SIMPLE AND EFFICIENT PROJECTIVE CLUSTERING

Clark F. Olson and Henry J. Lyons
Computing and Software Systems, University of Washington
Box 358534, 18115 Campus Way N.E., Bothell, WA 98011-8246, U.S.A.
cfolson@u.washington.edu, henry.j.lyons@gmail.com

Keywords: Projective clustering, Monte Carlo algorithm.

Abstract: We describe a new Monte Carlo algorithm for projective clustering that is both simple and efficient. Like previous Monte Carlo algorithms, we perform trials that sample a small subset of the data points to determine the dimensions in which the points are sufficiently close to form a cluster and then search the rest of the data for data points that are part of the cluster. However, our algorithm differs from previous algorithms in the method by which the dimensions of the cluster are determined and the method for determining the points in the cluster. This allows us to use smaller subsets of the data to determine the cluster dimensions and achieve improved efficiency over previous algorithms. The complexity of our algorithm is $O(nd^{1+\log\alpha/\log\beta})$, where n is the number of data points, d is the number of dimensions in the space, and α and β are parameters that specify which clusters should be found. To our knowledge, this is the lowest published complexity for an algorithm that is able to place high bounds on the probability of success. We present experiments that show that our algorithm outperforms previous algorithms on real and synthetic data.

1 INTRODUCTION

Data clustering is a technique for the unsupervised classification of data points into (possibly overlapping) sets that has many applications (Jain et al., 1999). However, clustering in many dimensions often yields poor results, since clusters may form in only a subset of the dimensions (Agrawal et al., 1998). In fact, it has been shown that, under some assumptions, the ratio between the distance to the nearest neighbor and the distance to the farthest neighbor approaches one as the dimensionality of the space increases (Beyer et al., 1999). In such a situation, clustering in the full space becomes nearly meaningless.

Projective clustering is a special case of the data clustering problem in which the clusters of data points are allowed to form in a subset of the dimensions of the full space. As opposed to dimensionality reduction techniques (Dash et al., 2002; Ding et al., 2002), the clusters are not constrained to form in the same subset of dimensions. While most projective clustering problems consider a many-dimensional space, we can illustrate the problem using three dimensions. Two projective clusters are present in Fig. 1(a), one in the x and y dimensions and one in the y and z dimensions. Figure 1(b) shows the projection of the points onto the x - y plane, where the first cluster is easy to see.

More formally, the projective clustering problem can be stated as follows. Given a set of points P in a d-dimensional space, find all maximal sets of points $C_i \subset P$ and corresponding sets of dimensions $D_i \subset [1,...,d]$ such that the point set is sufficiently close in each of the dimensions to form a cluster. We say that the points *congregate* in this dimension; *cluster* will be used only as a noun. We also say that the dimensions in D_i are the *congregating dimensions* of the cluster. For our purposes, we define this to be true if they are within a width w of each other:

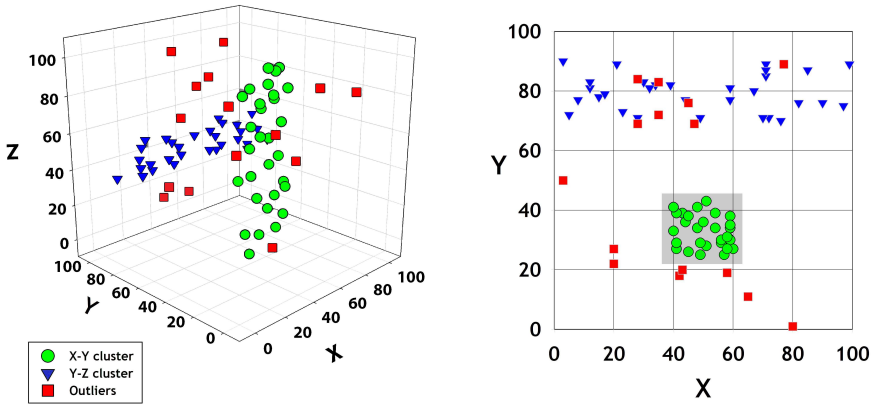
$$\forall p,q \in C_i, \forall j \in D_i, |p_j - q_j| \leq w. \quad (1)$$

Furthermore, to be reported, each cluster must surpass some criteria with respect to the cardinalities of C_i and D_i:

$$\mu(|C_i|,|D_i|) \geq \mu_0. \quad (2)$$

The clusters reported are maximal sets to prevent all subsets of each cluster from being reported. Most algorithms relax the requirement of reporting all maximal sets, since there are typically many overlapping maximal sets that meet the above requirements. This is necessary for an efficient algorithm, since it is possible for the number of such maximal sets to be quite large.

Our contribution is a new algorithm for projective clustering that is simple, robust and efficient. We call



(a) (b)

Figure 1: Projective cluster example. (a) Two projective clusters among outliers in a three-dimensional space. (b) The projection of the points onto the x-y plane illustrates one projective cluster, but not the other (which lies in the y-z plane).

it SEPC, for Simple and Efficient Projective Clustering. Using a Monte Carlo algorithm, we find projective clusters with high probability. The complexity of the algorithm has a linear dependence on the number of data points and a polynomial dependence on the number of dimensions in the space. The algorithm does not require the number of output clusters as an input, it is able to operate with clusters of arbitrary size and dimensionality, and it is robust to outliers. No assumptions are made about the distribution of the clusters or outliers, except that the clusters must have a diameter no larger than a user-defined constant in any of the cluster dimensions. In addition, the algorithm can be used either to partition the data into disjoint clusters (with or without an outlier set) or generate overlapping dense regions in the projective subspaces. Our algorithm generates tighter clusters than previous Monte Carlo algorithms, such as FastDOC (Procopiuc et al., 2002). The computational complexity of our algorithm is also less dependent on the cluster evaluation parameters and is lower overall. Our experiments show that we are able to find projective clusters that are missed by FastDOC. The performance of the algorithm on a real data set is superior to previously reported results.

In Section 2, we review prior work on this problem. We then discuss our approach in Section 3. The algorithm itself is described in Section 4, as well as an analysis of the efficiency of the algorithm and optimizations. Our experimental results on random and real data are presented in Section 5. Finally, our conclusions are given in Section 6.

