ProTraS: A Probabilistic Traversing Sampling Algorithm

Frédéric Ros[a,*], Serge Guillaume[b]

[a] *Laboratory PRISME, Orléans university, France*
[b] *ITAP, Irstea, Montpellier SupAgro, Univ Montpellier, Montpellier, France*

Abstract

In the process of knowledge discovery in big data, sampling is a technological brick that can be included in a more general framework to speed up existing algorithms and contribute to the scalability issue. Two challenging and connected problems arise with complexity: tuning and timing. *ProTraS*[1] is a new algorithm that fulfills both requirements. It is driven by a unique parameter, the sampling cost. The cost is overestimated by the maximum within group distance and the group cardinality. It is an iterative algorithm, at each step a new representative is added, chosen as the farthest-first traversal item from the representative in the group with the highest probability of cost reduction. The novel algorithm is robust to noise and time optimized. A detailed comparison with alternative algorithms, conducted on various synthetic and real world data sets, shows that the proposal yields competitive results in terms of quality of representation for clustering, sampling size and sampling time.

Keywords: Distortion cost, clustering, distance, density

1. Introduction

Defining a sample that behaves like the whole data set is a quite long-standing issue in data management. It has received fresh interest with the

*Corresponding author
Email addresses: frederic.ros@univ-orleans.fr (Frédéric Ros), serge.guillaume@irstea.fr (Serge Guillaume)

[1] A sample code is available at: http://frederic.rosresearch.free.fr/mydata/homepage/

challenge of big data, characterized by an increase in the volume, velocity and variety of the data. From the technical point of view, big data require to clean, analyze, secure and provide a granular access to massive data sets. Scalability is of major concern: the challenge is to propose fast, relevant and user-friendly processing techniques for knowledge discovery in such data. Various concepts and tools, including hardware components, e.g. for parallel processing, have been proposed. Sampling is a technological brick that can be included in a more general framework, such as feature selection or data aggregation, and contribute to the management of big data problems. A fast and easy to tune sampling algorithm could be used when human interaction is required, such as in active learning, or as a preprocessing step for more complex algorithms. Hierarchical clustering is an illustrative example: as it has, in its standard version, a $O(n^2)$ complexity, using a sample size 100 times smaller than the original set leads to a running time divided by 10000!

The first attempt was the Lloyd algorithm designed in 1957, but only published later (Lloyd, 1982). The goal was to find evenly spaced sets of points in subsets of Euclidean spaces, and partitions of these subsets into well-shaped and uniformly sized convex cells. It is closely related to the *k-means* algorithm, first proposed by Macqueen (1967) and made popular by Hartigan (1975), as both minimize the same objective function, called quantization distortion in signal processing. The main difference is that the Lloyd algorithm uses a Voronoi tessellation.

The Lloyd approach was generalized to any distribution, even with discrete components, by Linde, Buzo and Gray (Linde et al., 1980). Their technique does not involve any differentiation. This vectorial quantization yields an optimal codebook. The *LBG* algorithm is widely used in signal compression, either image or speech.

Recently, the concept of coreset, more precisely ε-coreset, was proposed (Agarwal et al., 2004). The idea is to quantify the distortion of a given monotonic measure when computed on a sample instead of on the whole set. Extensive research has been carried out to generate such coresets in different frame-

works (Har-Peled & Mazumdar, 2004).

Sampling and coresets have been applied to clustering. Clustering is an unsupervised task to organize, summarize and finally understand the data. In the field of big data, clustering algorithms are becoming more and more sophisticated in order to deal with complex data of various shapes and densities (Jain, 2010). Two challenging and connected problems arise with complexity: tuning and timing. Uniform sampling is the simplest and quickest way to proceed. Unfortunately, it requires very large sample sets to deal with shape and density variability. Smarter and more powerful algorithms have been proposed. These sampling algorithms, a recent review is available in (Ros & Guillaume, 2016b), are density or distance based and some of them combine the two notions under specific strategies. Density-based methods can be grouped in two main families for density estimation: space partition (e.g. grids, trees) (Palmer & Faloutsos, 2000; Ilango & Mohan, 2010) and local estimation, using neighborhood or kernel functions (Kollios et al., 2003). Both are highly sensitive to parameters, cell definition for grids, bandwidth or neighborhood for local estimators. With an inappropriate setting, these methods are either likely to sample noise or to miss regions of interest. Distance-based clustering algorithms are used for sampling with a number of samples much greater than the number of clusters. The most famous representative of this family is the *k-means* (Hartigan & Wong, 1979). Its sensitivity to initialization has been exhaustively investigated (Zahra et al., 2015; Arthur & Vassilvitskii, 2007; Celebi et al., 2013). Single scan approaches have also been proposed such as *leader* (Ling, 1981) clustering. The results are highly dependent on the distance threshold, even with improved versions (Sarma et al., 2013; Viswanath et al., 2013). Farthest first transversal, *fft*, algorithms (Rosenkrantz et al., 1977) are of interest as they do not require a distance parameter.

Moreover, improvements in accuracy often conflict with time performance, and response time is of major concern nowadays for data processing algorithms. The increased computational cost limits the application of some of the abovementioned algorithms to small or average size data sets. Several techniques

have been investigated to address these challenges (Lv et al., 2015; Zhong et al., 2015; Ma et al., 2015; Sarma et al., 2013). The stratification concept has been proposed to speed up algorithms with quadratic or exponential time complexity. To overcome sensitivity to splitting, extensions have been proposed to work with non disjoint partitions, using replication techniques (Machová et al., 2006).

In Feldman et al. (2011), an *Efficient Coreset Construction via Adaptive Sampling* was proposed, involving density and distance concepts while biasing the random sampling.

Two *fft* improved algorithms, *DIDES* (Ros & Guillaume, 2016b) and *DENDIS* (Ros & Guillaume, 2016a), were recently proposed to tackle the still open twofold challenge of tuning and timing. The two of them are iterative algorithms based on the hybridization of distance and density concepts. They differ in the priority given to distance or density, and in the stopping criterion defined accordingly. They benefit from an internal optimization that makes them faster than any competing approach. Their tuning is quite easy as they have only one meaningful and dimensionless parameter, called *granularity*. However, they also have limitations. First, the meaning that this unique parameter conveys is not the same for the two proposals, this may be an inconvenient from the user point of view. Second, they use internal heuristics based on parameters inferred from the data. Although the approach has been validated on a wide range of data sets with contrasted characteristics, heuristics elimination is always advisable.

The objective of this study is to propose a new sampling algorithm that is both easy to tune and scalable.

The proposed algorithm is only based on the sampling cost. It is a recursive partitioning algorithm: at each step a new representative is added to the sample until the cost falls below a threshold. The representative is chosen in the group with the highest probability of cost reduction. This probability is computed according to the within group distance and to the representativeness of the sample item. Therefore, *ProTraS* manages the concepts of distance and density in a new probabilistic way. The representative is the farthest item from the group representative. This ensures space coverage. *ProTraS* is explicitly

designed to produce a (k, ε)-coreset: the approximation level, ε, is its unique parameter and it is also used as the stopping criterion.

The paper is organized as follows. The *fft* algorithms, as well as the recent proposals *DIDES* and *DENDIS*, are detailed in Section 2. The relationship to the concept of coreset is analyzed in Section 3. The proposed algorithm is introduced in Section 4 and its properties are illustrated using synthetic data. Section 5 deals with the numerical experiments conducted on twenty-one synthetic and eight real world data sets. The sample representativeness is assessed with two well-known, and representative, algorithms, *DBSCAN* and *k-means*. The sensitivity to the unique parameter, the distortion cost, is analyzed and a detailed comparison with selected competitors is carried out. Finally, the main conclusions and open perspectives are stated in Section 6.

2. Previous work: *fft* algorithms

Sampling (Wang et al., 2011; Thompson, 2012) is an interesting way to ensure process tractability. A recent review shows that many algorithms have been proposed since the first attempt, which was the random sampling (Ros & Guillaume, 2016b). They are based on density or distance, and sometimes combine the two concepts. Among the distance based sampling method, the farthest-first traversal principle is of special interest as it does not require any distance parameter in contrast to the leader (Ling, 1981) or the mountain (Yager & Filev, 1994) methods. The first use of *fft* aimed to solve the traveling salesman problem in 1977 (Rosenkrantz et al., 1977). Then the idea was applied to clustering (Gonzalez, 1985), to minimize the maximum distance from any point to its nearest center, and, later, for image color quantization (Xiang, 1997). To initialize the *k-means* algorithm, *k-means++* (Arthur & Vassilvitskii, 2007) uses a *fft*-like algorithm: a probabilistic component is added to avoid the selection of outliers.

The *fft* algorithm iteratively adds a new item to the sample at each loop, until a stopping criterion is met. The new representative is the farthest from

the already selected ones and the stopping criterion in the original version was the sample size. The concept is summarized in Algorithm 1.

