C_1
C_3
C_2
x_2
x_1
x_3
$\mathcal{B}_{p,\mathcal{D}}$
$\mathcal{B}_{\mathcal{C},\mathcal{D}}$
p
x_2
w
$2w$
x_1

Figure 1: (a) Three subspace patterns; (b) Boxes corresponding to a projective cluster $(\mathcal{C}, \mathcal{D})$.

Recognizing the need for increased flexibility in reducing the data dimensionality, recent database research has proposed computing projective clusters, in which points that are closely correlated in some subspace are grouped together. Instead of projecting the entire dataset on a single subspace, these methods project each cluster on its associated subspace, which is generally different from the subspace associated with another cluster. Projective clustering algorithms have been successfully used for indexing [4], as well as pattern discovery in moderately high-dimensional data [1, 2]. All these previous approaches are partitioning methods, i.e. they divide the data into k clusters and an outlier set, and iteratively improve the quality of the clustering. However, no formal definition is given as to what constitutes an optimal k-clustering, and there is no guarantee on the quality of the output. As with most partitioning methods, these approaches also suffer from the inherent

[†] Author did this research when he was associated with the Compaq Research Lab.

[*] The work of the first and third authors is supported in part by National Science Foundation research grants CCR–9732287 and EIA–9870724, by Army Research Office MURI grant DAAH04–96–1–0013, by an NYI award, by a Sloan fellowship, and by a grant from the U.S.-Israeli Binational Science Foundation.

ACM SIGMOD '2002 June 4-6, Madison, Wisconsin, USA
Copyright 2002 ACM 1-58113-497-5/02/06 ...$5.00.

necessity of having the user provide the number of clusters k. This problem is less severe in indexing applications, where the choice of k is driven by outside considerations such as the desired tree fanout, and where small variations of k are unlikely to affect the index performance in a significant way. However, when the goal is to discover patterns in the data, even increasing or decreasing k by 1 can generate very different outputs. In such cases, the clustering method must be called a few times with different values of k, until the output is deemed "accurate" enough (either with respect to a quality measure, or by a human expert).

Our Contributions. In this paper we propose a mathematical definition for the notion of optimal projective cluster, starting from natural requirements on the density of points in subspaces. This allows us to develop a Monte Carlo algorithm that computes, with high probability, a good approximation of an optimal projective cluster. We call our algorithm DOC, from Density-based Optimal projective Clustering.

Density-based approaches have been used before, either for full-dimensional clustering [10], or for enumerating all the dense subspace regions in the data [3]. However, these methods have exponential dependence on the dimensionality, largely due to the fact that they use regular grids in order to find and connect dense areas. The number of relevant grid cells is, in general, exponential in the number of dimensions. To alleviate this problem, a recent technique called OptiGrid [9] uses irregular grids determined by hyperplanes that pass through areas of small density, in order to compute dense clusters. Since we are interested in projective clusters, we take a slightly different view, in that we require each cluster to be dense only in its corresponding subspace. As we will see in the next section, our density conditions refer only to the number of points that project inside an interval of given length, and do not make any assumption on the distribution of points. In our experiments we use synthetic data generated with various subspace distributions and show that our algorithm maintains high accuracy on all the sets.

Iterative clustering versus global partitioning. Once we have an efficient method for generating a provably good projective cluster, we iterate the algorithm in a greedy manner[1], until some termination criterion is met. During each iteration, we compute an approximation of an optimal cluster over the current set of points. The termination criterion can be defined in more than one way, e.g.: a certain percentage of the points have been clustered; or a user-specified number of clusters have been computed; or the quality measure of the clusters has decreased too much. Hence, by contrast to partitioning methods, the user need not specify the number of clusters k unless he wants to. This allows more flexibility in tuning the algorithm to the particular application that uses it.

Moreover, we can either require clusters to be disjoint, or allow them to have common points. In the first case, points that are clustered are eliminated from subsequent computations of clusters. In the second case, clustered points are not eliminated, and we simply check that we do not generate the same cluster twice. In the experimental section we discuss an application from image processing in which we found that overlapping clusters produce better results.

One particularly desirable property of our method is that it is accurate even when the cluster sizes vary significantly (in terms of number of points). Many partitioning methods rely on random sampling for computing an initial partition. Hence, small clusters are likely to be missed. As a result, their points are either assigned to other clusters, or declared outliers. We have observed this behavior in the experiments we conducted using the PROCLUS [1] and ORCLUS [2] algorithms. By contrast, the DOC algorithm does not miss small clusters. Although it is likely that it discovers only large clusters during the first iterations, once these clusters are eliminated from the current dataset the smaller clusters become large with respect to the remaining points, and will thus be discovered in subsequent iterations. In addition, our method handles outliers relatively easy. Since, during each iteration, we compute a projective cluster that approximates the optimal one, outliers tend to remain unclustered. We note that outlier handling is not an easy task in most partitioning algorithms, which use various heuristics in order to distinguish between cluster points and outliers.

We conclude that iterative clustering has significant advantages over partitioning methods. [2] Any greedy clustering method requires two main steps:

1. Define what an optimal cluster is. Since clusters are discovered one at a time, the extent to which this definition models the "natural" clusters in the data determines the quality of the result.

2. Design a fast and accurate method for computing one such optimal cluster, or a good approximation for it.

Our main contribution is to propose new solutions for these steps in the context of projective clustering. Because we provide a rigorous mathematical definition in step 1, we are able to give guarantees on the quality of the result computed by the method in step 2. Our experimental results, both on real and synthetic data, indicate that our definition of an optimal projective cluster is a good way of modeling subspace patterns. We restrict our attention to the case when all projections are along coordinate axes. This restriction is natural in many practical applications, in which coordinate axes have special meaning. For example, in a database of employees, one axis may represent the salary, another the length of employment with the company, and a third one the employees' age. Discovering a projective cluster in the subspace spanned by salary and employment length has the following interpretation: there is a correlation between salaries in range A and years of employment in range B, which is independent of employees' age. In fact, we provide experimental evidence that axis-parallel projective clusters are useful for another interesting dataset which consists of gray scale images (see below). However, it is not difficult to envision applications in which arbitrarily oriented projective clusters are more effective than axis-parallel ones. We intend to study this problem in our future work. We view our current results as a promising step in the direction of developing fast, provably accurate, and stable projective clustering methods.

Application to image processing. We apply our clustering ideas to the problem of detecting rotated human faces in cluttered images. Our database consists of low resolution gray scale images of 16×16 pixels. All images represent human faces in frontal view, roughly aligned with the image boundaries; see Figure 11. In addition, each face is rotated in-plane by one of 15 different rotation angles, resulting in 16 images that contain the same face tilted at various degrees from the vertical. We map each image i to a 256-dimensional point p^i, so that each dimension corresponds to

[1] In this paper, we use the terms *iterative clustering* and *greedy clustering* interchangeably.