2 PREVIOUS WORK

Considerable research has been performed on projective clustering. Parsons, Haque and Liu (2004) review many techniques for this problem. The CLIQUE algorithm of Agrawal, Gehrke, Gunopulos and Raghavan (1998, 2005) was likely the first algorithm to address the projective clustering problem. The algorithm uses a bottom-up strategy that initially finds clusters in projections onto single dimensions of the space. Clusters previously found in k-dimensional spaces are used to find clusters in $(k+1)$-dimensional spaces. Clusters are built with one additional dimensional at each step, until no more dimensions can be added. One drawback to the algorithm is that it is exponential in the number of dimensions in the output cluster. Other bottom-up algorithms include ENCLUS (Cheng et al., 1999) and MAFIA (Goil et al., 1999; Nagesh et al., 2001).

Aggarwal, Wolf, Yu, Procopiuc and Park (1999) developed the PROCLUS algorithm for projective clustering using a top-down strategy based on medoids. The medoids are individual points from the data set selected to serve as surrogate centers for the clusters. After initializing the medoids, an iterative hill-climbing approach is used to improve the set of points used as medoids. A final refinement stage generates the final projective clusters. This algorithm requires both the number of clusters and the average number of dimensions as inputs. Additional methods that use top-down strategies include ORCLUS (Aggarwal and Yu, 2000) (which considers non-axis parallel subspaces) and FINDIT (Woo et al., 2004).

Procopiuc, Jones, Agarwal and Murali (2002) developed the DOC and FastDOC algorithms for pro-

jective clustering. One of their contributions was a definition of an optimal projective cluster, which has the following properties:

1. It has a minimum density α (a fraction of the number of points in the data set).
2. It has a width of no more than w in each dimension in which it congregates.
3. It has a width larger than w in each dimension in which it does not congregate.
4. Among clusters that satisfy the above criteria, it maximizes a quality function $\mu(|C|,|D|)$, where $|C|$ is the number of points in the cluster and $|D|$ is the number of dimensions in which the cluster congregates.

While any function that is monotonically increasing in each variable can be used as the quality function, Procopiuc et al. use $\mu(|C|,|D|) = |C| \cdot (1/\beta)^{|D|}$, where $0 < \beta < 1$ is a parameter that determines the trade-off between the number of points and the number of dimensions in the optimal cluster. If a cluster C_1 congregates in f fewer dimensions than C_2, it must have $|C_1| > |C_2|/\beta^f$ points to surpass the score for C_2. An optimal cluster must have at least $1/\beta$ as many points as another cluster if it congregates in exactly one less dimension. We use the same definition in this work. Note, however, that the DOC algorithm is restricted to $\beta < 0.5$, while our algorithm is not.

With the above definition for an optimal cluster, Procopiuc et al. developed a Monte Carlo algorithm for finding an optimal cluster with high probability. Their algorithm uses two loops, the outer loop selects a single seed point and the inner loop selects an additional set of points from the data called the *discriminating set*. The seed point and all of the discriminating set must belong to the optimal cluster for the trial to succeed. The dimensions in the cluster are determined by finding the dimensions in which all of the points in the discriminating set are within w of the seed point (allowing an overall cluster width of $2w$). Given these dimensions, the cluster is estimated by finding all points in the data set within w of the seed point in these dimensions.

For fixed α, β and ε (the arbitrarily low probability of failure), the DOC algorithm has complexity $O(nd^{1+\frac{\log(\alpha/2)}{\log 2\beta}})$, where n is the number of points in the data set and d is the number of dimensions. In order to improve the speed of the algorithm, Procopiuc et al. propose the FastDOC algorithm in which a heuristic is used limiting the number of trials in each inner loop to d^2. This reduces the complexity of the algorithm to $O(nd^3)$.

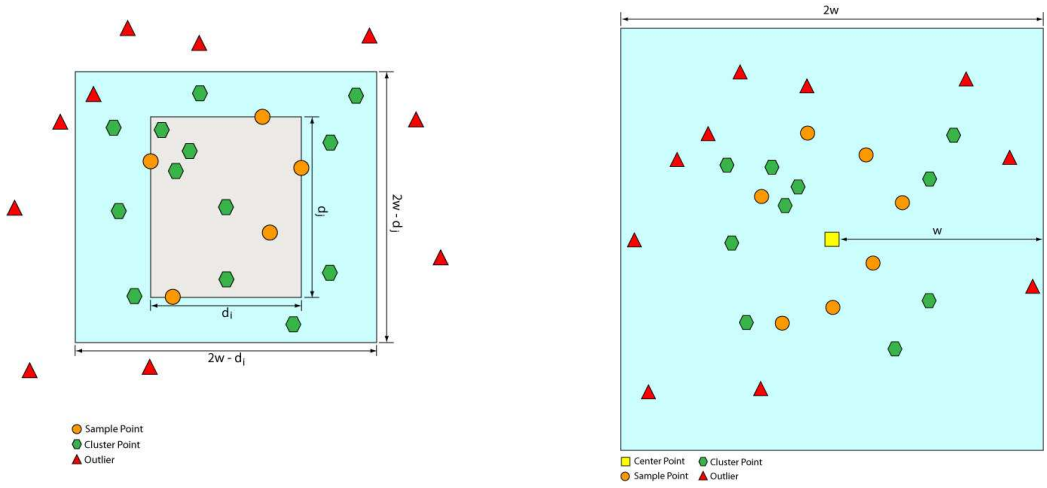
Yiu and Mamoulis (2005) describe the CFPC algorithm. As in the DOC algorithm, an outer loop is used that samples individual points from the data set. However, they replace the inner loop from the DOC algorithm with a technique adapted from mining frequent itemsets to determine the cluster dimensions and points (assuming that the sample point is from the optimal cluster). No formal analysis of the computational complexity is given, but Yiu and Mamoulis reported improved speeds in comparison to PROCLUS and FastDOC. However, the speed of the method is dependent on subsampling the data set to a small size (1000 data points) before processing.