Algorithm 1 Sampling algorithms: representative selection

Input: $T = \{x_i\}$, $i = 1 \ldots, n$
Output: $S = \{y_j\}$, $T(y_j)$, $j = 1, \ldots, s$
Select an initial pattern $x_{init} \in T$
$S = \{y_1 = x_{init}\}$, $T(y_1) = \{y_1\}$, $s = 1$
repeat
  for all $x_l \in T \setminus S$ **do**
    Find $d_{near}(x_l) = \min\limits_{y_k \in S} d(x_l, y_k)$
    $T(y_k) = T(y_k) \cup \{x_l\}$ {Set of patterns represented by y_k}
  end for
  for all $y_k \in S$ **do**
    Find $d_{max}(y_k) = \max\limits_{x_m \in T(y_k)} d(x_m, y_k)$
    Store $d_{max}(y_k)$, $x_{max}(y_k)$ {where $d_{max}(y_k) = d(x_{max}(y_k), y_k)$},
  end for
  {DENDIS: Select the farthest pattern from the representative of the most populated group}
  Sort $y_{(1)}, \ldots, y_{(s)}$ with $|T_{y_{(1)}}| \geq \ldots \geq |T_{y_{(s)}}|$, $x_w = x_{max}(y_{(1)})$
  {DIDES: Select the farthest pattern from the set of representatives}
  $y_w = \arg\max\limits_{y_k \in S} d_{max}(y_k)$, $MaxDmax = d_{max}(y_w)$, $x_w = x_{max}(y_w)$
  $S = S \cup \{x_w\}$
until Stopping condition is met
return S, $T(y_j)$, $j = 1, \ldots, s$

Let $T = \{x_i\}$ be the input set of n multidimensional data, and $S = \{y_j\}$ the size-s sample to be built, $S \subset T$. The set of patterns represented by y_k is: $T(y_k) = \{x_i \mid d(x_i, y_k) = \min\limits_j d(x_i, y_j)\}$.

The first pattern can be randomly chosen or it can be computed as the farthest, depending on the selected distance, from the minimum value in each input space dimension. After the initialization phase, the set S only counts this

initial pattern, x_{init} (lines 3-4).

The main loop (lines 5-19) includes two steps. First, each unselected pattern, $x \in T \setminus S$, is attached to the closest selected one in S (lines 6-9). At the first step, $T(y_1) = T \setminus \{x_{init}\}$. Then, for each set $T(y_k)$, the algorithm searches for the farthest attached pattern. The maximum distance in the group represented by y_k is noted $d_{max}(y_k)$ (lines 10-13).

The next selected representative, x_w, is the farthest item chosen in a given group. In the pioneering version it was chosen as the farthest of all the already selected items. So, boundary patterns are first chosen instead of inner ones. This way, the selected set spans the whole input space.

Recently, two *fft* improvements that combine distance and density concepts were proposed. *DIDES* (Ros & Guillaume, 2016b) and *DENDIS* (Ros & Guillaume, 2016a) differ in the group the new sample item is chosen from and, consequently, in the stopping criterion. The two algorithms are driven by a unique and meaningful parameter, called *granularity*. It impacts the S size, in that the lower the *granularity* the higher the number of representatives. It is data independent, and is combined with the whole set cardinality, n, to define a threshold: $th = n\ granularity$.

In *DENDIS*, density is of prime concern while distance is controlled: the representative is chosen in the most populated group. The stopping criterion is also based upon density: the algorithm stops when there are no more groups with a number of attached patterns higher than the threshold and with a maximum distance in the group high enough with respect to the whole distribution.

In *DIDES*, distance is the dominant criterion and the new representative is the farthest item from the already selected ones, as in the pioneering version. It is chosen in the group which corresponds to the maximum of the d_{max}, $MaxDmax$ (lines 16-17). The threshold th is the minimum size, in the initial set, T, for a cluster one wants to have representatives in S. A representative with fewer than th patterns attached is called a poor representative. When the proportion of T whose representative is a poor representative is high enough, the input space is homogeneously covered. Then, the d_{max} evolution curve can

be modeled to define the stopping criterion as a distance threshold. Density is managed in a post-processing step to discard outliers and consider the representation of connected areas.

The main goal of *DIDES* is to ensure space coverage while the one of *DENDIS* is density representation. However, the two algorithms regulate the compromise between these two objectives. This trade-off management requires some specific internal, or hidden, parameters inferred from the data.

3. Relationship to coresets

The challenge of big data has aroused a new interest in sampling techniques. But the idea is not new: vector quantization was introduced in the field of signal and image processing to summarize a data distribution by a finite number of vectors (Linde et al., 1980). More recently, the coreset framework introduced the idea of approximation quantification: a subset is called a coreset of a whole set if solving the optimization problem on the subset gives an ε-approximate solution on the whole input set. This section aims to investigate whether there is a relationship between *fft* algorithms and coresets.

This concept was initially analyzed by Agarwal, Har-Peled and Varadarajan (Agarwal et al., 2004) for the geometric approximation of point sets. Given a monotone measure function, μ, i.e. for $S \subseteq T$, $\mu(S) \leq \mu(T)$, and given $\varepsilon > 0$, $S \subseteq T$ is an ε-coreset for T with respect to μ, if $(1 - \varepsilon)\mu(T) \leq \mu(S)$. Typical measures include statistics about the set itself such as diameter, width or the geometric shape enclosing T, e.g. the smallest enclosing ball characteristics such as radius or volume.

They proved that this approximation can be obtained using a sample whose size is independent of the number of points and only dependent on ε.

This concept has been extended to clustering applications (Har-Peled & Mazumdar, 2004). The authors proposed the following definition.

Definition. A set S, of s items, is an (k, ε)-coreset for a set T, of $n > s$ items if:

$$(1-\varepsilon)Cost_T(C) \leq Cost_S(C) \leq (1+\varepsilon)Cost_T(C) \tag{1}$$

where $C = \{c_1, \ldots, c_k\}$ is a set of k centers.

Let $c_i^* \in C$ be the closest center for a given $x_i \in T$: $d(x_i, c_i^*) = \min\limits_{m \in 1,\ldots,k} d(x_i, c_m)$.

Similarly, let $c_j^{*'} \in C$ be the closest center for a given $y_j \in S$: $d(y_j, c_j^{*'}) = \min\limits_{m \in 1,\ldots,k} d(y_j, c_m)$.

With the *k-means* algorithm, the two costs are:

- $Cost_T(C) = \sum\limits_{i=1}^{n} d(x_i, c_i^*)$
- $Cost_S(C) = \sum\limits_{j=1}^{s} w_j \; d(y_j, c_j^{*'})$, with $w_j = |T(y_j)|$, i.e. the number of items from T whose closest point in S is y_j.

When this definition only holds for the optimal number of centers, k, S is called a weak coreset for T, otherwise, if it holds for any set C, it is called a strong coreset for T.

Theorem. *fft* algorithms yield a (k, ε)-coreset.

The proof is as follows:

As y_j also belongs to T, let $d(y_j, c_i^*)$ be the distance between the representative and its closest center computed from the whole set T.

One obtains $\forall j \in S$: $d(y_j, c_i^*) > d(y_j, c_j^{*'})$ if $c_i^* \neq c_j^{*'}$ otherwise $d(y_j, c_i^*) = d(y_j, c_j^{*'})$ then:

$$\sum_{j=1}^{s} w_j \; d(y_j, c_j^{*'}) \leq \sum_{j=1}^{s} w_j \; d(y_j, c_i^*) \tag{2}$$

The triangle inequality yields: $d(y_j, c_i^*) \leq d(x, y_j) + d(x, c_i^*)$

The maximum within group distance, $d_{max}(y_j) = d_j$ for group j, provides an easy to estimate upper bound of any elementary distance to the center. For

each group $j \in S$, the inequality can be applied to the set of all the items in the group:

$$w_j \ d(y_j, c_i^*) \leq w_j \ d_j + \sum_{l=1}^{w_j} \ d(x_l, c_i^*) \tag{3}$$

Considering the whole set of the representatives the index of the summation covers the n items and combining Eq (2) and Eq. (3) yields: $\sum_{j=1}^{s} w_j \ d(y_j, c_j^{*'}) \leq \sum_{j=1}^{s} w_j \ d_j + \sum_{i=1}^{n} d(x_i, c_i^*)$. Finally:

$$Cost_S(C) \leq \sum_{j=1}^{s} w_j \ d_j + Cost_T(C) \tag{4}$$

The quantity $\sum_{j=1}^{s} w_j \ d_j$ is the sampling cost, or the quantization distortion.