[2] Another class of methods consists of hierarchical clustering algorithms. However, the cost of such algorithms is usually quadratic in the number of objects, making them too expensive for large datasets, and for high dimensional spaces.

a pixel. The value of p^i on coordinate j is equal to the gray value of the jth pixel in i. A projective cluster in this dataset consists of a set of faces that have similar gray values on a subset of the pixels. For example, a pixel in the eye region is dark for most of the faces. If the faces in the cluster have eyes at about the same position relative to the image boundary, then eye pixels correspond to some of the dimensions in the cluster. On the other hand, background pixels will in general have a lot of variation, and we do not expect them to belong to the cluster. We discuss this application in detail in section 5.

Our paper is organized as follows. In Section 2 we formally define the notion of an optimal projective cluster. We then propose a Monte Carlo algorithm for approximating an optimal projective cluster in Section 3, and prove our claims on the quality of the cluster it returns. We also discuss practical implementation issues and heuristics for speeding up our method. Our experiments on synthetic data are presented in Section 4, and our results on the image database are described in Section 5. We conclude in Section 6.

2. DEFINITION OF AN OPTIMAL PROJECTIVE CLUSTER

Let $p = (p_1, \ldots, p_d)$ be a point in $\mathbb{R}^d$. We use $[d]$ to denote the set of the d coordinate axes. As in previous approaches, we view a projective cluster as a pair $(\mathcal{C}, \mathcal{D})$, where $\mathcal{C}$ is a subset of the data, and $\mathcal{D}$ is a subset of the coordinate axes. We require the set $\mathcal{C}$ to be dense in the subspace spanned by $\mathcal{D}$ in the following sense: $|\mathcal{C}|$ must be sufficiently large, and the projection of $\mathcal{C}$ on the subspace spanned by $\mathcal{D}$ must be contained in a hyper-cube of given side length w. However, we do not make any assumption on the distribution of $\mathcal{C}$ inside the subspace spanned by $\mathcal{D}$. We give the formal definition below.

DEFINITION 1. *Let S be a set of points in $\mathbb{R}^d$. For any $0 \leq \alpha \leq 1$ and $w \geq 0$, an α-*dense projective cluster *of width w in S is a pair $(\mathcal{C}, \mathcal{D})$, $\mathcal{C} \subseteq S$, $\mathcal{D} \subseteq [d]$, such that*

*(1) $\mathcal{C}$ is α-*dense, *i.e. $|\mathcal{C}| \geq \alpha |S|$;*

(2) $\forall i \in \mathcal{D}$, $\max_{p \in \mathcal{C}} p_i - \min_{q \in \mathcal{C}} q_i \leq w$;

(3) $\forall i \in [d] \setminus \mathcal{D}$, $\max_{p \in \mathcal{C}} p_i - \min_{q \in \mathcal{C}} q_i > w$.

We define the *dimensionality* of a projective cluster to be $|\mathcal{D}|$. We say that $\mathcal{D}$ is the set of bounded dimensions, and $[d] \setminus \mathcal{D}$ is the set of unbounded dimensions. For example, $\mathcal{D} = \{1, 2\}$ for $\mathcal{C}_1$ in Figure 1(a). The third condition ensures that $\mathcal{D}$ is the maximal set of bounded dimensions, i.e., $\mathcal{D}$ contains all the dimensions i for which $\mathcal{C}$ projects onto an interval of length at most w. A projective cluster $(\mathcal{C}, \mathcal{D})$ of width w has a natural geometric interpretation as an axis-aligned box $\mathcal{B}_{\mathcal{C},\mathcal{D}} = [l_1, h_1] \times [l_2, h_2] \times \cdots \times [l_d, h_d]$, where $l_i = -\infty$ and $h_i = \infty$ if $i \notin \mathcal{D}$, and $l_i = \min_{p \in \mathcal{C}} p_i$ and $h_i = \max_{p \in \mathcal{C}} p_i$ if $i \in \mathcal{D}$; see Figure 1(b). By definition, a point $p \in S$ is in $\mathcal{C}$ if and only if p is contained in $\mathcal{B}_{\mathcal{C},\mathcal{D}}$. In describing our algorithm, we will also use a slightly different geometric object: For any $p \in S$ and $\mathcal{D} \subseteq [d]$, let $\mathcal{B}_{p,\mathcal{D}} = [l_1, h_1] \times [l_2, h_2] \times \cdots \times [l_d, h_d]$, where $l_i = -\infty$ and $h_i = \infty$ if $i \notin \mathcal{D}$, and $l_i = p_i - w$ and $h_i = p_i + w$ if $i \in \mathcal{D}$. By definition, for any cluster $(\mathcal{C}, \mathcal{D})$ and for any $p \in \mathcal{C}$, $\mathcal{B}_{p,\mathcal{D}} \supseteq \mathcal{B}_{\mathcal{C},\mathcal{D}}$. We say that $\mathcal{B}_{\mathcal{C},\mathcal{D}}$ has width w and $\mathcal{B}_{p,\mathcal{D}}$ has width $2w$.

Throughout this paper, we assume w to be fixed. Unless otherwise specified, all the projective clusters we consider have width at most w. For any $0 \leq \alpha \leq 1$, let $\mathcal{P}_\alpha$ denote the set of all α-dense projective clusters of width at most w from S. We want to be able to compare clusters from the set $\mathcal{P}_\alpha$ in order to determine the optimal one. Clearly, we must define a quality measure on $\mathcal{P}_\alpha$, and we must do so by taking into account properties that characterize the "natural" clusters occurring in real data. Intuitively, we would like each cluster to have as many points as possible, in order to be statistically relevant. We would also like each cluster to have as large a dimensionality as possible, since more bounded dimensions encode more amount of correlation in the data. However, it is easy to see that the two objectives are at odds. Consider the two extreme cases. If $\mathcal{C} = S$, then $|\mathcal{D}|$ is likely to be 0 or a very small integer; such a cluster does not help us mine any useful information from the data. On the other hand, if $|\mathcal{D}| = d$ then S probably consists of just a few points, since data tends to be very sparse in the full-dimensional space; again, such a cluster is useless. The solution is to specify a trade-off between the number of points and the dimensionality of a cluster. Taking these issues into account, we propose the following definition for the quality of a projective cluster.

DEFINITION 2. *Let S be a set of n points in $\mathbb{R}^d$. Let $\mu : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ be a function such that $\mu(0,0) = 0$ and μ is monotonically increasing in each argument. We define the* quality *of a projective cluster $(\mathcal{C}, \mathcal{D})$ to be $\mu(|\mathcal{C}|, |\mathcal{D}|)$. For any fixed $0 \leq \alpha \leq 1$, a projective cluster $(\mathcal{C}, \mathcal{D}) \in \mathcal{P}_\alpha$ is μ-*optimal *(or optimal for brevity) if it maximizes μ over $\mathcal{P}_\alpha$.*

*For any $0 \leq \beta < 1$, we say that a measure μ is β-*balanced *if $\mu(a, b) = \mu(\beta a, b + 1)$ for all $a > 0, b \geq 0$.*

The projective cluster problem over $\mathcal{P}_\alpha$ is to compute an optimal cluster in $\mathcal{P}_\alpha$ under a given β-balanced measure (note that there may be more than one optimal projective cluster in the data). For example, one such measure is $\mu(a, b) = a(1/\beta)^b$. The monotonicity requirement for μ models the fact that we want projective clusters $(\mathcal{C}, \mathcal{D})$ so that both $\mathcal{C}$ and $\mathcal{D}$ are maximal. The β-balanced condition specifies the tradeoff between the number of points and the number of dimensions in a cluster. Intuitively, for any projective cluster $(\mathcal{C}, \mathcal{D})$, we are willing to throw away at most a $(1 - \beta)$ fraction of the points in $\mathcal{C}$ in order to add one more dimension to $\mathcal{D}$. This condition could be relaxed as follows: μ is (β_1, β_2)-balanced, $\beta_1 \leq \beta_2$, if $\mu(\beta_1 a, b + 1) \leq \mu(a, b) \leq \mu(\beta_2 a, b + 1)$, $\forall a > 0, b \geq 0$. For simplicity, we let $\beta = \beta_1 = \beta_2$.