3 APPROACH

Our approach to the projective clustering problem is a Monte Carlo algorithm inspired by the DOC algorithm of Procopiuc et al. (2002). In each trial of the algorithm, a small set of data points is sampled. Following Procopiuc et al., we will call this the discriminating set. Note, however, that (unlike DOC) we have no outer loop where individual seed points are sampled. A trial can succeed only if all of the points in the discriminating set are from the same cluster, among other conditions. Many trials are performed in order to achieve a high probability of success. In Section 4 we develop probability guarantees for the number of trials necessary. These guarantees hold for optimal clusters with sufficient density α as a percentage of the points in the data set.

In each trial, the discriminating set is used to determine the set of congregating dimensions for a hypothesized projective cluster. This is performed by selecting the dimensions in which the span of the discriminating set is less than the desired cluster width w. This requires only finding the minimum and maximum value in each of the dimensions. Note that this is a considerable improvement over the DOC algorithm, in which the width of the sheath used to determine the congregating dimensions is $2w$. The narrower sheath helps prevent incorrect dimensions from being included. This also allows us to use a larger value for β (the fraction of points necessary to remain in a cluster to add another congregating dimension). The DOC algorithm is limited to using $\beta < 0.5$. We are not constrained, except that β can never exceed one, for obvious reasons.

Given the hypothesized congregating dimensions, we need to determine the points in the cluster. The cluster points will not necessarily fall within the bounds given by the discriminating set, since the discriminating set will not generally include the extremal points in all congregating dimensions of the cluster. If the span of the discriminating set in cluster dimension



(a) (b)

Figure 2: Comparison between the SEPC and DOC algorithms for determining cluster dimensions and cluster points using a discriminating set. (a) The SEPC algorithm uses a sheath of width w to determine if a discriminating set congregates in a dimension. When the set congregates, a larger sheath with width between w and $2w$ is used to determine additional data points that are added to the cluster. (b) The DOC and FastDOC algorithms use a sheath with width $2w$ both to determine if the discriminating set congregates in a dimension and to find additional data points that are added to the cluster.

i is d_i, we need to allow an additional $w - d_i$ on each side of the span to ensure that any possible cluster with width w is detected. See Fig. 2(a). This can allow clusters that are wider than w, since the sheath for adding points to the cluster is $2w - d_i$. However, few (if any) outliers will be included, since they must congregate with the cluster in all of the congregating dimensions. In comparison, the DOC algorithm continues using a sheath with width $2w$ to determine which points are included in the cluster. See Fig. 2(b).

The cluster found in each trial is given a score using the DOC metric and retained if the score is sufficiently high. Typically, we retain only the top scoring cluster over all of the trials. This cluster is removed from the data and further clusters are found using additional iterations of the algorithm in order to generate disjoint clusters[1]. The process continues until no more clusters surpassing some criteria are found. Any remaining points can be classified as outliers or added to the cluster to which they are closest. If overlapping clusters are desired, multiple clusters can be located in a single iteration of the algorithm with a simple modification. In this case, we store not just the best cluster found, but all clusters of sufficient quality that are qualitatively different.

If the highest scoring cluster found in an iteration has density less than α, but meets the score criterion, we report it, even if clusters exist with density larger than α. If such clusters are discarded, then no probability guarantees can be made. Consider a large cluster with a density of exactly α. If a subcluster exists with one fewer point and one more dimension, then this subcluster will achieve a greater score, unless β is set arbitrarily close to one. In addition, this subcluster will usually prevent the larger cluster from being found. Unless the discriminating set contains the single additional point in the larger cluster, the congregating dimensions will be hypothesized to include the additional dimension that the remaining points congregate in and this will exclude the additional point from the detected cluster. If the subcluster is discarded as too sparse, then the larger cluster cannot be detected with high probability. Note that this is true not just for our algorithm, but for all similar Monte Carlo algorithms. Instead of discarding the cluster, we allow the smaller cluster to supersede the larger cluster, since it has a higher score. Our probability guarantees are such that, if an optimal cluster with density α exists in the data set, we will report it *or* a higher scoring cluster with high probability.

4 ALGORITHM

The basic form of the SEPC algorithm is given in Figure 3. The input to the algorithm includes the set of points P, the number of dimensions d, the maximum width of a cluster w, the number of trials k_{trials}, and

[1]Reducing the size of the data set improves the ability of the algorithm to find small clusters. If small clusters are not desired, then the cluster density can be computed with respect to the original number of points in the data set.

the cardinality of each discriminating set s. In each trial, we randomly sample (without replacement) s points from the data. Within the discriminating set, the set of dimensions in which the points congregate is determined (line 4). For each dimension, bounds are determined on the span of the cluster (infinite bounds are used for dimensions in which the cluster does not congregate). The points in the cluster are determined (line 10) by finding the intersection of the point set with the Cartesian product of each of the dimension ranges r_j. Finally, the cluster is saved if it is the best found so far.

SEPC(P, d, w, k_{trials}, s):
1 Let $\mu_{best} = 0$.
2 For $i = 1$ to k_{trials}:
3 Sample $S_i \subset P$ randomly, with $|S_i| = s$.
4 Let $D_i = \{j \mid \forall p,q \in S_i, |p_j - q_j| \leq w\}$.
5 For $j = 1$ to d:
6 If $j \in D_i$:
7 Let $r_j = [\max_{p \in S_i} p_j - w,\ \min_{p \in S_i} p_j + w]$.
8 Else:
9 Let $r_j = [-\infty, \infty]$.
10 Let $C_i = P \cap \prod_{1 \leq j \leq d} r_j$.
11 If $\mu(|C_i|, |D_i|) > \mu_{best}$:
12 Let $\mu_{best} = \mu(|C_i|, |D_i|)$, $C_{best} = C_i$, $D_{best} = D_i$.
13 Return C_{best}, D_{best}.

Figure 3: The basic SEPC algorithm for finding a projective cluster. This algorithm finds a single cluster and may be iterated after removing the cluster (for disjoint clustering). To detect non-disjoint clusters, multiple clusters can be found in a single iteration of the algorithm. In this case, all clusters of sufficient quality should be saved in lines 11-12, unless the clusters substantially overlap.

