Scalable Distributed Approximation of Internal Measures for Clustering Evaluation*

Federico Altieri† Andrea Pietracaprina† Geppino Pucci† Fabio Vandin†

Abstract

An important step in cluster analysis is the evaluation of the quality of a given clustering through structural measures of goodness. Measures that do not require additional information for their evaluation (but the clustering itself), called internal measures, are commonly used because of their generality. The most widely used internal measure is the *silhouette coefficient*, whose naïve computation requires a quadratic number of distance calculations, unfeasible for massive datasets. Surprisingly, there are no known general methods to efficiently approximate the silhouette coefficient of a clustering with rigorously provable high accuracy. In this paper, we present the first scalable algorithm to compute such a rigorous approximation for the evaluation of clusterings based on any metric distances. Our algorithm approximates the silhouette coefficient within a mere additive error $O(\varepsilon)$ with probability $1-\delta$ using a very small number of distance calculations, for any fixed $\varepsilon, \delta \in (0,1)$. We also provide a distributed implementation of the algorithm using the MapReduce model, which runs in constant rounds and requires only sublinear local space at each worker, thus making our estimation approach applicable to big data scenarios. An extensive experimental evaluation provides evidence that our algorithm returns highly accurate silhouette estimates, unlike competing heuristics, while running in a fraction of the time required by the exact computation.

Keywords— Clustering, Silhouette, MapReduce, Randomization, PPS

1 Introduction

Clustering is a fundamental primitive for the unsupervised analysis of datasets, and finds applications in a number of areas including pattern recognition, bioinformatics, biomedicine, and data management [2]. In its more general definition, clustering requires to identify groups of elements where each group exhibits high similarity among its members, while elements in different groups are dissimilar. Starting from this common definition, several algorithms have been proposed to identify clusters in a dataset [18], often formalizing clustering as an optimization problem (based on a cost function). The resulting optimization problems are usually computationally hard to solve, and algorithms providing rigorous approximations are often sought in such cases. More recently, the focus has been on developing efficient methods that scale to the massive size of modern datasets [4, 7, 9, 10, 24, 25] while still providing rigorous guarantees on the quality of the solution.

A common step after clustering has been performed is *clustering evaluation* (sometimes called clustering *validation*). Clustering evaluation usually employs an evaluation measure capturing the goodness of the structure of a clustering . Evaluation measures are classified into *unsupervised* or *internal* measures, which do not rely on external information, and *supervised* or *external* measures, which assess how well a clustering matches the structure defined by external information [31]. While external measures are useful only when additional external knowledge regarding the cluster structure of the data is available, internal measures find applications in every scenario.

The most commonly used internal measure for clustering evaluation is the *silhouette coefficient* [29] (for brevity, called *silhouette* in this paper). The silhouette of a clustering is the average silhouette of all elements in the clusters, and, in turn, the silhouette $s(e)$ of an element e in some cluster C is defined as the ratio $(b(e) - a(e))/\max\{a(e), b(e)\}$, where $a(e)$ is the average distance of e from the other elements of C, and $b(e)$ is the minimum average distance of e from the elements of another cluster C'. In other words, $s(e)$ provides and indication to what extent e is closer (on average) to elements in its cluster C than to elements in the "closest" cluster $C' \neq C$. The use of the silhouette for clustering evaluation is suggested by widely used data mining books [17, 31], and has found application in several important areas [19, 27, 30, 34]. The naïve computation of the silhouette for a clustering of n elements requires $\Theta(n^2)$ distance calculations, which is unfeasible for massive dataset that require distributed clustering solutions [5, 6, 14, 15, 24]. Surprisingly, while several methods have been proposed to efficiently cluster large datasets with rigorous guarantees on the quality of the solution, there are no methods to efficiently approximate the silhouette featuring *provably high accuracy.*

1.1 Related Work

While the silhouette is one of the most popular internal measures for clustering evaluation

*This work was supported, in part, by MIUR of Italy, under PRIN Project n. 20174LF3T8 AHeAD, and grant L. 232 (Dipartimenti di Eccellenza), and by the University of Padova under projects "STARS 2017: Algorithms for Inferential Data Mining" and "SID 2020: RATED-X".