3. APPROXIMATING AN OPTIMAL PROJECTIVE CLUSTER

In this section we describe our algorithm for approximating an optimal projective cluster. Intuitively, our approach is as follows. Let $(\mathcal{C}^*, \mathcal{D}^*) \in \mathcal{P}_\alpha$ be an optimal projective cluster, and let $\mu^* = \mu(|\mathcal{C}^*|, |\mathcal{D}^*|)$ denote its quality. We guess (via random sampling) a seed $p \in \mathcal{C}^*$ and then determine the set $\mathcal{D}^*$ (see below). Let $\mathcal{C} = S \cap \mathcal{B}_{p,\mathcal{D}^*}$. Then $(\mathcal{C}, \mathcal{D}^*)$ has at least as many points as $(\mathcal{C}^*, \mathcal{D}^*)$ and the same dimensionality, which implies that its quality is at least μ^*. The only problem, however, is that $(\mathcal{C}, \mathcal{D}^*)$ has width $2w$ instead of w, and is in this sense an infeasible solution (recall that we formulated our projective cluster problem for clusters of fixed width w). From a practical point of view, this means that $(\mathcal{C}, \mathcal{D}^*)$ may attract some nearby points that belong to other clusters or are outliers. However, a point is attracted only if it is close to the seed along *each* bounded dimension in $\mathcal{D}^*$. This is unlikely to happen for any but a very small number of data points that do not belong to the cluster. Hence, it is reasonable to accept $(\mathcal{C}, \mathcal{D}^*)$ as an approximation of the optimal cluster. We say that $(\mathcal{C}, \mathcal{D}^*)$ is a 2-approximate solution because it has width $2w$, instead of w.

The only non-trivial step in the above approach is determining the set of dimensions $\mathcal{D}^*$. Note that $\mathcal{D}^*$ depends on $\mathcal{C}^*$, which we do not know. However, we show below that $\mathcal{D}^*$ can be determined

```
DOC($S$, $\alpha$, $\beta$)
  $r = \log(2d)/\log(1/2\beta)$; $m = (2/\alpha)^r \ln 4$;

for $i = 1$ to $2/\alpha$
    Choose $p \in S$ uniformly at random.
    for $j = 1$ to $m$
        Choose $X \subseteq S$ of size $r$ uniformly at random;
        $\mathcal{D} = \{k \mid |q_k - p_k| \leq w, \forall q \in X\}$;
        $\mathcal{C} = S \cap \mathcal{B}_{p,\mathcal{D}}$;
        if ($|\mathcal{C}| < \alpha|S|$) then
            $(\mathcal{C}, \mathcal{D}) = (\emptyset, \emptyset)$;
    endfor
  endfor
return cluster $(\mathcal{C}_O, \mathcal{D}_O)$ that maximizes $\mu(|\mathcal{C}|, |\mathcal{D}|)$
        over all computed clusters $(\mathcal{C}, \mathcal{D})$.
```

Figure 2: Algorithm for approximating an optimal projective cluster.

if we know only a small subset $X \subseteq S$, called a *discriminating set*. The name comes from the fact that we can use this set to discriminate between the bounded and unbounded dimensions of a cluster. More exactly, given a projective cluster $(\mathcal{C}, \mathcal{D})$ and a point $p \in \mathcal{C}$, a discriminating set for $(\mathcal{C}, \mathcal{D})$ with respect to p satisfies the following two conditions:

(a) $\forall i \in \mathcal{D}$ and $q \in X$, $|q_i - p_i| \leq w$;

(b) $\forall i \notin \mathcal{D}$, $\exists q \in X$ such that $|q_i - p_i| > w$.

If we know $p \in \mathcal{C}^*$ and a discriminating set X with respect to p, we determine $\mathcal{D}^*$ as follows: If $|q_i - p_i| \leq w$ for all $q \in X$, then $i \in \mathcal{D}^*$, otherwise $i \in ([d] \setminus \mathcal{D}^*)$. We guess both p and X via random sampling. The optimality of $(\mathcal{C}^*, \mathcal{D}^*)$ will imply that it admits a small discriminating set of size $O(\log d)$. This is important for the efficiency of our method, which has polynomial dependency on d (recall that other density-based approaches are exponential in the dimensionality).

Figure 2 describes our Monte Carlo algorithm, called DOC(S, α, β). As is the case with many Monte Carlo methods, the algorithm itself is extremely simple. It consists of repeatedly choosing p and X via random sampling, computing the corresponding cluster $(\mathcal{C}, \mathcal{D})$ as outlined above, and then reporting the best found cluster. The challenge consists in proving that it works correctly. The values for r (size of random sample) and m (number of inner iterations) are chosen according to the theoretical analysis, which is provided in Section 3.1.

3.1 Correctness and running time

We prove the correctness of our algorithm using the following lemmas.

LEMMA 1. *Let α be such that*

$$\max_{(\mathcal{C},\mathcal{D}) \in \mathcal{P}_\alpha} \mu(|\mathcal{C}|, |\mathcal{D}|) = \max_{(\mathcal{C},\mathcal{D}) \in \mathcal{P}_{\alpha'}} \mu(|\mathcal{C}|, |\mathcal{D}|), \ \forall \alpha' \leq \alpha, \qquad (*)$$

Then, any optimal cluster $(\mathcal{C}^*, \mathcal{D}^*)$ of $\mathcal{P}_\alpha$ satisfies the following property: $\forall i \notin \mathcal{D}^*$, $\forall a \in \mathbb{R}$, $|\{p \in \mathcal{C}^* \mid p_i \in [a, a+w]\}| \leq \beta|\mathcal{C}^*|$. We say that $(\mathcal{C}^*, \mathcal{D}^*)$ is β-balanced.

The proof follows immediately from the fact that $(\mathcal{C}^*, \mathcal{D}^*)$ is optimal and the fact that the quality measure is β-balanced. It is easy to see that any $\alpha \leq \beta^d$ satisfies equation $(*)$, but we expect α to be much larger in practice.