4.1 Number of tTrials

A crucial parameter to determine is the number of trials that are sufficient to find a projective cluster with high probability. We will assume that a cluster exists containing at least $m = \lceil \alpha n \rceil$ points, since we otherwise make no claims about the likelihood of a cluster being found. For a trial to succeed, we need for two conditions to hold. First, the trial must select only points within the projective cluster (allowing us to find all of the dimensions in which the cluster is formed). Second, the trial must not select only points that randomly congregate in any dimension that is not a congregating dimension of the full projective cluster. For any fixed s, a lower bound can be placed on the probability of both conditions holding:

$$P_{trial} \geq \left(\frac{C_s^m}{C_s^n}\right)\left(1 - \frac{C_s^l}{C_s^m}\right)^d, \tag{3}$$

where $l = \lfloor \beta m \rfloor$ and C_k^j is the number of k-combinations that can be chosen from a set of cardinality j. The first term of Eq. 3 is a lower bound on probability of the first condition holding. The second term is a lower bound on the probability of the second condition holding, given that the first condition holds. This term is computed based on the fact that, for an optimal cluster of c points, no more than $\lfloor \beta c \rfloor$ points in the cluster can be within w of each other in any dimension that is not a congregating dimension of the cluster (otherwise, this subset would form a higher scoring projective cluster that includes the dimension)[2]. The second term is taken to the dth power, since the random clustering could occur in any of the d dimensions (except for those that the projective cluster does congregate in).

For large data sets, Equation 3 is well approximated by:

$$P_{trial} \geq \alpha^s (1 - \beta^s)^d. \tag{4}$$

After k_{trials} iterations, we want the probability that none of the trials succeed to be below some small constant ε (e.g., 10^{-2}). This is achieved with:

$$(1 - P_{trial})^{k_{trials}} \leq \varepsilon, \tag{5}$$

which yields:

$$k_{trials} \geq \frac{\log \varepsilon}{\log(1 - P_{trial})}. \tag{6}$$

4.2 Discriminating Set Cardinality

The above equations are valid for discriminating sets with arbitrary cardinality greater than one. However, the number of trials varies greatly depending on the cardinality of each discriminating set. If the cardinality is large, then it will be unlikely that the points in the set will all be in the desired cluster. If the cardinality is small, then it is likely that the points will congregate in (at least) one dimension in which the cluster does not. The optimal value can be obtained by computing k_{trials} for a small number of values and using the value that requires the fewest trials. We use a heuristic to approximate the optimal value. The heuristic sets the cardinality such that the probability of the discriminating set congregating in even one of the incorrect dimensions is bounded by 3/4. (The second term in Eq. 3 usually dominates the computation of k_{trials}.)

With this heuristic we have:

$$\left(1 - \frac{C_s^l}{C_s^m}\right)^d \geq \frac{1}{4}, \tag{7}$$

[2]Note that there may be more than C_s^l combinations of such points, but only if the cluster contains more than m points. This probability still serves as a lower bound.

which yields:

$$\frac{l!(m-s)!}{m!(l-s)!} \leq 1-4^{\frac{-1}{d}} \tag{8}$$

and

$$\sum_{i=l-s+1}^{m-s} \log i - \sum_{i=l+1}^{m} \log i \leq \log(1-4^{\frac{-1}{d}}). \tag{9}$$

$$\sum_{i=l-s+1}^{l} \log i - \sum_{i=m-s+1}^{m} \log i \leq \log(1-4^{\frac{-1}{d}}). \tag{10}$$

From this, we derive the following approximation to the optimal value of s:

$$s_{est} \approx \frac{\log(1-4^{\frac{-1}{d}})}{\log l - \log m} \tag{11}$$

$$= \frac{\log(1-4^{\frac{-1}{d}})}{\log\lfloor \beta\lceil \alpha n\rceil\rfloor - \log\lceil \alpha n\rceil} \tag{12}$$

$$\approx \frac{\log(1-4^{\frac{-1}{d}})}{\log\beta} \tag{13}$$

$$\approx \frac{\log(d^{-1}\ln 4)}{\log\beta} = \frac{\log(d/\ln 4)}{\log(1/\beta)} \tag{14}$$

Table 1: Optimal and approximated values for s with $\alpha = 0.1$, $n = 100,000$, and varying values for d and β.

$d\downarrow$	$\beta\rightarrow$	0.15	0.2	0.25	0.3	0.35
50	s_{opt}	**2**	**2**	**3**	**3**	**3**
	s_{est}	*1.9*	*2.2*	*2.6*	*3.0*	*3.4*
100	s_{opt}	**2**	**3**	**3**	**3**	**4**
	s_{est}	*2.3*	*2.7*	*3.1*	*3.6*	*4.1*
200	s_{opt}	**3**	**3**	**4**	**4**	**4**
	s_{est}	*2.6*	*3.1*	*3.6*	*4.1*	*4.7*
400	s_{opt}	**3**	**4**	**4**	**4**	**5**
	s_{est}	*3.0*	*3.5*	*4.1*	*4.7*	*5.4*

Table 1 compares the estimate for the optimal cardinality of the discriminating set with the true optimum with respect to the number of trials as specified by Eq. 6. For this comparison, α was held constant at 0.1 and n at 100,000. The levels of d and β were varied (since these are the most influential in determining the optimal number of trials). Our approximation (when rounded to the nearest whole number) overestimates the optimal number in some cases. Even in these cases, the average percentage difference in the number of trials is less than 30% from the optimal value. Over all of the cases considered, the average percentage difference from the optimal number of trials is 4.42%.