†Dept. of Information Engineering, University of Padova, Italy. Email: {altieri,capri,geppo,vandinfa}@dei.unipd.it

[26, 32, 35], the quadratic complexity of the naïve exact calculation makes its use impractical for clusterings of very large datasets. For this reason, some attempts have been made to propose variants that are faster to compute, or to simplify its calculation in some special cases.

Hruschka et al. [20] present the *simplified silhouette* for the evaluation of clusterings obtained on the basis of objective functions involving squared Euclidean distances (e.g., k-means clusterings [2]). The simplified silhouette is a variant of the silhouette, where for each element e in a cluster, the aforementioned quantities $a(e)$ and $b(e)$ are redefined, respectively, as the (squared) distance between e and the centroid of its cluster, and of the closest centroid of another cluster. In this fashion, the complexity of the whole computation reduces to $O\left(nk\right)$. While Hruschka et al. [20] and Wang et al. [33] provide empirical evidence that the simplified silhouette can be an effective evaluation measure for clusterings returned by Lloyd's algorithm [23], there is no evidence of its effectiveness for other types of clusterings (e.g, clusterings based on other distance functions) and, moreover, the discrepancy between the original silhouette and the simplified silhouette can grow very large.

Frahling and Sohler [16] proposed an optimization heuristic for speeding-up the computation of the exact silhouette for clusterings based on Euclidean distances. For each element e of a cluster C, while the term $a(e)$ is computed according to its definition, in an attempt to reduce the operations needed to compute the term $b(e)$, the heuristic first determines the average distance $d(e, C')$ between e and the elements of the cluster $C' \neq C$, whose centroid is closest to e, and then sets $b(e) = d(e, C')$ in case the distance between e and the centroid of any other cluster $C'' \notin \{C, C'\}$ is larger than or equal to $d(e, C')$, since in this case there is no need to compute any other average distance $d(e, C'')$. However, when this is not the case, $b(e)$ must be computed according to its definition and its worst case complexity remains clearly quadratic.

The Apache Spark programming framework[1] provides optimized methods for computing the silhouette of clusterings under d-dimensional squared Euclidean distances and under one formulation of cosine distance. Indeed, in these specific cases, simple algebra suffices to show that precomputing, for each of the k clusters, a limited number of values dependent on the coordinates of the cluster's points, yields a fully parallelizable algorithm featuring $O(nkd)$ work. However, using squared distance measures to compute the silhouette tends to push the measure closer to 1 compared to linear distances, thus amplifying positive and negative scores.

In our algorithm, we employ a *Probability Proportional to Size* (PPS) sampling scheme that samples each element with probability proportional to a "size" measure. The use of PPS sampling has been pioneered in the context of distance query processing and successfully applied to computing closeness centralities in graphs [11]. In the context of clustering, PPS has been used to obtain samples whose clustering cost approximates the clustering cost of the whole dataset, with guarantees on the quality of the approximation [12]. To the best of our knowledge, prior to our work, the use of PPS for efficient clustering evaluation had not been explored.

[1]`https://spark.apache.org/`

1.2 Our Contributions

In this work, we target the problem of the efficient computation of an accurate estimate of the silhouette of a given clustering under general metric distances. In this regard, our contributions are:

- We develop the first efficient, sampling-based algorithm for estimating the silhouette with provable approximation guarantees. For any fixed $\varepsilon, \delta \in (0, 1)$, our algorithm approximates the silhouette of a k-clustering of n elements within an additive error $4\varepsilon/(1-\varepsilon)$ with probability at least $1-\delta$, using only $O\left(nk\varepsilon^{-2}\log(nk/\delta)\right)$ distance computations, which constitutes a dramatic improvement over the $\Theta\left(n^2\right)$ distance computations required by the naïve exact algorithm.
- We provide a distributed implementation of our algorithm using the Map-Reduce framework [13, 28], which runs in constant rounds, and requires only sublinear space at each worker.
- We perform an extensive suite of experiments on real and synthetic datasets to assess the effectiveness and efficiency of our algorithm, and to compare it with known approaches for fast silhouette computation/approximation. The experiments show that, in practice, our algorithm provides silhouette estimates featuring very low absolute error (less than 0.01 in most cases) with small variance ($< 10^{-3}$) using a very small fraction of the distance computations required by the exact calculation. Moreover, the estimates returned by our algorithm are far superior in quality to those returned by a naïve yet natural heuristics based on uniform sampling, or by the simplifed silhouette [20] under general distances. We also show that the MapReduce implementation of our algorithm enables the estimation of the silhouette for clusterings of massive datasets (e.g., 1 billion elements) for which the exact computation is out of reach.

Our algorithm addresses the problematic issues posed by existing approaches by providing an efficiently computable and provably accurate approximation of the silhouette, allowing the use of the silhouette coefficient for very large datasets, for which its exact computation is unfeasible, and without the need to recur to other efficiently computable surrogates (such as the simplified silhouette coefficient) which in many cases provide estimates so divergent from the exact silhouette that they may lead to very different conclusions in the aforementioned scenarios of application.

In addition, while previously known approaches to efficiently compute or approximate the silhouette have been developed for specific distance functions, namely squared Euclidean and cosine distances, our algorithm provides provably accurate silhouette estimations for clusterings based on *any* metric distance. Finally, our algorithm can be generalized to compute rigorous approximations of other internal measures, such as cohesion and separation.

1.3 Organization of the paper The rest of the paper is structured as follows. Section 2 contains the description of our proposed strategy for silhouette estimation, its accuracy and performance analysis, and the MapReduce implementation. Section 3 reports the results of an extensive suite of experiments performed to evaluate the effectiveness and scalability of our silhouette estimation algorithm on synthetic and real datasets. Section 4 concludes the paper with some final remarks. For space constraints, some technical details, the extension of our approach to the cohesion and separation measures, and the description of additional experiments have been omitted in the paper, but they can be found in the extended version [3].

2 Methods

Consider a metric space U with distance function $d(\cdot,\cdot)$, and let $V = \{e_1, \dots e_n\} \subseteq U$ be a dataset of n elements. Let also $\mathcal{C} = \{C_1, \dots C_k\}$ be a *k-clustering* of V, that is, a partition of V into k disjoint non-empty subsets called *clusters*. A popular measure to assess clustering quality was introduced by Rousseeuw in 1987 [29]. Specifically, the *silhouette of an element* $e \in V$ belonging to some cluster C, is defined as

$$s(e) = \frac{b(e) - a(e)}{\max\{a(e), b(e)\}},$$

where

$$a(e) = \frac{\sum_{e' \in C} d(e,e')}{|C| - 1}, \qquad b(e) = \min_{C_j \neq C} \frac{\sum_{e' \in C_j} d(e,e')}{|C_j|}$$

denote, respectively, the average distance of e from the other elements in its cluster, and the minimum average distance of e from the elements in some other cluster. The *silhouette of the clustering* $\mathcal{C}$ is the average silhouette of all the elements of V, namely:

$$s(\mathcal{C}) = \frac{\sum_{i=1}^{n} s(e_i)}{n}. \tag{2.1}$$

From the above definitions it immediately follows that the values $s(e)$, with $e \in V$, and $s(\mathcal{C})$ range in $[-1, 1]$. A positive value for $s(e)$ implies that e has been assigned to an appropriate cluster, while a negative value implies that there might be a better cluster where e could have been placed. Therefore, $s(e)$ can be interpreted as a measure of the quality of the clustering from the perspective of element e. In turn, $s(\mathcal{C})$ provides a global measure of the quality of the whole clustering, where a value closer to 1 indicates higher quality. The exact computation of $s(\mathcal{C})$ requires $O(n^2)$ distance calculations, which is prohibitive when dealing with large datasets. In the following subsection, we present a randomized strategy to yield an estimate of $s(\mathcal{C})$, which is accurate within a provable error bound, with sufficiently high probability, and amenable to an efficient distributed computation.