LEMMA 2. *Let $1/(4d) \leq \beta \leq 1/2$. Let p^i be the point chosen during the ith outer iteration of* DOC(S, α, β)*, and let X^j be the set chosen during the corresponding jth inner iteration of* DOC(S, α, β)*. If $p^i \in \mathcal{C}^*$ then, with probability at least $3/4$, there exists j such that X^j is discriminating for $(\mathcal{C}^*, \mathcal{D}^*)$ with respect to p^i.*

PROOF. In the following, (a) and (b) refer to the two conditions in the definition of a discriminating set. Let j be a fixed inner iteration. Define X_k^j (resp., p_k^i) to be projection of X^j (resp., p^i) onto dimension k. If $X^j \subseteq \mathcal{C}^*$ then X^j satisfies (a). Hence

$$\Pr\{X^j \text{ satisfies (a) and (b)}\} \geq$$
$$\Pr\{X^j \subseteq \mathcal{C}^*\}(1 - \Pr\{X^j \text{ violates (b)} | X^j \subseteq \mathcal{C}^*\}).$$

Since $|\mathcal{C}^*| \geq \alpha|S|$, $\Pr\{X^j \subseteq \mathcal{C}^*\} \geq \alpha^r$. X^j violates (b) if and only if there exists $k \notin \mathcal{D}^*$ such that $X_k^j \subseteq [p_k^i - w, p_k^i + w]$. Recall that $(\mathcal{C}^*, \mathcal{D}^*)$ is β-balanced. This implies that for any $k \notin \mathcal{D}^*$, $|\mathcal{C}^* \cap [p_k^i - w, p_k^i + w]| \leq 2\beta|\mathcal{C}^*|$. We deduce

$$\Pr\{X^j \text{satisfies (a) and (b)}\} \geq \alpha^r(1 - d(2\beta|\mathcal{C}^*|/|S|)^r) \geq \alpha^r/2,$$

which implies

$$\Pr\{\forall j,\ X^j \text{ not discriminating for } p^i\} \leq (1 - \alpha^r/2)^m \leq 1/4$$

(we used the fact that $\beta \geq 1/(4d)$ to deduce $r \geq 1$, and so $\alpha^r/2 \geq \alpha^r/2^r$). Hence, with probability at least $3/4$, there exists j such that X^j is discriminating for $(\mathcal{C}^*, \mathcal{D}^*)$ with respect to p^i. □

LEMMA 3. *Let p^i be the point chosen during the ith outer iteration of the procedure* DOC(S,α, β)*. Then with probability at least $3/4$ there exists i such that $p^i \in \mathcal{C}^*$.*

PROOF. Since $|\mathcal{C}^*| \geq \alpha|S|$, we deduce

$$\Pr\{\forall i,\ p^i \notin \mathcal{C}^*\} \leq (1 - \alpha)^{2/\alpha} \leq e^{-2} < 1/4.$$

□

Lemmas 2 and 3 imply that the probability of success of our algorithm is at least $3/4 \cdot 3/4 > 1/2$. We thus conclude with the following.

THEOREM 1. *Let μ be a β-balanced quality measure, $1/(4d) \leq \beta < 1/2$, and let $0 < \alpha < 1$. Then, with probability at least $1/2$,* DOC(S, α, β) *returns a 2-approximate solution.*

By a standard technique, the probability of success can be boosted to $1 - 1/2^m$ by repeating DOC(S, α, β) m times and reporting the best overall cluster. In our experiments we noticed that one or two calls to DOC were sufficient to generate a good projective cluster.

REMARK 1. *The approximation guarantee can be improved if we have information on the distribution of cluster points inside the subspace. Let $\mathcal{C}^*_{\mathcal{D}^*}$ denote the projection of $\mathcal{C}^*$ onto the subspace spanned by $\mathcal{D}^*$. Then $\mathcal{C}^*_{\mathcal{D}^*}$ lies in a $|\mathcal{D}^*|$-dimensional hypercube H of size w. If, for example, we know that $\mathcal{C}^*_{\mathcal{D}^*}$ has a normal distribution with mean at the center of H and with known standard deviation, then we can easily modify DOC to compute a $(1 + \varepsilon)$-approximate solution, for any $\varepsilon > 0$. However, since we cannot always rely on having such a priori information on the data, we presented the algorithm in its most general form, which is independent of data distribution.*

Running time. During each iteration we compute $\mathcal{D}$ in $O(d)$ time and $\mathcal{C}$ in $O(nd)$ time. The total number of iterations is $m = (2d)^C \ln 4$, where C is a constant that depends only on α and β (more precisely, $C = \log(2/\alpha)/\log(1/(2\beta))$). Hence, the overall running time is $O(nd^{C+1})$.

```
FastDOC($S, \alpha, \beta$)
    $r = \log(2d)/\log(1/2\beta)$;
    $m = \min\{MAXITER, (2/\alpha)^r \ln 4\}$;

1. $\mathcal{D}_M = \emptyset$;
2. for $i = 1$ to $2/\alpha$
3.      Choose $p \in S$ uniformly at random.
4.      for $j = 1$ to $m$
5.           Choose $X \subseteq S$ of size $r$ uniformly at random;
6.           $\mathcal{D} = \{k \mid |q_k - p_k| \le w, \ \forall q \in X\}$;
7.           if ($|\mathcal{D}| \ge |\mathcal{D}_M|$) then
8.              $\mathcal{D}_M = \mathcal{D}$;
9.           if ($|\mathcal{D}_M| \ge d_0$) then
10.             go to $11$;
        endfor
   endfor
11. $\mathcal{C} = S \cap \mathcal{B}_{p,\mathcal{D}_M}$;
12. return cluster $(\mathcal{C}, \mathcal{D}_M)$.
```

Figure 3: Heuristic method for the projective cluster problem.

3.2 Speeding up the algorithm

From a theoretical point of view the DOC algorithm is very efficient, since its running time is linear in the size of data and polynomial in the dimensionality. However, it is important to note that the algorithm scans the entire data during each inner iteration to compute the set $\mathcal{C}$. This process can be very time consuming, since the points are high dimensional and the data files are usually large (in our experiments, they are in the range of 10–100M). Therefore, we propose a few simple heuristics to reduce the number of data scans and speed up the algorithm. These heuristics come with a price: we lose some of the quality guarantees proven in the previous subsection. However, as we discuss below, it is likely that the computed clusters are relevant in most practical applications.

We speed up our algorithm as follows. During each inner iteration, we only compute the set $\mathcal{D}$. After the m inner iterations are executed, let $\mathcal{D}_M$ be the largest set among the m sets of dimensions computed. We compute $\mathcal{C} = S \cap \mathcal{B}_{p,\mathcal{D}_M}$. Thus, we now read the data only once per outer iteration. We can no longer guarantee (in a probabilistic sense) that we return a cluster of quality at least μ^*. However, we can prove using arguments similar to the proof of Theorem 1 that we return a cluster of size at most $2w$ which is α-dense and has large dimensionality (at least as large as the highest dimensionality of an α-dense β-balanced cluster). For most applications requiring pattern discovery or data indexing, a projective cluster with these properties is good enough. We can further reduce the amount of computation as follows. Given a user-specified threshold d_0, once we discover a set $\mathcal{D}$ with $|\mathcal{D}| \ge d_0$, we compute the corresponding set $\mathcal{C}$ and return $(\mathcal{C}, \mathcal{D})$. Moreover, in order to reduce CPU time, we upperbound the number of inner iterations m by a value MAXITER. The resulting heuristic method, called FastDOC, is described in Figure 3. This approach no longer guarantees an α-dense cluster, but it is likely that it returns one with sufficiently many points.