The triangle inequality also gives: $d(x, c_j^{*'}) \leq d(x, y_j) + d(y_j, c_j^{*'})$

then $d(y_j, c_j^{*'}) \geq d(x, c_j^{*'}) - d(x, y_j)$

A similar reasoning, which comes to apply the latter first to a group using d_j as an upper estimate of the distance, second to all the groups, and finally stating that $\forall x, \ d(x, c_j^{*'} \geq d(x, c_i^*)$, gives the following inequality:

$\sum_{j=1}^{s} w_j \ d(y_j, c_j^{*'}) \geq \sum_{i=1}^{n} d(x_i, c_i^*) - \sum_{j=1}^{s} w_j \ d_j$

Meaning:

$$Cost_S(C) \geq Cost_T(C) - \sum_{j=1}^{s} w_j \ d_j$$

The final relation between the two costs is:

$$Cost_T(C) - \sum_{j=1}^{s} w_j \ d_j \leq Cost_S(C) \leq Cost_T(C) + \sum_{j=1}^{s} w_j \ d_j \tag{5}$$

Dividing Eq. 5 by $Cost_T(C)$, and then multiplying by $Cost_T(C)$, yields:

$$\left(1 - \frac{\sum_{j=1}^{s} w_j \ d_j}{Cost_T(C)}\right) Cost_T(C) \leq Cost_S(C) \leq \left(1 + \frac{\sum_{j=1}^{s} w_j \ d_j}{Cost_T(C)}\right) Cost_T(C)$$

which is Eq. 1 with $\varepsilon = \frac{\sum_{j=1}^{s} w_j \ d_j}{Cost_T(C)}$.

S is thus a (k, ε)-coreset for T, with ε equals the ratio of the sampling cost to the whole cost. As there is no assumption about C, S is a strong coreset for T.

As mentioned above *DIDES* and *DENDIS* are driven by a unique parameter called granularity. The sampling cost and, consequently, ε has a monotonic evolution with respect to this parameter. A lower granularity tends to yield a higher sample size and adding a new item to S decreases both w and d.

4. ProTraS: the proposed algorithm

The *fft*-based algorithms yield coresets even if they were not designed with this goal in mind. The objective of the present proposal is to generate coresets.

4.1. Description of the algorithm

The approximation level, which is the sampling cost as shown in the previous section, plays a central role in coreset generation. ε, is the unique parameter. It also serves as the stopping criterion: the algorithm ends when the cost is below the threshold. Finally it is used to guide the sampling process itself: at each iteration a new representative is added to the sample, in the group with the highest probability of cost reduction. This tends to limit the sample size.

The algorithm is summarized in Figure 1.

1. Add a new sample in the group with the highest probability of cost reduction
2. Assign each pattern to the nearest sample
3. Compute Cost
4. If (Cost > CostParam) goto Step 1

Figure 1: ProTraS: summary of the algorithm

The representative is the farthest-first traversal item of a given group.

As the sampling cost in a given group, j, is proportional to both the number of attached patterns, $w_j = |T_{y_j}|$, and the maximum within group distance,

$d_j = d_{max}(y_j)$, the new representative is chosen in the group that is likely to best contribute to the global cost, ε, reduction. The probability of cost reduction is a combination of two basic probabilities: the first one according to the distance and the other one according to the number of attached patterns. Each of them is estimated by normalizing the values by the maximum over all the groups.

For a given group, j, the probabilities are:

- Density-based probability: $P_{dens}(j) = \frac{w_j}{\max\limits_i \ w_i}$
- Distance-based probability: $P_{dist}(j) = \frac{d_j}{\max\limits_i \ d_i}$

Both have the same meaning: the higher the value, the higher the expected cost reduction. The probability of cost reduction is computed as the following combination: $P_{CostRed}(j) = \dfrac{w_j d_j}{\max\limits_i \ (w_i d_i)}$.

This probabilistic approach differs from greedy algorithms. For instance in Decision Trees, the node to be split is the one with the highest gain. This requires all the nodes to be tested, i.e. all the gains have to be computed. The probability estimation tends to speed up the sampling process.

The algorithm is detailed in Algorithm 2.

The initial pattern selection (line 3) is no longer done at random. The first representative is the closest to a virtual item computed as the minimum in each dimension. This makes the algorithm fully deterministic.

The main loop, lines 5-23, includes two *for* loops. The first one is described in Algorithm 1. The second loop is enriched compared to the previous version. It also computes the combined probability according to $|T_{y_k}|$ and $d_{max}(y_k)$ (line 14), identifies the group with the highest cost reduction probability (line 16) and updates the global cost before splitting (line 18). The new representative is the farthest-first traversal, x_*, of the group with the highest probability, represented by y_*. To update the current cost, the cost due to former y_* is subtracted and the new costs, induced by y_* and x_*, are added. The cost is computed at each iteration without any additional computational effort.

Algorithm 2 The ProTraS algorithm

Input: $T = \{x_i\}$, $i = 1 \ldots, n$, ε

Output: $S = \{y_j\}$, $T(y_j)$, $j = 1, \ldots, s$

Select an initial pattern $x_{init} \in T$

$S = \{y_1 = x_{init}\}$, $s = 1$, $T(y_1) = \{y_1\}$

repeat

  for all $x_l \in T \setminus S$ **do**

    Find $d_{near}(x_l) = \min\limits_{y_k \in S} d(x_l, y_k)$

    $T(y_k) = T(y_k) \cup \{x_l\}$ {Set of patterns represented by y_k}

  end for

  $MaxWD = 0$, $Cost = 0$

  for all $y_k \in S$ **do**

    Find $d_{max}(y_k) = \max\limits_{x_m \in T(y_k)} d(x_m, y_k)$

    Store $d_{max}(y_k)$, $x_{max}(y_k)$ {where $d_{max}(y_k) = d(x_{max}(y_k), y_k)$}

    $p_k = |T(y_k)|\ d_{max}(y_k)$

    if $(p_k > MaxWD)$ **then**

      $MaxWD = p_k,\ y_* = y_k$

    end if

    $Cost := Cost + p_k/n$

  end for

  $x_* = x_{max}(y_*)$

  $S = S \cup \{x_*\}$, $s = s + 1$, $T(y_s) = \{x_*\}$

  Update Cost

    {Remove the part due to former y_* and add the two new costs, induced by y_* and x_*}

until $(Cost < \varepsilon)$

return $Cost$, S, $T(y_j)$, $j = 1, \ldots, s$

The algorithm uses, in line 18, a normalized distortion cost: $\frac{\sum_{j=1}^{s} w_j \ d_j}{n}$, meaning that the approximation, ε, is now a proportion of the whole normalized cost, $\frac{Cost_T(C)}{n}$. This new expression carries the same meaning for all the data, irrespective of the data size.

A better cost estimation could be reached by choosing in each group the medoid as the representative.

4.2. Algorithm properties

Some interesting properties of the algorithm are illustrated using the $S1$ data shown in Figure 2. This synthetic data set includes 3000 $2D$-points and was used in (Kärkkäinen & Fränti, 2002) under the name *A.set 1.*

4.2.1. Robust to noise

An increasing amount of uniform random noise (from 1% to 19%) was added to the data. The new values were computed independently in each dimension. according to the whole range of the given feature: $noise_f = \min_f + U[0. + 1] * (\max_f - \min_f)$.

To assess the quality of representation, the resulting partitions of the same clustering algorithm applied to the whole set and the selected sample were compared using the Rand Index(Rand, 1971).

As the optimum number of clusters is unknown, tests were carried out within a range $[2, 17]$. The *k-means* algorithm was run 10 times for each configuration. Table 1 reports the sample size and the RI averaged over all the trials.

Noise clearly impacts the sampling: the sample size tends to increase with the amount of noise, even if these results also include an initialization variation. RI values show that the sampling algorithm is still able to identify and represent the data structure even with a significant level of noise.

4.2.2. Sample size mainly dependent on data structure

The test consists in comparing the algorithm performances on the initial set and an enriched similar one. The latter results from the aggregation of new pat-

Table 1: Noise sensitivity: sample size and RI, on $S1$ data, $cost = 0.1$

% noise	Size	s	RI
0	3000	261	0.975
1	3030	309	0.979
2	3060	323	0.978
3	3090	342	0.990
4	3120	348	0.984
5	3150	362	0.982
6	3180	356	0.977
7	3210	364	0.978
8	3240	369	0.978
9	3270	368	0.985
10	3300	378	0.983
11	3330	374	0.984
12	3360	370	0.975
13	3390	384	0.988
14	3420	381	0.977
15	3450	380	0.976
16	3480	392	0.985
17	3510	388	0.983
18	3540	388	0.978
19	3570	393	0.979
μ		361	0.980
σ		32	0.004