Table 2 compares the cardinality of the discriminating set and required number of trials between the SEPC algorithm and the DOC algorithm. For the DOC algorithm, we use the total number of points that are sampled in testing a cluster, thus including the seed point. The number of trials for the DOC algorithm includes both the inner iterations and the outer iterations (i.e., the total number of trials performed before a cluster is returned). This yields the following equations for DOC (we use our notation rather than that of Procopiuc et al. (2002)):

Table 2: Comparison of discriminating set cardinality and number of trials required in SEPC and DOC algorithms with $\alpha = 0.1$, $n = 100,000$, and varying values for d and β. Note that the cardinality for the DOC algorithm includes the seed point and the number of trials includes all outer iterations per seed point.

d	Alg.	$\beta\rightarrow$	0.15	0.2	0.25	0.3	0.35
50	SEPC	s_{opt}	2	2	3	3	3
		k_{trials}	$1.4\cdot10^3$	$3.5\cdot10^3$	$1.0\cdot10^4$	$1.8\cdot10^4$	$4.1\cdot10^4$
	DOC	s	5	7	8	11	14
		k_{trials}	$4.4\cdot10^6$	$1.7\cdot10^9$	$3.5\cdot10^{10}$	$2.8\cdot10^{14}$	$2.3\cdot10^{18}$
100	SEPC	s_{opt}	2	3	3	3	4
		k_{trials}	$6.5\cdot10^3$	$1.0\cdot10^4$	$2.2\cdot10^4$	$7.1\cdot10^4$	$2.1\cdot10^5$
	DOC	s	6	7	9	12	16
		k_{trials}	$8.9\cdot10^7$	$1.7\cdot10^9$	$7.1\cdot10^{11}$	$5.7\cdot10^{15}$	$9.1\cdot10^{20}$
200	SEPC	s_{opt}	3	3	4	4	4
		k_{trials}	$9.0\cdot10^3$	$2.3\cdot10^4$	$1.0\cdot10^5$	$2.3\cdot10^5$	$9.4\cdot10^5$
	DOC	s	6	8	10	13	18
		k_{trials}	$8.9\cdot10^7$	$3.5\cdot10^{10}$	$1.40\cdot10^{13}$	$1.1\cdot10^{17}$	$3.6\cdot10^{23}$
400	SEPC	s_{opt}	3	4	4	4	5
		k_{trials}	$1.8\cdot10^4$	$8.7\cdot10^4$	$2.2\cdot10^5$	$1.2\cdot10^6$	$3.8\cdot10^6$
	DOC	s	7	9	11	15	20
		k_{trials}	$1.8\cdot10^9$	$7.1\cdot10^{11}$	$2.8\cdot10^{14}$	$4.5\cdot10^{19}$	$1.5\cdot10^{26}$

$$s^{(DOC)} = \left\lceil 1 + \frac{\log 2d}{\log\frac{1}{2\beta}} \right\rceil \tag{15}$$

$$k_{trials}^{(DOC)} = \left\lceil \frac{2}{\alpha} \right\rceil \left\lceil \left(\frac{2}{\alpha}\right)^{s-1} \ln 4 \right\rceil \tag{16}$$

It can be seen in the table that the SEPC algorithm requires at least 3 orders of magnitude less trials in each of the cases considered and almost 20 orders of magnitude less trials for the most complex case considered. It should be noted that the FastDOC algorithm uses a heuristic where the number of inner iterations is limited to d^2 (Procopiuc et al., 2002). However, this removes any probability guarantees and our experiments (Sec. 5) indicate that this results in missed clusters.

4.3 Efficiency

For fixed α, β, and d, the running time of the algorithm is $O(n)$, since the number of trials does not depend on n. However, the running time has a more interesting relationship with α, β, and d. The number of trials is:

$$k_{trials} = \left\lceil \frac{\log \varepsilon}{\log(1-\alpha^s(1-\beta^s)^d)} \right\rceil, \qquad (17)$$

with $s \approx \log(d/\ln 4)/\log(1/\beta)$. Since $\ln(1-\delta) < -\delta$ for $0 < \delta < 1$, we have:

$$k_{trials} < 1 + \frac{\ln 1/\varepsilon}{\alpha^s(1-\beta^s)^d} \qquad (18)$$

Recall that s was chosen specifically to make $(1-\beta^s)^d \geq 1/4$. In fact, the bound is not tight, since d includes the (unknown) number of dimensions in which the cluster congregates. This yields:

$$\begin{aligned} k_{trials} &< 1 + \frac{4\ln 1/\varepsilon}{\alpha^{\lceil \log(d/\ln 4)/\log(1/\beta)\rceil}} && (19)\\ &< 1 + 4\alpha^{\log(d/\ln 4)/\log(\beta)-1} \ln \frac{1}{\varepsilon} && (20)\\ &= 1 + \frac{4}{\alpha}\left(\frac{d}{\ln 4}\right)^{\frac{\log\alpha}{\log\beta}} \ln \frac{1}{\varepsilon} && (21) \end{aligned}$$

Equations 19-21 imply that the number of trials required is $O(\frac{1}{\alpha} d^{\frac{\log\alpha}{\log\beta}} \log\frac{1}{\varepsilon})$. The complexity of the overall algorithm is $O(\frac{1}{\alpha} n d^{1+\frac{\log\alpha}{\log\beta}} \log\frac{1}{\varepsilon})$, since each trial in the algorithm requires $O(nd)$ time.

The complexity of the SEPC algorithm can be contrasted with the DOC algorithm, which is $O(\frac{1}{\alpha} n d^{1+\frac{\log(\alpha/2)}{\log 2\beta}} \log\frac{1}{\varepsilon})$. With $\alpha = 0.1$, $\beta = 0.25$ and fixed ε, our algorithm is $O(nd^{2.661})$, while the DOC algorithm is $O(nd^{5.322})$. The difference between the exponents increases as the problem becomes more difficult (decreasing α and/or increasing β). For these parameters, our algorithm also has a lower computational complexity than the FastDOC algorithm, which is $O(nd^3)$ and, as we will see, much better cluster detection performance.

4.4 Parameter Setting

Given the increase in the number of trials necessary as β increases, it might be questioned why a larger value is desirable. A larger value is necessary in some cases to prevent subsets of a cluster from producing a larger score than the "correct" cluster owing to random accumulation in one or more dimensions.

Consider a projective clustering problem in which each dimension ranges over [0, 1]. When determining the points in a cluster, we use a sheath of width $v \leq 2w$ in the dimensions in which the discriminating set congregates. (The DOC and FastDOC algorithms use $v = 2w$.) A random outlier will fall within this sheath (in a single dimension) with probability v. When β is less than v, the inclusion of an incorrect dimension usually leads to a cluster with a better score than the desired projective cluster, since it will include an extra dimension (although fewer points). The fraction of points in the smaller cluster that are captured from the larger cluster is expected to be v (although it varies around this fraction given random data). If the larger cluster has score μ_1, the smaller cluster is expected to have a score of approximately $\frac{v}{\beta}\mu_1$. For example, if $\beta = 0.25$ and $v = 0.30$, it is nearly certain that the inclusion on an incorrect dimension in D_i (the hypothesized set of dimensions in the ith trial) will result in an output cluster with one or more extra dimensions and fewer points[3].