Note that we now compute $\mathcal{C}$ outside the iterations. Hence, we scan the data only once for each computed cluster. In addition, we also have to access the data in order to choose the random samples. However, we can choose all random samples in one scan and store them in memory, since the size of each sample is small, and the number of samples m is at most MAXITER. In our experiments, we choose MAXITER to be $d^2 < 10^6$ when d is of the order of hundreds of dimensions. Hence, this approach requires two data scans per computed cluster. However, we can reduce the I/O complexity even further, by pipelining the computation across clusters. Recall that we call FastDOC repeatedly, in order to compute the projective clusters one by one. During the computation of the ith cluster, the data is scanned in step 11 of the algorithm in order to compute the set $\mathcal{C}$. While doing so, we also choose and store the random samples that will be used in the computation of the $(i+1)$st cluster. Thus, during the computation of the $(i+1)$st cluster, steps 3 and 5 no longer require accessing the data on disk. The only exception is the computation of the first cluster, which does need to select its random samples directly from the disk. We conclude that the overall number of data scans we need is exactly one more than the number of computed clusters.

4. EXPERIMENTS ON SYNTHETIC DATA

We perform our simulations on a Pentium III 800MHz machine with 256MB of RAM running the Linux operating system. In order to evaluate the competitiveness of our method, we implemented both PROCLUS [1] and ORCLUS [2], the two previous algorithms that compute axis-parallel projective clusters. In fact, ORCLUS is a more general algorithm that is able to compute arbitrarily oriented clusters. However, it can be restricted to compute only axis-parallel clusters, and we use it in this sense for comparison reasons. Our intention was to evaluate all three methods on the same datasets. However, PROCLUS proved to be prohibitively expensive for datasets of 100,000 points in 200 dimensions. The reason is that PROCLUS is a hill-climbing technique that executes a large number of iterations (usually 100 or more) in order to successively improve the current clustering. We attempted to set the number of iterations to a lower value (first 30, then 60), but it proved to be insufficient and the resulting clusterings were quite inaccurate. We did obtain accurate results with PROCLUS on smaller sets and lower dimensional spaces. However, since it did not scale well, we decided to drop it from further experiments.

Data generation. We use the data generator described in the PROCLUS paper. The ORCLUS generator is extremely similar, except that it generates arbitrarily oriented clusters. However, since we focus on axis-parallel clusters, the two generators are the same.

All the generated points have coordinates in the range $[0 \ldots 100]$. Unless otherwise specified, each dataset has 100,000 points that are 200-dimensional, and consists of 5 clusters. We slightly modify the generator in the following sense: Recall that when generating cluster $i+1$, about 50% of its bounded dimensions are chosen from among the bounded dimensions of cluster i. This is intended to model the fact that different clusters often share some dimensions. However, the centers of cluster i and $i+1$ are chosen completely independent of each other. In other words, even if clusters i and $i+1$ share some dimensions, they are likely to be well separated in that respective subspace. We modify the algorithm for choosing the cluster centers in the following sense. Suppose that clusters i and $i+1$ both have bounded dimension j. Let c_j^i and c_j^{i+1} be the j-coordinates of centers c^i and c^{i+1}. We choose $c_j^{i+1} \le c_j^i + 2\sigma_j^i$, where σ_j^i is the standard deviation of the points in cluster i along dimension j. Hence, some of the points in clusters i and $i+1$ will be close to each other in the subspace spanned by the common bounded dimensions of the clusters. This introduces an additional challenge for distinguishing between the clusters.

Furthermore, we want to investigate the performance of DOC and ORCLUS under different distributions of the cluster points in the subspace of bounded dimensions. The original data generator only allowed for clusters whose points have a normal distribution along the bounded dimensions. We modify it so that we can generate the following types of projective clusters:

1. U-clusters: Cluster points are uniformly distributed in a small hyper-cube in the subspace of bounded dimensions. Usually, the hyper-cube size is 15 or 20.

2. MGq-clusters: Cluster points follow a mixture of q Gaussians distribution in the subspace of bounded dimensions. For a fixed cluster, let c be the center of the cluster, selected as before. For any bounded dimension j let σ_j be the standard deviation generated as in the original algorithm. We define the mean values of the Gaussians along dimension j to be equally spaced in the interval $[c_j - \sigma_j, c_j + \sigma_j]$. The standard deviation of each Gaussian along dimension j is σ_j / q.

3. N-clusters: These are generated as in the original algorithm, so that cluster points follow a normal distribution in the bounded subspace. Any N-cluster can be viewed as an MG-cluster with $q = 1$.

We say that a dataset is a U-set, MGq-set or N-set depending on the type of clusters it contains.

Finally, we wish to verify our intuition that DOC should not miss small projective clusters. In order to do so, we modify the procedure for generating the number of points in each cluster. Recall that the original generator assigns n_i points to cluster i, where n_i is proportional to the realization of an exponential random variable. This results in all clusters having reasonably close sizes. To force imbalance in the cluster sizes, we start by computing the values n_i in a similar manner, and then do the following. Assume that k is the number of generated clusters. For each $i \leq k/2$, let $n'_i = \delta n_i$, and $n'_{i+k/2} = n_{i+k/2} + (1-\delta) n_i$, where $0 < \delta < 1$ is a parameter (in our simulations, we used $\delta \in \{0.2, 0.33, 0.5\}$). We let the values n'_i be the number of points that we generate in each cluster.

4.1 Comparison with ORCLUS

Because ORCLUS requires all projective clusters to exist in the same number of dimensions, we generate projective clusters under this additional restriction. Unless otherwise specified, the cluster dimensionality is 40. Note that this is a fairly strong restriction, since real data is unlikely to exhibit subspace patterns of the same dimensionality. However, DOC is also able to discover clusters of different dimensionality. In fact, our method is completely oblivious to the dimensionality in which the clusters have been generated, and it decides for itself the appropriate dimensionality of each cluster. In order to test its accuracy in this respect, the results we report for DOC are averaged over two data files: one in which all clusters have dimensionality 40, and one in which each cluster has different dimensionality, and the *average* cluster dimensionality is 40. ORCLUS is only tested on the first set. Moreover, all reported results are averaged over three runs on the same file(s), in order to offset the effects of randomization. We note that DOC returned qualitatively similar results on both types of files, i.e. the averaging over different files does not make any of the two cases look better.

The accuracy results that we report throughout this section are computed as follows. For each output cluster i, we identify the input cluster j with which it shares the largest number of points. We say that output cluster i *corresponds* to input cluster j, and that all points in their common intersection are labeled correctly. All the other points of output cluster i are labeled incorrectly. The output and input outlier sets are treated similarly. We then report the percentage of points that are correctly labeled by each algorithm. We noticed a rather surprising instability of ORCLUS over the MG-sets and N-set. In each case, one or two runs returned an accurate clustering, while the remaining runs computed clusterings with significant error (mostly, a large percentage of cluster points were labeled outliers). By contrast, ORCLUS was very stable on the U-set, on which it consistently returned a good clustering. DOC did not exhibit any instability over the different runs or data distributions. This confirmed our intuition that, since we do not use any information on the data distribution, the algorithm should be stable under any distribution. ORCLUS makes implicit assumptions on the data distribution, since it uses measures of data correlation when deciding which dimensions to choose for each cluster. However, this does not seem to fully explain its instability. This issue may be worthy of further examination, since it has the potential for improving the performance of ORCLUS.