2.1 A Fast Algorithm for Silhouette Estimation

Consider the estimation of the silhouette $s(\mathcal{C})$ for a k-clustering $\mathcal{C} = \{C_1, \dots C_k\}$ of a set V of n elements from a metric space U. For each $e \in V$ and $C_j \in \mathcal{C}$, define

$$W_{C_j}(e) = \sum_{e' \in C_j} d(e,e').$$

Note that for an element e of a cluster C, the quantities $a(e)$ and $b(e)$ in the definition of the silhouette $s(e)$ can be rewritten as $a(e) = W_C(e)/(|C| - 1)$ and $b(e) = \min_{C_j \neq C}\{W_{C_j}(e_i)/|C_j|\}$. Building on this observation, our approach to approximating $s(\mathcal{C})$ is based on estimating the $W_{C_j}(e)$'s by exploiting the *Probability Proportional to Size* (PPS) sampling strategy used in [11]. In particular, for each cluster $C_j \in \mathcal{C}$, a suitable small sample S_{C_j} is computed, and, for every element e, the value of $W_{C_j}(e)$ is approximated in terms of the distances between e and the elements of S_{C_j}, ensuring a user-defined error bound with high probability. The elements of S_{C_j} are chosen according to a carefully designed probability distribution which privileges the selection of elements that are distant from some suitable "central" element of the cluster. The details are given in what follows.

Consider a fixed error tolerance threshold $0 < \varepsilon < 1$ and a probability $0 < \delta < 1$. Our algorithm, dubbed PPS-SILHOUETTE (whose pseudocode can be found in [3]), consists of two *steps*. In Step 1, for each cluster $C_j \in \mathcal{C}$, the algorithm computes a sample S_{C_j} of expected size $t = \lceil (c/2\varepsilon^2)\ln(4nk/\delta) \rceil$ for a suitably chosen constant c, while in Step 2 the approximate values of the $W_{C_j}(e)$'s and, in turn, the approximations of the $s(e)$'s are computed through the S_{C_j}'s. More precisely, in Step 1, each cluster C_j is processed independently. If $|C_j| \le t$, then S_{C_j} is set to C_j, otherwise, *Poisson sampling* is performed over C_j, that is, each element $e \in C_j$ is included in S_{C_j} independently with a suitable probability p_e. Probability p_e, for $e \in C_j$, is determined as follows:

- First, an initial sample $S^{(0)}_{C_j}$ is selected, again by Poisson sampling, where each $e \in C_j$ is included in $S^{(0)}_{C_j}$ independently with probability $(2/|C_j|)\ln(2k/\delta)$. This initial sample will contain, with sufficiently high probability, the aforementioned "central" element of C_j, namely, one that is close to a majority of the elements of C_j, a property which is necessary to enforce the quality of the final sample S_{C_j};
- For each $\bar{e} \in S^{(0)}_{C_j}$ the value $W_{C_j}(\bar{e})$ is computed, and, for each $e \in C_j$, p_e is set to $\min\{1, t\gamma_e\}$, where γ_e is the maximum between $1/|C_j|$ (corresponding to uniform sampling) and the maximum of the values $d(e,\bar{e})/W_{C_j}(\bar{e})$, with $\bar{e} \in S^{(0)}_{C_j}$ (representing the relative contributions of e to the $W_{C_j}(\bar{e})$'s of the sample points).

In Step 2, for each $e \in V$ and each cluster C_j, the sample S_{C_j} is used to compute the value

$$\hat{W}_{C_j}(e) = \sum_{e' \in S_{C_j}} \frac{d(e,e')}{p_{e'}}$$

which is an accurate estimator of $W_{C_j}(e)$, as will be shown by the analysis. Once all these values have been computed, then, for each $e \in V$ belonging to a cluster C we compute the estimates $\hat{a}(e) = \hat{W}_C(e)/(|C| - 1)$ and $\hat{b}(e) = \min_{C_j \neq C}\{\hat{W}_{C_j}(e)/|C_j|\}$, which are in turn used to estimate the silhouette as $\hat{s}(e) = (\hat{b}(e) - \hat{a}(e))/(\max\{\hat{a}(e), \hat{b}(e)\})$. Finally, we estimate the silhouette of the whole clustering as