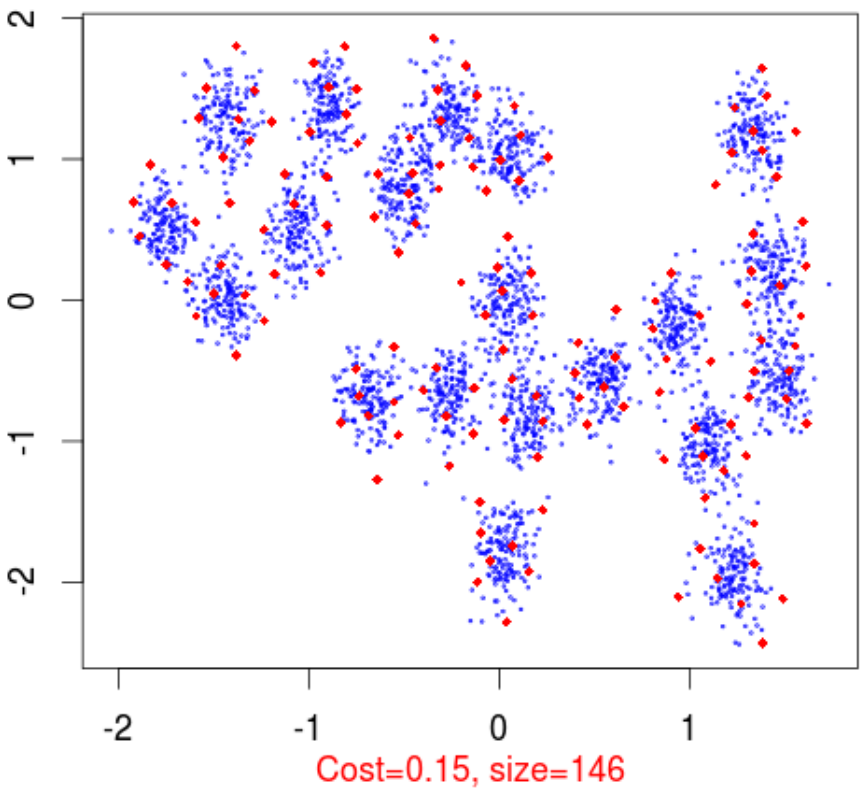

Figure 2: The $S1$ data

terns: each data point was replicated k times, $k = 2, \ldots 20$, with an additional uniform random noise in the range $[-0.1, +0.1]\sigma_f$ where σ_f is the standard deviation for feature f. It is worth mentioning that the noise generation is distinct from the one used in the previous section. The expected magnitude is lower as the goal is just to generate non identical items. The sampling is applied with the same cost. In Table 2 the sample size and the RI are given for the initial set, in the first row, and the enriched ones.

The sample size is significantly increased at the first step of the procedure due to the additional noise: the data structure is significantly modified as the noise

Table 2: Data size sensitivity: sample size and RI, on $S1$ data, $cost = 0.1$

Size	s	RI
3000	261	0.976
6000	309	0.973
9000	323	0.980
12000	342	0.977
15000	348	0.983
18000	354	0.970
21000	363	0.980
24000	357	0.981
27000	366	0.979
30000	369	0.980
33000	368	0.983
36000	377	0.984
39000	374	0.982
42000	381	0.983
45000	376	0.983
48000	382	0.985
51000	381	0.977
54000	380	0.976
57000	392	0.985
60000	388	0.986
μ		0.980
σ		0.004

is randomly defined for each input dimension. Then it tends to become more stable. The sample size ranges from 261 to 392 while the whole size becomes 20 times bigger. This does not significantly impact the results. This experiment shows that the outcome of *ProTraS* depends more on the data structure than on the data size.

4.2.3. Time optimized

At a given iteration only a small part of the input space is modified. It is possible to avoid a lot of distance computations using the triangle inequality and a limited amount of memory space.

When a new representative in S has been selected, y_*, the question is: should a given initial pattern, x, be attached to y_* instead of remaining in $T(y_j)$? In the case illustrated in Figure 3, three groups are defined and the new representative belongs to the first one. The triangular inequality states: $d(y_j, y_*) \leq d(x, y_j) + d(x, y_*)$ and x would be attached to y_* if $d(x, y_*) < d(x, y_j)$. So, if $d(y_j, y_*) \geq 2\ d(x, y_j)$, x remains in $T(y_j)$. Storing the maximum within group distance, $d_{max}(y_j)$, makes the test useless for all the items in the group if the inequality holds for the farthest from y_j. That means that if $d(y_j, y_*) \geq 2\ d_{max}(y_j)$ no change has to be made. This is the case for *Group 2* in the upper part of the plot, the farthest item from y_2 is x_2.

The number of patterns managed by this group level optimization increases with the sample size, as the averaged induced volume decreases. When the whole group is not managed by the previous test, only one alternative group must be considered, the closest from the one with the new representative, located at the distance $d_{1nn}(y)$. The same triangle inequality applied at the pattern level provides a useful threshold. All $x \in T(y_j)$ with $d_{near}(x) \leq 0.5\ d(y_j, y_*)$ remain attached to $T(y_j)$. $d_{near}(x)$ is the distance from x to the closest element in S, its representative, see Algorithm 1. In the figure, $y_* = y_4$ is the farthest item in *Group 1*, x_1, and the closest representative from y_1 is y_3. The limits of the new group, *Group 4*, are plotted in dashed lines. *Group 3* is reduced and x_i becomes the farthest element from y_3, and it is now labeled as x_3. Similarly, x_j

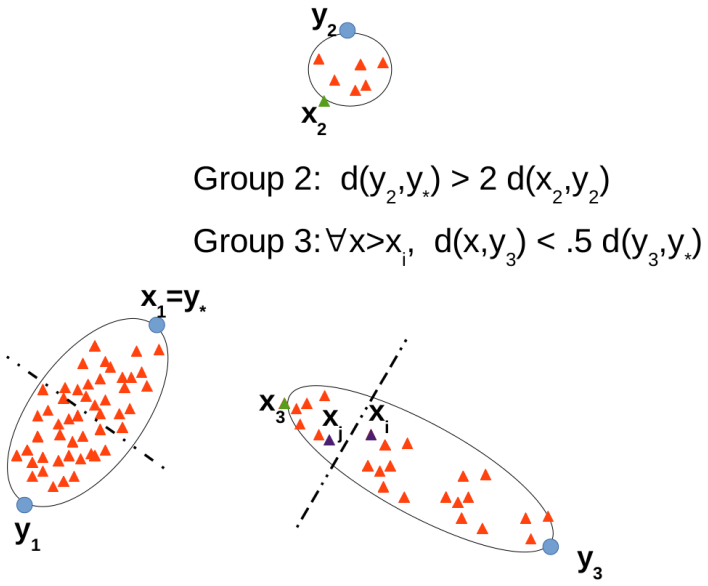

Figure 3: Optimization illustration

becomes x_4.

The spatial complexity for this time optimization can be considered as reasonable: the final version stores $n + 2 * s$ distances. n $d_{near}(x)$, s $d_{max}(y)$ and $d_{1nn}(y)$ as well as the corresponding elements, y for $d_{near}(x)$, and x for $d_{max}(y)$.

5. Numerical experiments

Twenty-nine databases were used, 21 are synthetic, $S1$ to $S21$, and 8 are real world data sets, $R1$ to $R8$.

The selected data are from the data clustering repository of the computing school of Eastern Finland University[2], *Finland* hereafter, the UCI machine learning repository[3], *UCI*, the github clustering benchmark [4], *clustering*, or were proposed in the published literature.

[2]https://cs.joensuu.fi/sipu/datasets/
[3]https://archive.ics.uci.edu/ml/
[4]https://github.com/deric/clustering-benchmark

Data are of various dimensions (from 2 to 10), sizes (from tiny to very large), shapes and densities. Their main characteristics and origin are summarized in Table 3.

The data were standardized to have a zero mean and a unit variance. Some synthetic data sets are plotted in Figure 4.

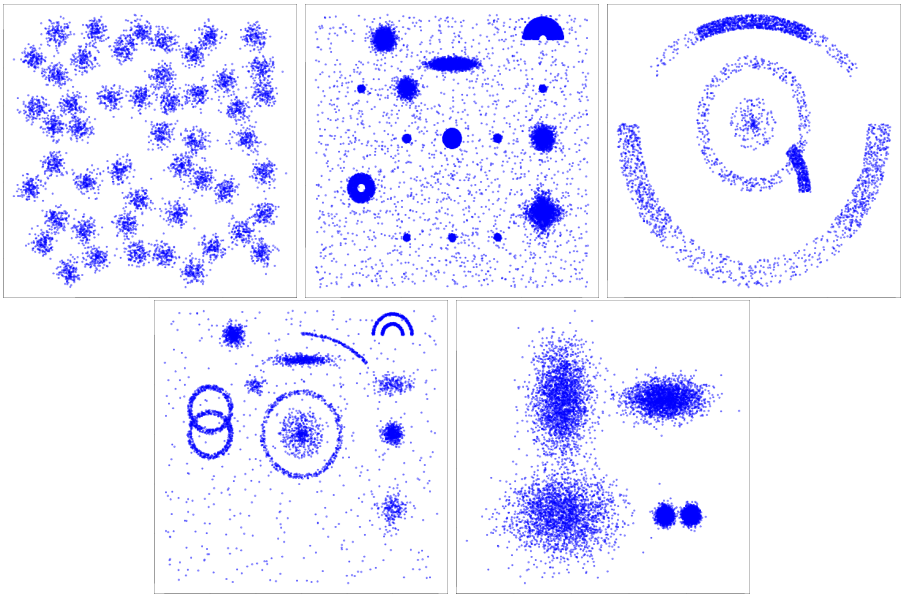

Figure 4: Data sets S3, S17, S18, S20 and S21

5.1. Sample representativeness

Two different clustering techniques were tested to assess the sample representativeness: *k-means* and *DBSCAN* (Zhou et al., 2000). The former is distance-based, the latter is density-based. Various improvements have been recently proposed (Tran et al., 2013) but they are beyond the scope of this paper. The native versions were used in the present study.