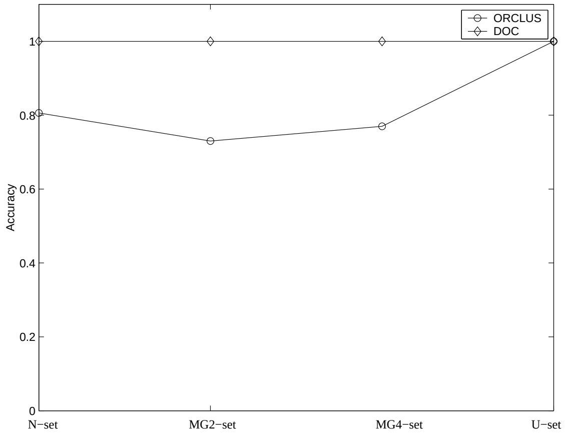


Figure 4: Dependence of accuracy on data distribution.

We also report our results on the accuracy of the two methods when the cluster sizes vary significantly. The x-axis in Figure 5 is labeled by the ratio between the sizes of the smallest and largest clusters. The smaller the ratio, the more imbalance there is in the cluster sizes. All data sets we used were U-sets, since ORCLUS proved stable on them. As expected, DOC does not miss the small clusters and has very high accuracy (the accuracy is not 1, since 5 points are mislabeled in some cases, but they are statistically insignificant). The accuracy of ORCLUS is also reasonably good. However, a closer look shows that in fact ORCLUS entirely misses one or two small clusters in each case. The fact that few points are mislabeled is due to the fact that those clusters do not have many points to begin with.

Parameter Choices. In all the experiments reported so far, we set $\alpha = 0.1$, $\beta = 0.25$, and $w = 15$. We used the following considerations and guidelines for choosing these values.

One heuristic for choosing w is as follows: For a data point p^i, let q^i denote its nearest neighbor. Let $w^i = \sum_{j=1}^{d} |p^i_j - q^i_j|/d$ be the average distance between p^i and q^i along one dimension. Then one can choose $w = C \cdot \sum_{i=1}^{n} w^i/n$ for some small constant C. We also use this strategy with good results on our real dataset.

Intuitively, α can be viewed as a minimum required cluster density. The smaller we choose α, the more likely it is that we discover a particular input cluster. However, if α is too small we may execute too many outer iterations. On the other hand, if α is too large, we can miss an input cluster entirely. Figure 6 illustrates how α influences the accuracy of the result (we fix $\beta = 0.25$). In these experiments output clusters are not required to be disjoint, i.e., we do not eliminate the points that belong to a computed cluster. Hence, when α is too large, we miss smaller clusters. We also ran the ex-

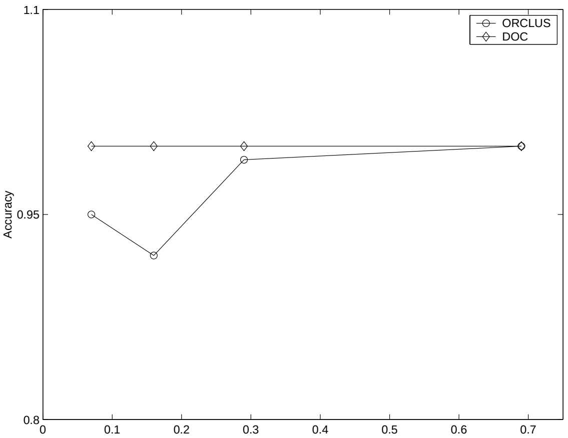


Figure 5: Dependence of accuracy on cluster sizes.

periments with the requirement that output clusters be disjoint. In that case α did not influence the accuracy. This can be explained by the fact that, when we eliminate the points of a computed cluster, the density of the remaining clusters with respect to the remaining set of points increases.

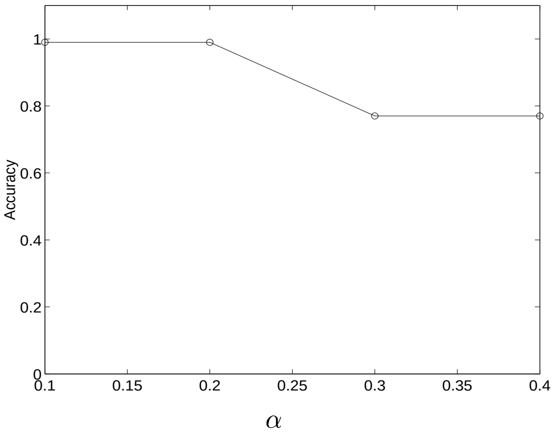


Figure 6: Dependence of accuracy on α.

Parameter β represents the user's "opinion" on the relative importance of points versus number of dimensions in a cluster. Figure 7 illustrates how β influences the accuracy. As expected, when β is small, the algorithm chooses more dimensions for a cluster at the cost of throwing away cluster points. At the other end, when β is large (i.e., $\beta = 0.35$), the algorithm merges two input clusters. The resulting output cluster has fewer dimensions than any of the original clusters, but it has twice as many points. Hence, in this case the algorithm favors points over dimensions. However, one could argue that the output is still interesting, as the resulting cluster has 22 dimensions and it contains about 40% of the entire dataset.

The accuracy of the method changes if the cluster width is larger. We report experiments on a dataset for which the maximum cluster width is 40. We run the algorithm with $w = 25$, $\alpha = 0.1$, and $\beta = 0.4$. Since the width is large, we need a larger β to avoid choosing too many extra dimensions in the cluster (recall, though, that Theorem 1 requires $\beta < 1/2$). Even for $\beta = 0.4$, the algorithm generates a corresponding output cluster with more dimensions and fewer points. Therefore, to avoid having many cluster points thrown away as outliers, we generate 15 output clusters (three times the number of input clusters). Only 0.1% of cluster points are mis-labeled outliers by the algorithm.

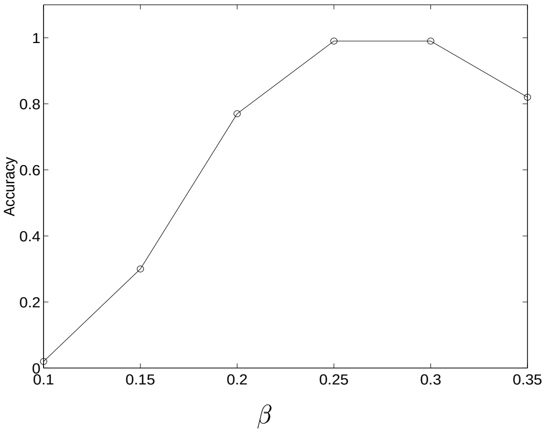


Figure 7: Dependence of accuracy on β.

In Figure 8 we show a snapshot of the Confusion Matrix, defined as follows: entry (i, j) is equal to the number of data points assigned to output cluster i, that were generated as part of input cluster j. The last row and column are output, resp. input, outliers. Each output cluster contains about $5 - 10\%$ of the data, and most clusters have at least 85% of dimensions in common with the corresponding input clusters. The lowest such percentage is 50%. Thus, although the algorithm does not discover the original clusters, it nevertheless detects clusters that are interesting and likely to be useful.