$$\hat{s}(\mathcal{C}) = \frac{\sum_{e \in V} \hat{s}(e)}{n}. \tag{2.2}$$

2.2 Analysis

In this section we show that, with probability $1 - \delta$, the value $\hat{s}(\mathcal{C})$ computed by PPS-Silhouette approximates the true silhouette $s(\mathcal{C})$ within a small error bound, expressed in terms of ε. The key ingredient towards this goal, stated in the following theorem, is a probabilistic upper bound on the relative error of the estimate $\hat{W}_{C_j}(e)$ with respect to true value $W_{C_j}(e)$.

Theorem 2.1. *There is a suitable choice of the constant c in the definition of the expected sample size t used by* PPS-Silhouette*, which ensures that, with probability at least $1-\delta$, for every element e and every cluster C_j, the estimate $\hat{W}_{C_j}(e)$ is such that*

$$\left| \frac{\hat{W}_{C_j}(e) - W_{C_j}(e)}{W_{C_j}(e)} \right| \leq \varepsilon.$$

Proof. The proof mimics the argument devised in [11]. Recall that $t = \lceil \frac{c}{2\varepsilon^2} \ln(4nk/\delta) \rceil$. Consider an arbitrary cluster C_j with more than t elements (in the case $|C_j| \leq t$ the theorem follows trivially). For an element $e \in C_j$, let $m(e)$ denote the median of the distances from e to all other elements of C_j. Element e is called *well positioned* if $m(e) \leq 2\min_{e' \in C_j} m(e')$. It is easy to see that at least half of the elements of C_j are well positioned. Hence, the initial random sample $S^{(0)}_{C_j}$ will contain a well positioned element with probability at least $(1 - (2/|C_j|)\ln(2k/\delta))^{|C_j|/2} \geq 1 - \delta/(2k)$. An easy adaptation of the proof of [11, Lemma 12], shows that if $S^{(0)}_{C_j}$ contains a well positioned element and c is a suitable constant, then the Possion sample S_{C_j} computed with the probabilities derived from $S^{(0)}_{C_j}$ is such that $|(\hat{W}_{C_j}(e) - W_{C_j}(e))/W_{C_j}(e)| \leq \varepsilon$, with probability at least $1 - \delta/(2nk)$. By the union bound, it follows that the probability that there exists a cluster C_j such that the initial sample $S^{(0)}_{C_j}$ does not contain a well positioned element is at most $k\delta/(2k) = \delta/2$. Also, by conditioning on the fact that for all clusters C_j the initial sample $S^{(0)}_{C_j}$ contains a well positioned node, by using again the union bound we obtain that the probability that there exists an element e and a cluster C_j for which $|(\hat{W}_{C_j}(e) - W_{C_j}(e))/W_{C_j}(e)| > \varepsilon$ is at most $nk\delta/(2nk) = \delta/2$, which concludes the proof. □

From now on, we assume that the relative error bound stated in Theorem 2.1 holds for every element $e \in V$ and every cluster C_j, an event which we will refer to as *event E*. Consider an arbitrary element e, and let $\hat{s}(e)$ be the estimate of the silhouette $s(e)$ computed by PPS-Silhouette. The following key technical lemma, whose lengthy proof can be found in [3], establishes a bound on the absolute error of the estimate.

Lemma 2.1. *If event E holds, then $|\hat{s}(e) - s(e)| \leq 4\varepsilon/(1-\varepsilon)$.*

An upper bound to the absolute error incurred when estimating $s(\mathcal{C})$ through the value $\hat{s}(\mathcal{C})$ computed by PPS-Silhouette, is established in the following theorem, whose proof is an immediate consequence of the definition of the two quantities (Equations 2.1 and 2.2), and of Theorem 2.1 and Lemma 2.1.