[5]The original data set, with 240 items, was enriched up to 12500 items. Each duplicated point is added a random Gaussian noise in each dimension, f,: $x'_f = x_f + \mathcal{N}(x_f, 0.1\sigma_f)$.

Table 3: The synthetic and real world databases

	Size	Dim	Name	Origin
S1	3000	2	A.set 1	(Kärkkäinen & Fränti, 2002)
S2	5250	2	A.set 2	(Kärkkäinen & Fränti, 2002)
S3	7500	2	A.set 3	(Kärkkäinen & Fränti, 2002)
S4	12000[5]	2	FLAME	(Fu & Medico, 2007)
S5	100000	2	Birch-set 3	(Zhang et al., 1997)
S6	373	2	JAIN	(Jain & Law, 2005)
S7	5000	2	S.sets 1	(Fränti & Virmajoki, 2006)
S8	5000	2	S.sets 2	(Fränti & Virmajoki, 2006)
S9	5000	2	S.sets 3	(Fränti & Virmajoki, 2006)
S10	5000	2	S.sets 4	(Fränti & Virmajoki, 2006)
S11	1351	2	Dim sets 1	*Finland*
S12	2701	4	Dim sets 2	*Finland*
S13	4051	6	Dim sets 3	*Finland*
S14	5401	8	Dim sets 4	*Finland*
S15	6751	10	Dim sets 5	*Finland*
S16	8000	2	cluto-t8-8k	*clustering*
S17	40000	2	Homemade	Fig. 4
S18	3800	2	Homemade	Fig. 4
S19	8000	2	t4.8k	(Karypis et al., 1999)
S20	5500	2	Homemade	Fig. 4
S21	12500	2	Homemade	Fig. 4
R1	68040	9	Color moments	*Finland*
R2	169308	3	Differential coordinates	(Fränti & Virmajoki, 2006)
R3	13467	2	User Location (Finland)	(Fränti & Virmajoki, 2006)
R4	950	10	Stock	(Alcalá et al., 2010)
R5	857357	3	Transactions90k	(Alcalá et al., 2010)
R6	1837	3	House5	(Fränti & Virmajoki, 2006)
R7	34112	3	House8	(Fränti & Virmajoki, 2006)
R8	45781	3	Tamildanu	*UCI*

As the optimum number of clusters is unknown, tests were carried out within a range $[2, 17]$. Finding one configuration with a given number of clusters with *DBSCAN* is quite tricky, unlike with *k-means*, for which this number is an input parameter. Using the *k-means*, when the number of clusters gets high, some of them may be very small, or even empty, for a given data set. In this case the range is restricted to the representative clusters.

DBSCAN is run with two parameters: the distance, d, that defines a spherical neighborhood, and the minimum number of points, $minp$, required to consider the data point as a core point, otherwise it is labeled as an outlier. A specific process was designed to set these two parameters in order to obtain a desirable number of clusters with a reasonable number of outliers, less than 10%.

First, *DBSCAN* was run on the sample with $minp = max(1, 0.01n)$ to identify some distances that fit the above mentioned conditions. Let d_{init} be the distance that gives 2 clusters. Then the range $[0, d_{init}]$ is equally divided into 1000. *DBSCAN* was run with all these distance parameters on the samples and on the whole sets. All the configurations that gave an acceptable number of clusters and with the proportion of outliers less than the threshold were kept. The partitions were compared for all the pairs, one element with the sample and the other with the whole set, for which the difference in the number of clusters was at most 2.

As *DBSCAN* is computationally expensive, data size was limited to 5000 to complete the test within a reasonable time. The extra items were randomly removed.

As *k-means* is sensitive to initialization, a given number of trials, 10 in this paper, were run for each number of clusters.

The resulting sample size as well as the computational cost were carefully studied as they have a strong impact on the practical use of the algorithm. The CPU cost is characterized by a time ratio. It is computed as the sum of the sampling time and the time for clustering the sample divided by the time required to cluster the whole data.

As in the previous section, the RI is used to assess the quality of representation. The value reported in the following tables is the average of all experiments, number of runs and number of clusters.

The same cost, 0.1, was used for the sampling. As it is normalized by the data size, it carries the same meaning for all the data sets.

The results are summarized in Table 4: t is the whole set size, s the sample one, s/t is thus the sampling ratio, and the RI and time ratio are reported for both algorithms.

It is interesting to see that the sampling ratio ranges from $0.\%$ ($R3$) to 73% ($R4$). Similar values are obtained with the synthetic data: 0.4% for $S5$ and 70% for $S4$. This highlights that there is no relationship between the cost, which is 0.1 for all the data sets, and the sample size or the sampling ratio. This is a strong asset of *ProTraS*: the sample size depends only on the data structure. The sample size is limited to 1999 by the program. When this limit is reached, the real cost may be higher than the program parameter. This occurred for $R1$, $R5$ and $R8$. Even if the cost is not reached, the representativeness of the sample is satisfactory.

For a large number of clusters, i.e. when k tends to n, the relationship between the cost and the RI is clear. With the range studied here, however, it is not so clear. Even if the RI are usually high, some comments are in order. With the $R2$ data, this value is below 0.9 for the *k-means*. In these cases *DBSCAN* yields a better index. A detailed analysis of $R2$ experiments shows that some configurations are not stable for the *k-means*, given this cost (0.1) and the corresponding sample rate (about 1%). With $k = 2$, the mean RI is 0.76, and it is 0.77 with $k = 5$.

The last two columns of Table 4 report the time ratio given in percent. It is, as expected, clearly lower for *DBSCAN* than *k-means*. For the latter, sampling becomes interesting when dealing with large databases or well structured data. In this case, the sample size is much smaller than the whole set.

Table 4: *ProTraS* with $cost = 0.1$ and two clustering algorithms

				RI		Time ratio (%)	
	t	s	s/t	k-means	DBSCAN	k-means	DBSCAN
S1	3000	261	0.089	0.978	0.960	122.8	1.8
S2	5250	315	0.060	0.978	0.960	52.5	0.95
S3	7500	341	0.046	0.972	0.975	45.1	0.6
S4	12000	133	0.696	0.952	1.000	22.5	0.29
S5	100000	259	0.004	0.979	1.000	25.7	0.03
S6	373	108	0.308	0.958	0.980	178.9	18.0
S7	5000	237	0.048	0.985	1.000	52.7	0.8
S8	5000	327	0.066	0.981	1.000	50.0	1.1
S9	5000	422	0.084	0.981	1.000	54.8	1.3
S10	5000	448	0.084	0.969	1.000	56.7	1.3
S11	1351	17	0.012	0.978	1.000	71.4	0.6
S12	2701	17	0.007	0.997	0.980	30.6	0.6
S13	4051	20	0.106	0.995	1.000	53.8	3.4
S14	5401	416	0.277	0.994	0.990	152.2	8.8
S15	6751	379	0.296	0.995	1.000	208.6	9.9
S16	8000	303	0.061	0.993	0.980	42.3	0.08
S17	40000	475	0.006	0.976	0.940	22.3	1.0
S18	3800	236	0.044	0.979	1.000	38.8	0.7
S19	8000	475	0.060	0.971	0.980	36.8	0.1
S20	5500	253	0.060	0.971	0.980	36.8	0.1
S21	12500	238	0.060	0.975	0.980	36.8	0.1
R1	68040	1999	0.029	0.933	1.000	98.2	0.02
R2	169308	1008	0.011	0.884	1.000	39.1	0.03
R3	13467	103	0.008	0.934	1.000	27.1	0.2
R4	950	695	0.733	0.999	1.000	210.2	60.9
R5	857357	1999	0.020	0.991	0.985	70.2	0.004
R6	1837	1252	0.682	0.977	1.000	312.1	51.6
R7	34112	1672	0.049	0.982	0.970	95.5	0.3
R8	45781	1999	0.027	0.961	0.950	93.1	0.05

5.2. Cost sensitivity

The cost plays a central role in the *ProTraS* algorithm. Not only is it the input parameter, it is also used to select the sample items and serves as a stopping criterion. To assess the algorithm sensitivity to the cost, 10 cost values, from 0.08 to 0.26 by steps of 0.02, were used with all the data sets. For each data set and a given cost the *k-means* was run 10 times with a number of clusters from 2 to 17 on the sample and the whole set. The RI was averaged over all these trials.

The results are summarized in Table 5. The first two columns report the maximum and minimum sample size over all the costs, the following two ones show the same information for the RI, and finally the differences are computed for both indicators.