It is worth noting that the presence of trials with such extra dimensions is common. We set s such that the probability of finding at least one incorrect dimension is no more than 3/4 in a discriminating set that is entirely part of the optimal cluster. This is normally acceptable, as long as we find at least one case without such an incorrect dimension. However, if β is too small, it will result in the cluster subset scoring higher than the full projective cluster.

Procopiuc et al. (2002) noticed this effect in their experiments. Their solution was to generate more output clusters than were present in the input data in order to find all of the cluster points, even though they are broken into multiple output clusters.

Another heuristic that is useful in some cases is to discard any cluster that does not surpass some minimum cardinality. While this would remove the probability guarantee for finding clusters, it helps eliminate clusters with incorrect dimensions, since they are significantly smaller than the true clusters. In cases where the larger cluster is found with frequency comparable to the smaller cluster, it is likely to be detected even if the smaller cluster is also detected (and discarded).

4.5 Optimizations

An optimization that is useful is to limit the number of passes through the entire data set, particularly if it is too large to fit in memory and accessing it from a disk is time consuming. The idea here is to generate many discriminating sets S_i during a single pass through the

[3] In this case, we expect approximately 30% of the cluster points to remain in the cluster with the extra dimension, but the scoring function increases the score by a factor of $1/\beta = 4$.

data. After the discriminating sets are constructed, they can be examined jointly in a single pass through the data (and further discriminating sets can be generated during this pass). An additional parameter is required with this optimization, k_{sets}, which is the number of discriminating sets to consider simultaneously.

In the FastDOC algorithm, Procopiuc et al. (2002) propose to consider only one discriminating set for each outer iteration of the algorithm. This discriminating set is chosen by finding the one that congregates in the largest number of dimensions. We can similarly consider only one of the k_{sets} discriminating sets each full pass through the data in the SEPC algorithm. This optimization removes the probability guarantees, since not every trial is fully evaluated, but good results can be achieved since the discriminating sets yielding many cluster dimensions often yield high scoring clusters. The speedup achieved is roughly k_{sets}, since scanning the data set for potential cluster points dominates the running time.

Although we could use a heuristic to limit the number of trials as is done in FastDOC (Procopiuc et al., 2002), we choose not to do this, since the number of trials required is much lower in the SEPC algorithm and it would reduce the likelihood of finding a projective cluster.

5 EXPERIMENTS

This section discusses our experiments on real and synthetic data. Most experiments were run on a 2 GHz Intel Core Duo CPU with 2 GB RAM running Windows Vista. We compare primarily against the FastDOC algorithm, since previous experiments have shown it to be superior to algorithms such as PROCLUS and ORCLUS (when applied to clusters that are in axis-aligned subpaces) (Procopiuc et al., 2002).

5.1 Random Data Generation

For synthetic data, the data sets were generated following the methodology originally described by Aggarwal et al. (1999) and used (with some variations) by Procopiuc et al. (2002). Each data set is composed of 100,000 points in 200 dimensions and each dimension has range [0, 100].

Clusters were generated with an average of 40 dimensions in which they congregate, but the actual number of dimensions was varied as a Poisson random variable. After generating the dimensions in which the first cluster congregates, each remaining cluster was generated such that half of the congregating dimensions in each new cluster were the same as congregating dimensions in the previously generated cluster. The remaining dimensions were selected randomly. In the congregating dimensions, the points were generated according to a normal distribution with $\sigma \in [2,4]$ (uniformly distributed). Cluster centers, outliers, and non-congregating dimensions were uniformly distributed over [0, 100].

When multiple clusters were generated, the cluster sizes were generated to be proportional to independent exponential random variables. We use the technique of Procopiuc et al. to enforce significant variation in cluster size (Procopiuc et al., 2002). This method divides the clusters into two sets and removes points from clusters in one set while adding them to the clusters in the other set (prior to computation of the point locations).

5.2 Multiple Clusters (Random Data)

Our initial experiments were similar to those in previous work (Aggarwal et al., 1999; Procopiuc et al., 2002) with 5 clusters totaling 95,000 points and 5,000 outliers. We tested our implementations of SEPC and FastDOC, with the algorithms continuing to generate clusters until no clusters were found having a score better than would be achieved with the minimum number of dimensions (20 in these experiments) and the minimum density (0.1). After each cluster was found, the points in the cluster were removed from the data before detecting another cluster. This is a relatively easy problem in the sense that the optimal cluster remaining in the data at any iteration rarely (if ever) comes close to the minimum density that the algorithms seek to allow (0.1).

The experiments of Procopiuc et al. used $w = 15$ and $\beta = 0.25$. We note that this can be problematic in an algorithm that allows points to congregate in a dimension over a range of $2w$ (as does FastDOC in all cases and SEPC in the worst case). The reason is that this allows 30% of the points in a dimension to congregate at random around any cluster center. If only above 25% of the points are required to form a higher scoring cluster, then the optimal clusters will always include extra dimensions in which the points are in fact uniformly distributed. Ironically, FastDOC does not usually find these clusters (even if they meet the definition of an optimal cluster), since the number of trials performed is insufficient to find small clusters. We use two sets of parameters ($w = 10, \beta = 0.25$) and ($w = 15, \beta = 0.35$) in which this problem is unlikely. With the first set of parameters, we restrict the standard deviation of the points in the congregating dimensions to $\sigma = 0.2$. Ten experiments were run for each set of parameters, with 2000 outer iterations in

each experiment.

Once clusters were found, each point in the data set was classified as belonging to one of the clusters or to the outlier set. We allowed nearly identical clusters to merge after all clusters were extracted to allow for cases where a subcluster with an additional dimensions was detected along with (or instead of) the full cluster. A one-to-one correspondence between the detected clusters and the input clusters was then formed and the fraction of correctly classified points was calculated. If the number of clusters was uneven, all points in the extra cluster(s) were considered to be classified incorrectly.