Theorem 2.2. *Let V be a dataset of n elements, and let $\mathcal{C}$ be a k-clustering of V. Let $\hat{s}(\mathcal{C})$ be the estimate of the silhouette of the clustering $s(\mathcal{C})$ computed by* PPS-Silhouette *for given parameters ε and δ, with $0 < \varepsilon, \delta < 1$, and for a suitable choice of constant $c > 0$ in the definition of the sample size. Then,*

$$|\hat{s}(\mathcal{C}) - s(\mathcal{C})| \leq \frac{4\varepsilon}{1-\varepsilon}$$

with probability at least $1 - \delta$.

We now analyze the running time of PPS-Silhouette, assuming that the distance between two points can be computed in constant time. In Step 1, the running time is dominated by the computation of the distances between the points of each sufficiently large cluster C_j and the points that form the initial sample $S^{(0)}_{C_j}$. A simple application of the Chernoff bound shows that, with high probability, from each such cluster C_j, $O(\log(nk/\delta))$ points are included in S^j_0. Thus, Step 1 performs $O(n\log(nk/\delta))$ distance computations, altogether. For what concerns Step 2, its running time is dominated by the computation of the distances between all points of V and the points of the union of all samples. A simple adaptation of the proof of [11, Corollary 11] and a straightforward application of the Chernoff bound shows that there are $O(kt)$ sample points overall, with high probability, where t is the expected sample size for each cluster. Recalling that $t = \lceil (c/2\varepsilon^2)\ln(4nk/\delta) \rceil$, we have that Step 2 performs $O(nkt) = O\left((nk\varepsilon^{-2})\log(nk/\delta)\right)$ distance computations overall. As a consequence, the running time of the algorithm is $O\left(nk\varepsilon^{-2}\log(nk/\delta)\right)$ which, for reasonable values of k, ε and δ, is a substantial improvement compared to the quadratic complexity of the exact computation.

We remark that while the ε^{-2} factor in the upper bound on the sample size might result in very large samples to achieve high accuracy, this quadratic dependence on ε reflects a worst-case scenario and is needed to obtain the analytical bound. In fact, Section 3 will provide experimental evidence that, in practice, rather modest sample sizes are sufficient to achieve high levels of accuracy.

2.3 Map-Reduce Implementation

To enable the estimation of the silhouette of clusterings of very large pointsets, we demonstrate that our approximation strategy can be effectively ported to a distributed setting. Specifically, in this section, we outline an implementation of PPS-Silhouette in *MapReduce* [13, 21, 22, 28], which is one of the reference frameworks for big data computing. (Full details are provided in [3].) We recall that a MapReduce algorithm executes a sequence of *rounds*. In a round, a multiset X of key-value pairs is first transformed into a new multiset X' of key-value pairs by applying a given *map function* independently to each pair, and then into a further multiset Y of pairs by applying a given *reduce function* independently to each subset of pairs of X' sharing the same key. On a parallel platform, several instances of the map (resp., reduce) function of a round can be distributed among the available computation nodes. Relevant performance indicators for a MapReduce algorithm are: the number of rounds, the amount M_L of *local memory* required by any instance of the map/reduce functions, and the amount M_A of *aggregate memory* required in each round.

Algorithm PPS-Silhouette can be implemented in 4 MapReduce rounds as follows. In order to exploit parallelism, the points of V are partitioned into w subsets V_i of size n/w each, for $1 \leq i \leq w$, that will be processed in parallel, where $w \in [0, n-1]$ is a design parameter used to exercise suitable parallelism-memory tradeoffs. The first two rounds are needed to select the initial Poisson sample $S_C^{(0)}$ and to compute the value $W_C(\bar{e})$ for each cluster C and $\bar{e} \in \cup_C S_C^{(0)}$. These values are made available to all partitions so that in each V_i, in parallel, the sample probabilities $p(e)$ for every $e \in V_i$ can be computed by the end of the second round. Based on these probabilities, in the third and fourth rounds the final Poisson sample $\cup_C S_C$ is selected so that in each V_i, in parallel, all values $\hat{s}(e)$ with $e \in V_i$ can be computed and summed together, yielding the final estimate $\hat{s}_{\mathcal{C}}$.