The maximum size was limited to 1999 in 5 cases. In one of them, $R1$, it is also the minimum size, meaning that none of the cost approximations could be reached with such a limited size. For the $R1$ data, the differences in the RI are due only to the *k-means* initialization.

For 3 data sets, the difference in the sample size is higher than 1500: $S14$, $R7$ and $R8$. The corresponding variation for the RI is not that high.

The expected trend is that the smaller the cost, the better the representation will be, but the sample size and the RI are also determined by the data structure. Two real data sets, $R2$ and $R3$ have a minimum RI less than 0.9.

These experiments show that for a large range of distortion costs, the sample is able to well represent the whole according to the RI even with a reduced size. This highlights the quality of the representative selection using the *ProTraS* algorithm.

5.3. Comparison with alternative approaches

This section focuses on the comparison between *ProTraS*, *DENDIS*, *DIDES* and two of the twelve methods already used to assess the relevance of *DENDIS* or *DIDES*. The two of them proved competitive regarding their computational

Table 5: Cost sensitivity, with RI for the *k-means*

	s_{MaxC}	s_{MinC}	RI_{MaxC}	RI_{MinC}	Δs	ΔRI
$S1$	353	66	0.988	0.934	287	0.054
$S2$	429	76	0.982	0.941	353	0.040
$S3$	472	74	0.979	0.939	398	0.040
$S4$	403	87	0.976	0.943	316	0.033
$S5$	360	70	0.984	0.939	290	0.044
$S6$	140	37	0.962	0.951	103	0.011
$S7$	332	65	0.981	0.960	267	0.022
$S8$	438	82	0.992	0.951	356	0.040
$S9$	582	106	0.988	0.945	476	0.043
$S10$	582	106	0.980	0.953	476	0.027
$S11$	25	10	0.974	0.968	15	0.006
$S12$	41	10	0.996	0.969	31	0.027
$S13$	721	21	0.995	0.960	700	0.035
$S14$	1999	69	0.996	0.965	1930	0.030
$S15$	472	164	0.998	0.987	308	0.011
$S16$	666	102	0.976	0.935	564	0.041
$S18$	302	92	0.991	0.974	210	0.017
$S19$	298	53	0.972	0.934	245	0.038
$S20$	335	80	0.990	0.952	255	0.035
$S21$	239	47	0.982	0.945	192	0.037
$R1$	1999	1999	0.937	0.929	0	0.008
$R2$	1446	214	0.930	0.889	1232	0.041
$R3$	130	43	0.928	0.912	87	0.016
$R4$	736	412	0.997	0.980	324	0.017
$R5$	1999	533	0.987	0.968	1466	0.019
$R6$	1369	260	0.981	0.942	1109	0.040
$R7$	1999	237	0.983	0.953	1762	0.031
$R8$	1999	416	0.984	0.962	1583	0.022

cost and quality of representation. To be fair, the comparison is carried out with a same sample size, the one given by the *ProTraS* algorithm.

The first one, denoted *AdaCor*, described in (Feldman et al., 2011), proposes an *Efficient Coreset Construction via Adaptive Sampling*. The key idea is to build an approximate solution (sample set, S) and to use it to bias the random sampling. The first step is achieved by an iterative algorithm that samples a small number of points, β, and removes half of the data set (T) closest to the sampled points. In the second step the sampling is biased with probabilities, for each point in T, which are roughly proportional to their squared distance to S.

The second approach, denoted *AdaGri*, combines the pioneering idea from Kollios et al. (2003) to bias the sampling process with an adaptive grid partitioning (Lin et al., 1997; Pintore et al., 2002). At each step the densest bin (subspace) is split. The final subspace set is used to estimate the local densities $\hat{f}(x)$.

Once the local densities have been estimated, they are used to bias the sampling process as follows:

$$\frac{s}{\sum \hat{f}(x)} \hat{f}(x)^{\alpha}$$

where s is the desired sample size and α the bias parameter. If $\alpha = 0$, the process reduces to a random sampling $\left(\frac{s}{n}\right)$. Otherwise, if $\alpha > 0$ (respectively $\alpha < 0$) high density regions are sampled at a higher (lower) rate.

This algorithm is sensitive to its parameters: the bias, α, the number of cuts tested per axis, N_{cut}, the maximum number of bins Max_b and the minimum number of points in a bin, Occ_m.

A given cost from *ProTraS* yields a sample size which is used as an input for the other algorithms. The relevant input parameters for *AdaCor* and *AdaGri* are similar to the ones used by the authors in the referenced papers: $\beta = 100, \alpha = -0.15$, $N_{cut} = 2$, $Max_b = 200$, $Occ_m = 5$. *DENDIS* and *DIDES* are run with the granularity that yields a similar number of samples to *ProTraS*. To avoid a random effect, these two algorithms are initialized, as *ProTraS*, by the *fft* from a virtual extreme item. Only *AdaCor* is thus sensitive to random,

the three others being deterministic. 10 samples are generated from *AdaCor* for each configuration. For each data set, the *k-means* was run 10 times with a number of clusters from 2 to 17, both on each of the samples and the whole set. The results were averaged over all these trials. The comparison is made, for the same sample size, on the running time and the quality of representation. As the running time for *DENDIS* and *DIDES* is similar to the one of *ProTraS*, it is not analyzed.

Two contrasted cases were studied, corresponding to two costs: a high approximation level, 0.1, and a low one, 0.2. The results are given in Tables 6 and 7.

The high approximation level yielded quite good results with *ProTraS*: the mean *RI* for the synthetic data is 0.979, it is higher than 0.95 in all cases and above 0.99 for 7 sets. For any alternative approach the result shown in the tables is computed as the *ProTraS* value minus the alternative corresponding one. *DENDIS* was better than *ProTraS* for $S6$ and $S21$ when an absolute value of 0.005 for the *RI* was used as a threshold of significance. In average, *DENDIS* and *AdaCor* performed better than *DIDES* and *AdaGri* with the synthetic data. *AdaCor* proved slower than the others. For the eight real world data sets, the average quality was decreased to 0.961 for *ProTraS* and, in two cases, *AdaCor* and *AdaGri* did better than the proposal. The standard deviation of the *RI* are quite similar, ranging between 0.02 in average for *ProTraS* and *DIDES* to 0.04 for the others.

The result for the low approximation level are given in Table 6. The samples were, as expected, significantly smaller and the running time for *ProTraS* was half of the other one in average. The quality of representation is also decreased: only two *RI* for the synthetic data were higher than 0.99 and two of them are below 0.95. The difference in the averaged *RI* was increased for all the competitors except *DIDES*, which yielded a similar result to the previous experiment. With the real data, the result of the low level approximation is confirmed: *AdaCor* gave a similar quality of representation to *ProTraS*, but it is much slower. The standard deviation of the *RI* was also decreased by a factor 2 with respect

Table 6: Alternative approaches: comparison for high level approximation

	ProTraS			*DENDIS*	*DIDES*	*AdaCor*		*AdaGri*	
	s	t (ms)	*RI*	ΔRI	ΔRI	Δt	ΔRI	Δt	ΔRI
S1	261	25	0.978	0.008	0.017	-17	0.018	7	0.032
S2	315	31	0.978	0.006	0.017	-68	0.021	2	0.042
S3	341	36	0.972	0.017	0.014	-122	0.014	1	0.035
S4	133	46	0.952	0.005	0.019	-215	0.032	0	0.034
S5	259	207	0.979	0.123	0.130	-2392	0.012	-152	0.050
S6	108	11	0.959	-0.005	0.054	8	0.001	10	0.013
S7	237	19	0.986	0.005	0.038	-62	0.011	-5	0.024
S8	327	24	0.981	0.017	0.048	-65	0.024	-2	0.031
S9	422	26	0.981	0.007	0.044	-68	0.014	-1	0.025
S10	448	36	0.969	-0.001	0.016	-59	0.018	9	0.055
S11	17	26	0.979	0.005	0.047	-68	0.013	-1	0.027
S12	17	6	0.997	0.034	0.029	-6	0.070	-1	0.088
S13	20	9	0.995	0.038	0.012	-26	0.020	-16	0.032
S14	476	46	0.994	0.012	-0.002	-70	0.000	-26	0.012
S15	379	126	0.995	-0.001	0.005	-143	-0.001	36	0.005
S16	303	221	0.993	-0.004	0.020	-265	-0.002	-12	-0.004
S17	475	38	0.977	0.006	0.046	-147	0.009	-9	0.045
S18	236	98	0.979	0.055	0.119	-1682	0.043	-57	0.057
S19	475	71	0.971	0.008	0.043	-116	0.005	23	0.023
S20	253	27	0.971	0.022	0.097	-68	0.010	-1	0.032
S21	238	39	0.975	-0.008	0.090	-252	0.012	-17	-0.002
mean		55	0.979	0.017	0.043	-281	0.016	-10	0.031
R1	1999	1204	0.929	0.014	0.019	-5133	-0.010	9	0.020
R2	1008	329	0.925	0.054	0.042	-2824	0.033	13	0.069
R3	103	25	0.934	0.013	0.044	-285	0.039	-29	-0.012
R4	695	20	0.999	0.010	0.021	-6	0.001	-6	0.010
R5	1999	609	0.990	0.030	0.120	-3892	0.010	160	0.039
R6	1252	48	0.958	0.003	0.019	-3	-0.023	34	-0.009
R7	1672	331	0.978	0.051	0.110	-1927	0.008	145	0.000
R8	1999	575	0.976	0.017	0.044	-3358	0.007	164	0.048
mean		393	0.961	0.024	0.052	-2179	0.008	61	0.021