For the first set of parameters ($w = 10, \beta = 0.25$), SEPC classified over 99.99% of the points correctly in every experiment[4]. All of the incorrect classifications occurred because cluster points were classified as outliers owing to points that were unusually far from the cluster center. (No constraint was made during data generation enforcing a maximum radius.)

FastDOC classified 95.5% of the points correctly, on average. Most of the incorrectly classified points were cluster points that were classified as outliers. This occurred more frequently for FastDOC, since it forces the cluster center to be at the location of a seed point, which is not necessarily at the true center of the cluster. Additional errors occurred owing to misclassification between clusters, and some small clusters that were missed entirely (clusters as small as 747 points were observed). In this data, the SEPC algorithm required 31.3 minutes, on average, while the FastDOC algorithm required 107.5 minutes, on average.

For the second set of parameters ($w = 15, \beta = 0.35$), SEPC classified 99.96% of the points correctly. Most misclassifications were the result of cluster points being classified as outliers. FastDOC performed poorly in this case, detecting only 44.0% of the clusters and classifying 67.6% of the points correctly (typically the larger clusters were detected). Most failures occurred because several small clusters were not found, since a discriminating set of 17 points is required in FastDOC with $\beta = 0.35$ and $d = 200$. Too few trials were performed in order to find valid discriminating sets for these clusters. FastDOC required 54.8 minutes, on average, compared to 130.8 minutes for the SEPC algorithm. Some of the difference in running time can be attributed to the clusters missed by FastDOC, since each experiment ended as soon as an iteration was unable to find a cluster.

FastDOC performed better on this data set with $\beta = 0.25$ than with $\beta = 0.35$. However, the performance was still worse than SEPC. FastDOC achieved a 95.5% success rate in this case, although it did not find the clusters that met the definition of an optimal cluster, since these had more congregating dimensions and fewer points than the input clusters. The average running time increased to 102.1 minutes in this case, since more iterations of the algorithm were performed.

[4]Only 8 errors were made over the 10^6 points in all of the experiments.

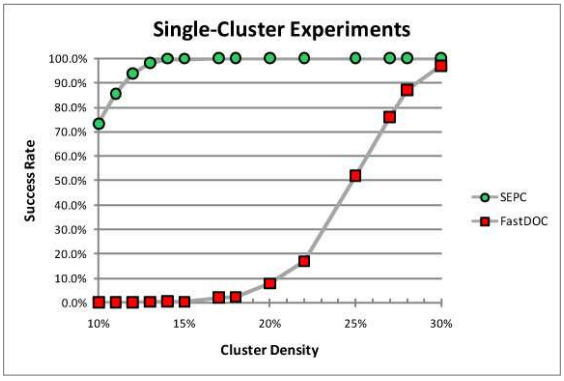


Figure 4: Performance comparison between SEPC and FastDOC for data sets containing a single cluster.

In none of these experiments was an outlier ever classified as part of one of the detected clusters, even with relaxed criteria for adding points to clusters. This is to be expected, since uniformly distributed outliers are captured by clusters with very low probability. (A cluster with 20 dimensions that captures 30% of the points in each dimension would capture a random outlier with probability $0.3^{20} = 3.49 \times 10^{-11}$.)

5.3 Single Clusters (Random Data)

Based our analysis, we hypothesized that the SEPC algorithm would succeed in cases where the FastDOC algorithm would fail owing to the FastDOC heuristic that the number of inner iterations was limited to d^2 in order to speed up the algorithm. These failures should occur in cases where the clusters are close to the minimum density α, since reducing the number of trials significantly would make it unlikely that any trial would find a valid discriminating set.

To test our hypothesis, we ran a set of experiments using data sets with a single cluster and a high fraction of outliers. In these experiments, each data set had one cluster C_1 with between 10,000 and 30,000 points that congregated in $D_1 = 40$ dimensions. Each cluster size was tried 1,000 times with both SEPC and FastDOC, with the following algorithm parameters: $\alpha = 0.1$, $\beta = 0.25$, and $w = 15$. An experiment was counted as a success if the detected clus-

ter (C_{best}) met the following criteria: $|C_{best}| \geq 7000$, $|D_{best} \cap D_1| \geq 36$. (Since the clusters have a normal distribution with $\sigma \in [2,4]$ in each of the cluster dimensions, a significant number will not fall within the cluster width of $w = 15$ in each congregating dimension.)

Figure 4 shows the results of these experiments. It can be observed that SEPC is much more effective at detecting small clusters. Some failures occurred for clusters comprising less than 15% of the data set owing, in part, to clusters that exceeded the maximum width *w*. FastDOC failed to detect the small clusters, since the number of trials was not sufficient to ensure a high probability of success. In fact, the FastDOC algorithm had failures up to clusters comprising 30% of the data set.

The observed success rates for FastDOC are consistent with the theoretical probability. For a cluster containing 20% of the data set, a trial with a seed point and a discriminating set of 9 points will have a probability of 1.02×10^{-7} of sampling points that are all within the cluster. Over 20 seed points and 40,000 discriminating sets per seed point, the probability of at least one combination of a seed point and a discriminating set being entirely from the cluster is 7.8%. (For a cluster, containing 25% of the data set, the probability rises to 48.7%.)

For each data point, the SEPC algorithm ran in fewer seconds than the FastDOC algorithm averaging 8 seconds per experiment as compared to 40 seconds for FastDOC. The SEPC running times increased with the cluster size (note that α was held constant in the algorithm). The reason is that more full scans of the data set were required as the cluster size increased. The full DOC algorithm was not tested, since it would require greater than 10^{13} trials for these parameters.