Overall, the above 4-round MapReduce algorithm requires local memory $M_L = O\left(n/w + (w + 1/\varepsilon^2)k \log(nk/\delta)\right)$, with high probability, and aggregate memory $M_A = O\left(wM_L\right)$. It is easy to see that for fixed values of ε and δ, and for $k = O\left(n/\log n\right)$, by choosing $w = \Theta\left(\sqrt{n/(k \log n)}\right)$, we obtain local memory $M_L = O\left(\sqrt{nk \log n}\right)$ and linear aggregate memory, with high probability. Observe that for reasonable values of k, the required local memory is substantially sublinear, which is the "holy grail" of MapReduce algorithms [28].

We remark that a similar implementation can be devised for the *Massively Parallel Computation* (MPC) model [8], which is a popular alternative to MapReduce for big data processing.

3 Experimental Evaluation

We ran extensive experiments with the twofold objective of assessing the quality and evaluating the performance of our PPS-Silhouette algorithm. Concerning quality assessment, we evaluate the accuracy of the estimate provided by PPS-Silhouette, and compare it against known heuristic and exact (specialized) baselines. For what concerns performance, we evaluate the scalability of the MapReduce implementation of PPS-Silhouette and compare its performance against the one of the aforementioned baselines.

3.1 Baselines We gauge the performance of our PPS-Silhouette algorithm against five *baselines*. The first baseline is the exact computation based on the definition. The second baseline is a variation of PPS-Silhouette, where the samples S_{C_j} are chosen via a uniform Poisson sampling, using the same probability $p(e) = t/|C_j|$ for each $e \in C_j$. The other three baselines implement, respectively, the simplified silhouette of [20, 33], the Frahling-Sohler optimization of the exact computation [16], and the optimized exact method available in the Apache Spark library for squared Euclidean distances (see Section 1.1) Our quality assessment compares PPS-Silhouette against the approximation baselines, namely uniform sampling and simplified silhouette, while our performance evaluation compares PPS-Silhouette against all baselines except for simplified silhouette, since the latter turned out to have low accuracy.

3.2 Implementation and Environment For quality assessment, we devised Java-based sequential implementations of PPS-Silhouette, of the exact computation, based on the definition, and of the uniform sampling and simplified silhouette baselines. For performance evaluation, we devised MapReduce implementations, using the Apache Spark programming framework with Java[2], of all baselines except simplified silhouette and the Apache Spark optimized method, whose code is provided in the Spark library. The MapReduce implementation of the uniform sampling baseline is patterned after the one of PPS-Silhouette, by only changing the sampling probabilities. The MapReduce implementation of the exact silhouette computation aims at minimizing the computational load per worker by evenly partitioning the operations among the workers and by providing each worker with an entire copy of the dataset. Finally, the MapReduce implementation of the Frahling-Sohler optimization enforces the optimistic scenario where the heuristics for the $b(e)$ term, in the computation of $s(e)$, is always successful, in order to appreciate the maximum speed-up that can be achieved over the non-optimized implementation[3].

In our experiments we fixed $\delta = 0.1$, resulting in a 90% confidence on the approximation guarantee, and we opted to control accuracy by varying the composite parameter t directly rather than fixing the independent variables ε and c separately (see Section 2.2).

The experiments were run on CloudVeneto[4], an institutional cloud, which provided us a cluster of 9 PowerEdge M620 nodes, each with octa-core Intel Xeon E5-2670v2 2.50 GHz and 16 GB of RAM, running Ubuntu 16.04.3 LTS with Hadoop 2.7.4 and Apache Spark 2.4.4.