Table 7: Alternative approaches: comparison for low level approximation

	ProTraS			*DENDIS*	*DIDES*	*AdaCor*		*AdaGri*	
	s	t (ms)	*RI*	ΔRI	ΔRI	Δt	ΔRI	Δt	ΔRI
S1	97	19	0.938	0.009	0.002	-19	0.020	1	0.028
S2	116	23	0.956	0.023	0.018	-57	0.036	-4	0.056
S3	119	24	0.952	0.008	0.002	-109	0.033	-10	0.031
S4	261	68	0.963	-0.005	0.009	-215	0.032	22	0.028
S5	104	124	0.959	0.212	0.130	-2387	0.106	-158	0.086
S6	56	6	0.940	0.008	0.045	4	0.011	5	0.030
S7	96	15	0.968	0.020	0.036	-60	0.034	-11	0.040
S8	120	15	0.962	0.032	0.035	-60	0.034	-11	0.033
S9	155	14	0.963	0.015	0.039	-63	0.015	-14	0.033
S10	166	16	0.956	0.039	0.077	-62	0.021	-10	0.058
S11	15	15	0.962	0.007	0.042	-62	0.020	-11	0.028
S12	10	6	0.967	0.090	0.100	-5	0.065	0	0.104
S13	10	6	0.988	0.035	0.022	-29	0.252	-19	0.306
S14	175	25	0.977	0.020	0.009	-60	-0.004	-40	0.036
S15	175	16	0.991	0.038	0.016	-109	0.000	-69	0.007
S16	492	36	0.993	0.002	-0.001	-198	0.006	-186	-0.003
S17	152	20	0.949	0.012	0.027	-129	0.009	-25	0.046
S18	116	69	0.961	0.059	0.110	-1639	0.037	-83	0.031
S19	152	30	0.955	0.011	0.029	-118	-0.001	-15	0.076
S20	118	25	0.965	0.022	0.093	-63	0.013	-4	0.036
S21	83	31	0.967	-0.006	0.098	-240	0.002	-26	0.001
mean		28	0.963	0.031	0.045	-270	0.035	-32	0.052
R1	1999	1198	0.931	0.027	0.157	-4900	0.002	267	0.025
R2	321	179	0.882	0.148	0.130	-2477	0.020	-125	0.104
R3	56	20	0.891	-0.023	0.007	-281	-0.018	-34	0.005
R4	508	13	0.997	0.005	0.022	-9	0.004	-13	0.008
R5	809	294	0.978	0.013	0.109	-2878	0.019	-177	0.042
R6	488	17	0.854	0.087	0.099	-14	0.002	5	0.010
R7	396	116	0.964	0.004	0.097	-1422	0.003	-110	0.029
R8	795	286	0.971	0.042	0.083	-2515	0.007	-162	0.037
mean		265	0.935	0.038	0.088	-1812	0.005	-74	0.033

to the high approximation level.

The similarity of the results yielded by *ProTraS* and *AdaCor* is not surprising: they share the idea of tracking dense areas without forgetting the others. In the *AdaCor* approach, dense areas are first covered by a biased random sampling, then the initial patterns represented by this sample are no longer considered giving more opportunity for less dense areas to be represented. *ProTraS* is clearly faster than *AdaCor* and this is more visible for large data sets ($S5$, $S17$ and most of the real world ones). The algorithm complexities are similar but the number of distance calculations is reduced in *ProTraS* thanks to an internal optimization.

In contrast, *AdaGri* may be faster than *ProTraS* but is outperformed by the latter. Although adaptive, the grid bins are likely to mix outliers or noisy data with more representative ones. Yet, the same bias is applied to the whole cell.

DIDES showed the worst performance for the same sample size. This was expected as the main goal of this algorithm is to ensure space coverage. *DENDIS*, whose aim is density representation, gave intermediate results.

To summarize, *ProTraS* proved to give similar, even better, results compared with *AdaCor* while being as fast as *AdaGri*, and without any post-processing step, in contrast with *DENDIS* or *DIDES*. The main characteristics of the compared algorithms are given in Table 8.

Table 8: Comparison of the different algorithms

	Input params	Internal params	Stopping criterion	Cons	Pros
DIDES	Granularity	Yes	Distance	Post-processing on density	Space coverage
DENDIS	Granularity	Yes	Density	Post-processing on distance	Density representation
AdaCor	β, ε or sample size	Yes	ε or sample size	Slow	Hybrid
AdaGri	α, N_{cut}, Max_b, Occ_m	No	Based on input params	Tuning may be difficult	Fast
ProTraS	ε	No	ε		Fast, ε

5.4. Wilcoxon test

To assess how significant the differences between *ProTraS* and the four studied algorithms are, a Wilcoxon signed-rank test was performed on the mean *RI* values. The 29 synthetic and real world data sets were considered. The test was based upon the sign of the difference of the observed values, and the *R Project*[6] implementation was used. The results are given in Table 9.

Table 9: Wilcoxon signed-rank test results

Approx level		*ProTraS*	*DENDIS*	*DIDES*	*AdaCor*	*AdaGri*
high	mean	0.974	0.955	0.929	0.960	0.946
	p-value		0.00242	0.00008	0.00421	0.00005
low	mean	0.955	0.923	0.899	0.929	0.909
	p-value		0.00632	0.00001	0.00980	0.00015

For the two studied approximation levels the *RI* means, over the 29 data sets are reported for *ProTraS* and the alternative approach considered. The *p-value*

[6]https://r-project.org, wilcox.test function with paired=TRUE.

is the probability of observing the computed statistic (not shown), given the experimental conditions, under the null hypothesis, meaning that both RI values come from a unique population, i.e. there is no difference between *ProTraS* and its competitor.

The Wilcoxon test clearly supports the rejection of the null hypothesis in all cases: the *p-value* are all below 1%. As stated before, the closest results are obtained with *AdaCor* for the low approximation level.

6. Conclusion

The new sampling algorithm proposed in this paper achieves an iterative partitioning of the input space based on distance and density considerations. At each step a new item is added to the sample set. It is chosen as the farthest from the already selected one in a given group. *ProTraS* is a *fft*-based algorithm.

It is shown in the present work that such *fft* algorithms yield $(k, \varepsilon)-$coresets. For a given group, i, the approximation is proportional to the sampling cost, i.e. the product of the maximum within group distance, d_i, and the group cardinality, w_i. As these data are updated at each step of the algorithm, an upper estimate of the sampling cost is available without any additional computational effort.

ProTraS, in contrast to previous *fft* algorithms such as *DIDES* or *DENDIS*, aims to design a coreset. Hence, the sampling cost is given the central role. First, it is the unique and meaningful parameter. Second, it also serves as the stopping criterion: the algorithm stops when the desired approximation level is reached. No hidden parameters or procedures are required. Finally, it is also used to guide the sampling process: at each step the new representative is added to the group with the highest probability of cost reduction. This value is the combination of two basic probabilities: one according to the within group distance, the other according to the group cardinality. The semantics are the same: the higher the distance or the cardinality, the higher the probability. The probabilistic approach in *ProTraS* achieves a trade-off between space coverage

and density representation.

The tests carried out on a synthetic data set already used in (Kärkkäinen & Fränti, 2002) show that *ProTraS* shares some properties with the previous two *fft* algorithms. It is robust to noise, the sample size mainly depends on the data structure instead of on the data size and it is time optimized. As the first steps of the algorithm are the most time consuming, a grid approach could really impact the process time.

Numerical experiments using 29 data sets, both synthetic and real world, show that the sample yielded by *ProTraS* is representative of the initial set. For two famous clustering algorithms, *k-means* and *DBSCAN*, the partitions computed from the whole and the sample are similar. The time ratio, i.e. sampling time plus time to cluster the sample over time to cluster the whole data, proved extremely low for a smart, but costly, algorithm like *DBSCAN*, enhancing the usefulness of the sampling approach. Using the *k-means*, the interest of *ProTraS* is restricted to large and/or structured data sets. The sensitivity to the unique parameter showed that for a large range of distortion costs, the sample is able to accurately represent the whole according to the Rand Index, even with a reduced size. A comparison with competitors selected from previous studies showed that *ProTraS* yielded the best *RI* mean on average over all the data sets for the studied configurations. A Wilcoxon signed-rank test clearly supported the alternative hypothesis, i.e. the two *RI* series come from two distinct distributions, the *ProTraS* and the competitor ones, in all the configurations with a p-value below 1%.