5.4 Image Segmentation Data

Our final set of experiments used the SEPC algorithm to perform clustering in a data set consisting of 3×3 pixel regions from images that were randomly selected from a database of outdoor images (Asuncion and Newman, 2007). Each region in the data set is classified as one of the following classes: Brickface (B), Sky (S), Foliage (F), Cement (C), Window (W), Path (P), and Grass (G). This data set has 19 attributes (dimensions) and 2,100 points (300 of each class). Yiu and Mamoulis used this data set to test FastDOC and other projective clustering techniques (Yiu and Mamoulis, 2005). For their algorithm (CFPC) and FastDOC, they used $\alpha = 0.13$, $\beta = 0.25$, and $w = 0.25$. Like Yiu and Mamoulis, we preprocessed the data to be in the range [0, 1] using min-max normalization. We use $\beta = 0.25$ and $w = 0.19$ for the SEPC algorithm.[5] The parameters were tested over 100 independent experiments. For each experiment, seven clusters were detected in the data. Bipartite matching between the detected clusters and the correct clusters was performed. Over the 100 trials, 77.3% of the data points were assigned to the correct cluster.

Table 3: Confusion matrix for classifying pixel regions using the SEPC algorithm. Results are shown for a representative experiment with 80.7% success rate. (The average success rate with these parameters over all experiments was 80.5%.)

Input	C_3	C_4	C_5	C_6	C_7	C_1	C_2
B	251	0	2	4	43	0	0
S	0	300	0	0	0	0	0
F	16	0	217	35	31	1	0
C	49	0	1	222	10	18	0
W	109	0	67	35	89	0	0
P	0	0	0	35	0	265	0
G	0	0	0	1	2	0	297

Table 4: Comparison of SEPC classification results. All results except for SEPC are from Yiu and Mamoulis. SEPC 1 is the result of our algorithm without training. SEPC 2 is the result using a small training set to determine the clusters.

Technique	Accuracy
SEPC	77.3%
CFPC	69.3%
FastDOC	64.1%
PROCLUS	62.1%
KMED	60.2%

Table 3 shows a confusion matrix for a representative experiment. This experiment achieved 78.1% success. The results are perfect on the Sky class and very good on the Grass class. The worst performance is on the Window class with only 89 of 300 points clustered together. This is consistent with the results found by Yiu and Mamoulis and this group appears to be the most difficult to classify based on the given attributes.

Table 4 compares the SEPC results to the results published by Yiu and Mamoulis (2005). Our algorithm achieves a higher accuracy than any of the results reported in (Yiu and Mamoulis, 2005) by a significant margin.

We also tested the SEPC algorithm in a supervised training mode, where a 210 point training set

[5] We expect SEPC to perform worse with the parameters tuned by Yiu and Mamoulis (2005) and CFPC to perform worse with the parameters we use. We compare SEPC against the best results reported by Yiu and Mamoulis.

was used to determine the cluster dimensions and locations and a 2100 point testing set (disjoint from the training set) was used for classification. In this experiment, 74.8% of the points were classified correctly. This demonstrates that the algorithm is able to determine the cluster properties from a small set of examples and apply them to previously unknown points. The performance is lower than the unsupervised experiments owing to the smaller set of points used to determine the clusters.

6 CONCLUSIONS

We have presented a new algorithm called SEPC for locating projective clusters using a Monte Carlo method. The algorithm is straightforward to implement and has low complexity (linear in the number of data points and low-order polynomial in the number of dimensions). In addition, the algorithm does not require the number of clusters or the number of cluster dimensions as input and does not make assumptions about the distribution of cluster points (other than that the clusters have bounded diameter). The algorithm is widely applicable to projective clustering problems, including the ability to find both disjoint and non-disjoint clusters. The performance of the SEPC algorithm surpasses previously reported results on both synthetic and real data.

REFERENCES

Aggarwal, C. C., Wolf, J. L., Yu, P. S., Procopiuc, C., and Park, J. S. (1999). Fast algorithms for projected clustering. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 61–72.

Aggarwal, C. C. and Yu, P. S. (2000). Finding generalized projected clusters in high dimensional spaces. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 70–81.

Agrawal, R., Gehrke, J., Gunopulos, D., and Raghavan, P. (1998). Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 94–105.

Agrawal, R., Gehrke, J., Gunopulos, D., and Raghavan, P. (2005). Automatic subspace clustering of high dimensional data. *Data Mining and Knowledge Discovery*, 11:5–33.

Asuncion, A. and Newman, D. (2007). UCI machine learning repository. University of California, Irvine, School of Information and Computer Sciences, http://www.ics.uci.edu/∼mlearn/MLRepository.html.

Beyer, K., Goldstein, J., Ramakrishnan, R., and Shaft, U. (1999). When is"nearest neighbor" meaningful? In *Proceedings of the 7th International Conference on Database Theory*, pages 217–235.

Cheng, C. H., Fu, A. W., and Zhang, Y. (1999). Entropy-based subspace clustering for mining numerical data. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 84–93.

Dash, M., Choi, K., Scheuermann, P., and Liu, H. (2002). Feature selection for clustering - a filter solution. In *Proceedings of the IEEE International Conference on Data Mining*, pages 115–122.

Ding, C., He, X., Zha, H., and Simon, H. D. (2002). Adaptive dimension reduction for clustering high dimensional data. In *Proceedings of the IEEE International Conference on Data Mining*, pages 147–154.

Goil, S., Nagesh, H., and Choudhary, A. (1999). MAFIA: Efficient and scalable subspace clustering for very large data sets. Technical Report No. CPDC-TR-9906-010, Northwestern University.

Jain, A. K., Murty, M. N., and Flynn, P. J. (1999). Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323.

Nagesh, H., Goil, S., and Choudhary, A. (2001). Adaptive grids for clustering massive data sets. In *Proceedings of the SIAM International Conference on Data Mining*.

Parsons, L., Haque, E., and Liu, H. (2004). Subspace clustering for high dimensional data: A review. *SIGKDD Explorations*, 6(1):90–105.

Procopiuc, C. M., Jones, M., Agarwal, P. K., and Murali, T. M. (2002). A Monte Carlo algorithm for fast projective clustering. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 418–427.

Woo, K.-G., Lee, J.-H., and Lee, Y.-J. (2004). FINDIT: A fast and intelligent subspace clustering algorithm using dimension voting. *Information and Software Technology*, 46(4):255–271.

Yiu, M. L. and Mamoulis, N. (2005). Iterative projected clustering by subspace mining. *IEEE Transactions on Knowledge and Data Engineering*, 17(2):176–189.