In	A	B	C	D	E	$\mathcal{O}$
Out						
1	5842	0	0	0	0	0
4	230	13944	0	0	0	0
5	0	0	347	13177	1242	0
6	1835	0	0	0	0	0
15	2084	11247	0	0	0	0
$\mathcal{O}$	7	14	65	30	10	5000

Figure 8: Confusion Matrix for large cluster width (snapshot).

Influence of dimensionality and data size. Finally, we study how the accuracy of our method is influenced by the cluster dimensionality and the size of the dataset.

Figure 9 shows how the accuracy varies with the average number of dimensions in a cluster. Each point on the graph corresponds to a different file containing five clusters $(\mathcal{C}_i, \mathcal{D}_i)$ so that $\sum_i |\mathcal{D}_i|/5$ is approximately the x-value on the graph. For each file we report the most accurate result out of three executions. However, the variation in the accuracy across different executions on the same set was generally insignificant. The experiments show that the algorithm works very well unless the average value of $|\mathcal{D}|$ is very small. Figure 10 shows that the accuracy does not change when the size of the

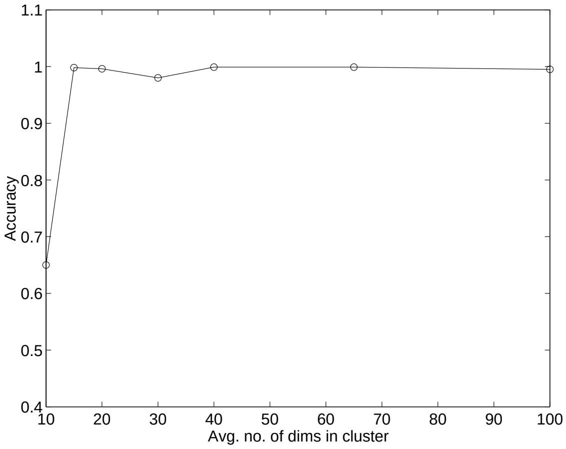


Figure 9: **Dependence of accuracy on average number of cluster dimension.**

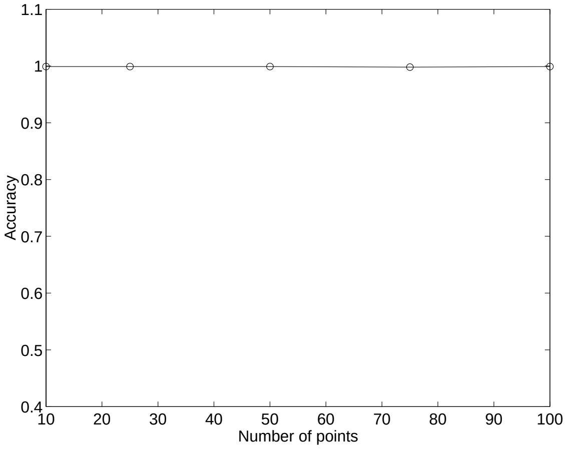


Figure 10: **Dependence of accuracy on number of points.**

training data decreases. The method takes between 2–30 seconds to generate a projective cluster, depending on the input size and the outcome of coin flips. The variation is due to randomization. For the right choice of parameters α and β (see above), the method usually computes a projective cluster in less than 5 seconds.

5. FACIAL ROTATION ESTIMATOR

In this section we describe an application of projective clustering in computer vision. Our database consists of gray scale images of size 16×16 pixels representing human faces. Each face is frontal but may be rotated in-plane. We repeatedly apply the heuristic described in the previous section, in order to discover projective clusters in the data. We allow projective clusters to overlap, since we determined experimentally that this approach is better than computing disjoint clusters. A possible explanation is that an image is likely to fit well in more than one pattern, and that each pattern contains a relatively small percentage of images. For example, an image representing a high-cheekboned man with a moustache may be selected in a cluster of high-cheekboned faces, and in a cluster of faces with a moustache. Although these clusters overlap, we expect the overlap to be small, and so they should not be merged into just one cluster. On the other hand, a cluster may have small overlaps with a large number of other clusters. If we eliminated the points that are already clustered, the data would quickly become too fragmented and the computation of additional projective clusters would be meaningless.

We propose an algorithm that uses the projective clusters to classify query images into one of 16 rotation classes. This task is very useful in enabling face detectors to detect rotated faces. Successful solutions to face detection (see, e.g., [13, 14] and references therein) typically work by training a classifier that takes a fixed size image (e.g., 16×16 pixels) as input and outputs 1 if the image is a face and 0 otherwise. To find all faces in a large image, a 16×16 window is scanned over all possible positions in the image as well as at all possible scales. The detector is evaluated on each window. Successful detectors handle frontal upright faces. These detectors can be augmented to handle in-plane rotations as follows: a rotation estimator determines the angle by which an image window should be derotated to make it upright and the derotated image is used as input to the frontal upright face detector. A previous approach estimated rotation using a neural network [12]. Our rotation estimator throws away most non-face windows. However, it also accepts a certain percentage of non-face windows, which it is likely to derotate by a random amount; the face detector should still classify the derotated windows as non-faces.

To estimate the in-plane rotation angle for a face image, we first discretize the rotation angle into 16 distinct classes, each spanning $22.5°$. For each rotation class we use a training set of about $5,000$ faces (see Figure 11). All the original face images belong to rotation class 0, which covers roughly $-11.25°$ to $11.25°$. We synthetically generated training examples for the other 15 rotation classes by rotating the upright images appropriately. For each rotation class, we computed 500 projective clusters, which are enough to cover most of the training faces in that class. We used the value $w = 90$ in our experiments (the maximum value for w is 256, i.e. the number of gray values). The average dimensionality of the computed clusters was 45 (almost 20% of the total number of dimensions).

Since the dimensions of the data space are image pixels, there is an underlying correlation among some of the dimensions. For example, adjacent pixels are likely to have similar values. However, our projective cluster algorithm assumes that dimensions are independent. This assumption can make the resulting projective

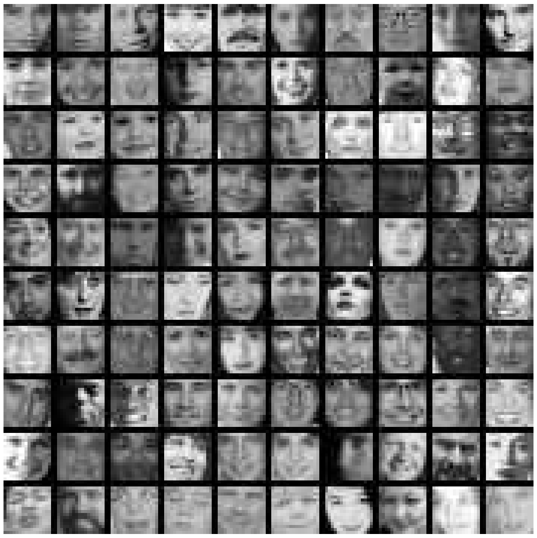


Figure 11: Example of frontal upright face images used for training.

clusters very sensitive to changes like small translation or scaling of the images. Such sensitivity is undesirable, since small transformations do not affect the overall appearance of an image, and both the original and the transformed image should be covered by the same cluster. In order to reduce this sensitivity, we apply a transformation such that the pixel values in the resulting image reflect a certain correlation among neighboring pixels in the original image. Thus, even though the projective cluster method still treats the pixels independently, the data points themselves reflect pixel correlations. In addition, our transformation also reduces the effects of different illuminations of the images. The transformation first converts each 8-bit gray scale pixel to one of the values 0, 128, or 255 (i.e. black, gray, or white), as follows. A pixel is set to 255 if its value is significantly larger than the average value of its neighbors. Similarly, a pixel is set to 0 if its value is significantly smaller than the average value of its neighbors. Otherwise, it is set to 128. The resulting image is then smoothed using standard techniques. Figure 12 shows an example of a face image and its transformation.