The proposal could be improved to deal with a huge number of observations. Several ways could be explored or combined. To begin with, it is possible to speed up the first steps of the algorithm, using a grid. This is likely to impact the running time as the first steps are the most expensive ones. Other optimization techniques can also be envisaged such as a kd-tree implementation or a neighborhood graph.

A stratification strategy could be developed He et al. (2011): divide the strata, run the algorithm on each subset and combine the separate samples to

yield the final result. As there is no post-processing stage, a streaming version Guha & Mishra (2016) can also be imagined.

The main challenge is to deal with high dimensional data, to understand and visualize these data. The most common approaches propose a space reduction, using a feature selection or a space transform process, before sampling. These two objectives can be combined in a single optimization problem to design a system that performs a dual selection of features and patterns according to a given criterion. Some attempts are based on evolutionary algorithms Ros & Guillaume (2007). The proposal could be adapted to be included in such a system.

References

Agarwal, P. K., Har-Peled, S., & Varadarajan, K. R. (2004). Approximating extent measures of points. *J. ACM*, *51*, 606–635. doi:`http://doi.acm.org/10.1145/1008731.1008736`.

Alcalá, J., Fernández, A., Luengo, J., Derrac, J., García, S., Sánchez, L., & Herrera, F. (2010). Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic and Soft Computing*, *17*, 255–287.

Arthur, D., & Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms* (pp. 1027–1035). Society for Industrial and Applied Mathematics.

Celebi, M. E., Kingravi, H. A., & Vela, P. A. (2013). A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Systems with Applications*, *40*, 200–210.

Feldman, D., Faulkner, M., & Krause, A. (2011). Scalable training of mixture models via coresets. In *Advances in Neural Information Processing Systems* (pp. 2142–2150).

Fränti, P., & Virmajoki, O. (2006). Iterative shrinking method for clustering problems. *Pattern Recognition*, *39*, 761–765.

Fu, L., & Medico, E. (2007). Flame, a novel fuzzy clustering method for the analysis of dna microarray data. *BMC bioinformatics*, *8*, 3.

Gonzalez, T. F. (1985). Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, *38*, 293 – 306. URL: `http://www.sciencedirect.com/science/article/pii/0304397585902245`. doi:`http://dx.doi.org/10.1016/0304-3975(85)90224-5`.

Guha, S., & Mishra, N. (2016). Clustering data streams. In *Data Stream Management* (pp. 169–187). Springer.

Har-Peled, S., & Mazumdar, S. (2004). On coresets for k-means and k-median clustering. In *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing* STOC '04 (pp. 291–300). New York, NY, USA: ACM. doi:`10.1145/1007352.1007400`.

Hartigan, J. A. (1975). *Clustering Algorithms*. Wiley.

Hartigan, J. A., & Wong, M. (1979). A k-means clustering algorithm. *Applied Statistics*, *28*, 100–108.

He, Y., Tan, H., Luo, W., Mao, H., Ma, D., Feng, S., & Fan, J. (2011). Mr-dbscan: an efficient parallel density-based clustering algorithm using mapreduce. In *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on* (pp. 473–480). IEEE.

Ilango, M. R., & Mohan, V. (2010). A survey of grid based clustering algorithms. *International Journal of Engineering Science and Technology*, *2*, 3441–3446.

Jain, A., & Law, M. (2005). Data Clustering: A User's Dilemma. In *Proceedings of the First international conference on Pattern Recognition and Machine Intelligence* (pp. 1–10).

Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, *31*, 651–666.

Kärkkäinen, I., & Fränti, P. (2002). *Dynamic local search algorithm for the clustering problem*. Technical Report A-2002-6 Department of Computer Science, University of Joensuu Joensuu, Finland.

Karypis, G., Han, E.-H., & Kumar, V. (1999). Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, *32*, 68–75.

Kollios, G., Gunopulos, D., Koudas, N., & Berchtold, S. (2003). Efficient biased sampling for approximate clustering and outlier detection in large data sets. *IEEE Transactions on Knowledge and Data Engineering*, *15*, 1170–1187.

Lin, Y., Cunningham, G. A., & Coggeshall, S. V. (1997). Using fuzzy partitions to create fuzzy systems from input-output data and set the initial weights in a fuzzy neural network. *IEEE Transactions on Fuzzy Systems*, *5*, 614–621.

Linde, Y., Buzo, A., & Gray, R. (1980). An algorithm for vector quantizer design. *IEEE Transactions on Communications*, *28*, 84–95. doi:10.1109/TCOM.1980.1094577.

Ling, R. F. (1981). Cluster analysis algorithms for data reduction and classification of objects. *Technometrics*, *23*, 417–418.

Lloyd, S. P. . (1982). Least squares quantization in pcm. *IEEE Trans. Inf. Theor.*, *28*, 129–137.

Lv, Y., Ma, T., Tang, M., Cao, J., Tian, Y., Al-Dhelaan, A., & Al-Rodhaan, M. (2015). An efficient and scalable density-based clustering algorithm for datasets with complex structures. *Neurocomputing*, *In Press*.

Ma, X., Pan, Z., Li, Y., & Fang, J. (2015). High-quality initial codebook design method of vector quantisation using grouping strategy. *IET Image Processing*, *9*, 986–992.

Machová, K., Puszta, M., Barčák, F., & Bednár, P. (2006). A comparison of the bagging and the boosting methods using the decision trees classifiers. *Computer Science and Information Systems*, *3*, 57–72.

Macqueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *In 5-th Berkeley Symposium on Mathematical Statistics and Probability* (pp. 281–297).

Palmer, C. R., & Faloutsos, C. (2000). Density biased sampling: An improved method for data mining and clustering. In *ACM SIGMOD Intl. Conference on Management of Data* (pp. 82–92). Dallas.

Pintore, M., Audouze, K., Ros, F., & Chretien, J. R. (2002). Adaptive fuzzy partition in database mining: application to olfaction. *Data Science Journal*, *1*, 99–110.

Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, *66*, 846850.

Ros, F., & Guillaume, S. (2007). An efficient nearest neighbor classifier. In *Hybrid Evolutionary Systems* (pp. 131–150). Hybrid Evolutionary Systems, Studies in Computational Intelligence, Vol 75, Springer.

Ros, F., & Guillaume, S. (2016a). Dendis: A new density-based sampling for clustering algorithm. *Expert Systems with Applications*, *56*, 349–359. doi:10.1016/j.eswa.2016.03.008.

Ros, F., & Guillaume, S. (2016b). Dides: a fast and effective sampling for clustering algorithm. *Knowledge and Information Systems*, (p. In press). doi:10.1007/s10115-016-0946-8.

Rosenkrantz, D. J., Stearns, R. E., & Lewis, P. M., II (1977). An analysis of several heuristics for the traveling salesman problem. *SIAM journal on computing*, *6*, 563–581.

Sarma, T. H., Viswanath, P., & Reddy, B. E. (2013). Speeding-up the kernel k-means clustering method: A prototype based hybrid approach. *Pattern Recognition Letters*, *34*, 564–573.

Thompson, S. K. (2012). *Sampling*. (3rd ed.). Wiley.

Tran, T. N., Drab, K., & Daszykowski, M. (2013). Revised dbscan algorithm to cluster data with dense adjacent clusters. *Chemometrics and Intelligent Laboratory Systems*, *120*, 92–96.

Viswanath, P., Sarma, T., & Reddy, B. (2013). A hybrid approach to speed-up the k-means clustering method. *International Journal of Machine Learning and Cybernetics*, *4*, 107–117.

Wang, L., Leckie, C., Kotagiri, R., & Bezdek, J. (2011). Approximate pairwise clustering for large data sets via sampling plus extension. *Pattern Recognition*, *44*, 222–235.

Xiang, Z. (1997). Color image quantization by minimizing the maximum inter-cluster distance. *ACM Trans. Graph.*, *16*, 260–276. URL: `http://doi.acm.org/10.1145/256157.256159`. doi:`10.1145/256157.256159`.

Yager, R. R., & Filev, D. P. (1994). Generation of fuzzy rules by mountain clustering. *Journal of Intelligent and Fuzzy Systems*, *2*, 209–219.

Zahra, S., Ghazanfar, M. A., Khalid, A., Azam, M. A., Naeem, U., & Prugel-Bennett, A. (2015). Novel centroid selection approaches for kmeans-clustering based recommender systems. *Information Sciences*, .

Zhang, T., Ramakrishnan, R., & Livny, M. (1997). Birch: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, *1*, 141–182.

Zhong, C., Malinen, M., Miao, D., & Fränti, P. (2015). A fast minimum spanning tree algorithm based on k-means. *Information Sciences*, *295*, 1–17.

Zhou, S., Zhou, A., Jin, W., Fan, Y., & Qian, W. (2000). Fdbscan: a fast dbscan algorithm. *Ruan Jian Xue Bao*, *11*, 735–744.