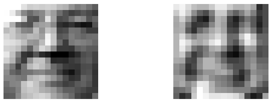


Figure 12: Original image (left) and resulting transformed image (right).

To estimate the rotation class for a query image, we determine the projective cluster that contains it (if any), and label the query accordingly. If more than one projective cluster contains the query, we label it with the rotation class that has the most clusters containing the query. We tested the accuracy of rotation estimation as follows. We rotated each face in a test set of 60 frontal upright faces in 1° increments from 0° to 360°. We labeled each of these 21600 images with its rotation class and used the projective clusters to estimate the rotation class as above. We computed the accuracy of the rotation estimator as follows: if a face is in rotation class i and its rotation class is estimated as j, the error is $j - i$, except for $j = 15$ and $i = 0$, when the error is -1. Figure 13 shows the resulting histogram of errors, with the x-axis corresponding to the error. The figure shows that the rotation class for the majority of the examples is correctly classified. About 85% of the examples have an error of 0 or ±1. Thus, projective clusters can determine the rotation class of a query with relatively high accuracy. It is interesting to note that, aside from 0 and ±1, the only other error values corresponding to a significant percentage of points are ±8. This happens because, due to the low resolution, the mouth and eye regions look similar, and the algorithm may "switch" them, discovering the image upside-down.

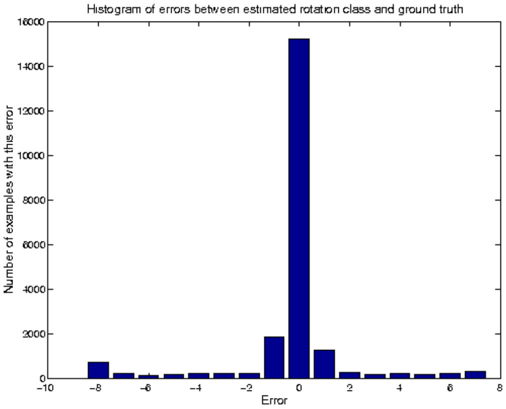


Figure 13: Histogram of errors for rotation estimation.

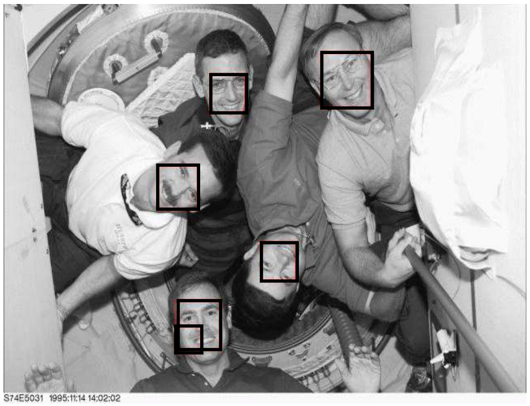


Figure 14: Output of the face detector on the "Mir" image.

Finally, we integrated the rotation estimator with a frontal face detection method as described earlier. The resulting system used our rotation estimator as the front end and the Viola-Jones frontal face detector [14] as the back end. Figure 14 shows the result of the algorithm on a photograph of cosmonauts inside the Mir space station. The black rectangles indicate the windows in which the algorithm discovers human faces (for clarity purposes, if two windows overlap significantly, only one of them is drawn).

6. CONCLUSIONS

We have proposed a novel approach for computing projective clusters in large, high-dimensional data sets. Departing from the traditional partition-based approach, we employ a greedy method that computes each cluster in turn. While this approach has many advantages, its accuracy depends on finding a good definition for an optimal projective cluster. We propose one such definition starting from practical considerations, and show through extensive experiments that it is, indeed, a good model for our problem. Based on this model, we also propose a fast, provably accurate randomized algorithm for computing projective clusters. Our implementation proves highly accurate and stable for various types of data, on which partitioning algorithms are not always successful. Moreover, we use our ideas in an image processing application, for which we are able to achieve good accuracy results.

One obvious direction for furthering this work is to extend our approach to the case of arbitrarily oriented clusters. This will require new insights into the properties of such clusters, in order to model and compute optimal subspace patterns.

7. REFERENCES

[1] C. C. Aggarwal, C. M. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park. Fast algorithms for projected clustering. In *Proc. of ACM SIGMOD Intl. Conf. Management of Data*, pages 61–72, 1999.

[2] C. C. Aggarwal, and P. S. Yu. Finding generalized projected clusters in high dimensional spaces. In *Proc. of ACM SIGMOD Intl. Conf. Management of Data*, pages 70–81, 2000.

[3] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. ACM SIGMOD Conf. on Management of Data*, pages 94–105, 1998.

[4] K. Chakrabarti and S. Mehrotra. Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *Proc. 26th Intl. Conf. Very Large Data Bases*, pages 89–100, 2000.

[5] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. 2nd Intl. Conf. Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[6] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. Density-connected setsand their application for trend detection in spatial databases. In *Proc. 3rd Intl. Conf. Knowledge Discovery and Data Mining*, 1997.

[7] S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In *Proc. ACM SIGMOD Intl. Conf. Management of Data*, pages 73–84, 1998.

[8] A. Hinneburg, C. C. Aggarwal, and D. A. Keim. What is the nearest neighbor in high dimensional spaces? In *Proc. 26th Intl. Conf. Very Large Data Bases*, pages 506–515, 2000.

[9] A. Hinneburg and D. A. Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In *Proc. 25th Intl. Conf. Very Large Data Bases*, pages 506–517, 1999.

[10] A. Hinneburg and D. A. Keim. An efficient approach to clustering in large multimedia databases with noise In *Proc. 4th Intl. Conf. Knowledge Discovery and Data Mining*, 1998.

[11] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *Proc. 20th Intl. Conf. Very Large Data Bases*, pages 144–155, 1994.

[12] H. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 20:22–38, 1998.

[13] H. Schneiderman and T. Kanade. A statistical method for 3d object detection applied to faces and cars. In *Proc. IEEE Intl. Conf. Comput. Vision*, 2000.

[14] P. Viola and M. Jones. Robust real-time object detection. Technical Report 2001/01, Compaq Cambridge Research Lab, 2001.

[15] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. In *Proc. ACM-SIGMOD Intl. Conf. Management of Data*, pages 103–114, 1996.