[2]`https://spark.apache.org`

[3]All our implementations are available at `https://github.com/CalebTheGame/AppxSilhouette`

[4]`http://cloudveneto.it`

Table 1: Maximum and average absolute error of the silhouette estimates returned by PPS-Silhouette (label "PPS") and the uniform sampling algorithm, (label "uniform") relative to the k-medoid clustering of the synthetic dataset with $n = 2 \cdot 10^4$ points, $k = 2, 3 \ldots, 10$, and $t = 64, 256$ and 1024, alongside with absolute error of the simplified silhouette. Values $< 10^{-3}$ are denoted by 0^*.

k	**Absolute Error**												
		PPS						**Uniform**					
	Simplified silhouette	t**=64**		t**=256**		t**=1024**		t**=64**		t**=256**		t**=1024**	
		max	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**
2	.283	.005	.001	.002	0^*	.001	0^*	.207	145	.206	.143	.199	.116
3	.404	.018	.005	.008	.002	.004	.001	.369	.235	.403	.217	.392	.163
4	.026	.012	.003	.005	.001	.002	0^*	.492	.291	.473	.301	.458	.150
5	.561	.015	.005	.008	.002	.004	.001	.174	.404	.487	.322	.438	.159
6	.314	.101	.015	.034	.007	.010	.002	.174	.082	.126	.049	.052	.005
7	.449	.047	.010	.016	.004	.010	.002	.280	.174	.273	.121	.097	.025
8	.321	.084	.017	.028	.006	.008	.002	.168	.075	.149	.349	.013	.002
9	.455	.078	.013	.015	.003	.003	0^*	.295	.187	.254	.188	.068	.002
10	.457	.050	.013	.013	.003	.003	0^*	.284	.180	.246	.108	.071	.002

3.3 Datasets We considered both synthetic and real datasets. Synthetic datasets have been chosen so to contain a few outlier points, with the intent of making the accurate estimation of the silhouette more challenging. Specifically, for different values of n (ranging from tens of thousands to one billion), we generated $n - 10$ points uniformly at random within the sphere of unit radius, centered at the origin of the 3-dimensional Euclidean space, and 10 random points on the surface of the concentric sphere of radius 10^4. For real datasets, we used reduced versions of the "Covertype" and "HIGGS" datasets[5]. The former contains 100000 55-dimensional points, corresponding to tree observations from four areas of the Roosevelt National Forest in Colorado; the latter contains 500000 7-dimensional points and is used to train learning algorithms for high-energy Physics experiments (as in previous works [10, 24], only the 7 summary attributes of the original 28 have been retained). For all datasets, the clusterings used to test the algorithms have been obtained by applying the k-medoids clustering algorithm implemented in the Java Machine Learning Library[6] [1], using the Euclidean distance.

3.4 Quality Assessment In the first experiment, we compared the accuracy of the silhouette estimations returned by our PPS-Silhouette algorithm and by the uniform sampling algorithm, using synthetic data. In order to make the computation of the exact value feasible, we considered instances of the synthetic dataset with a relatively small number of elements ($n = 2 \times 10^4$) and k-clusterings $\mathcal{C}$ with $k \in \{2, \ldots, 10\}$. As explained in Section 3.2, the accuracy of PPS-Silhouette, and, clearly, also of the uniform sampling algorithm, depends on the (expected) sample size t. Hence, to test different levels of accuracy, for each value of k we conveniently varied the value t directly. We remark that, in general, controlling the sample size t directly, rather than indirectly through the several parameters contributing to its definition, is more effective in practice since it affords a better tailoring of the estimation to the available resources. Specifically, we ran both PPS-Silhouette and the uniform sampling algorithm with $t \in \{64, 128, 256, 512, 1024\}$. As a measure of accuracy for the estimated silhouette $\hat{s}(\mathcal{C})$, we use the absolute error $|\hat{s}(\mathcal{C}) - s(\mathcal{C})|$. Table 1 reports maximum and average of the absolute error over 100 runs of the two algorithms, for each configuration of parameters k and a selection of t's (the results for the other values of t, omitted for brevity, are similar to the ones reported in the table).

The results show that PPS-Silhouette provides a very accurate approximation to the silhouette already with $t = 64$, for which the average absolute error is at most 0.017 for all values of k and the maximum value is at most 0.084 for all cases but $k = 6$. The approximation quickly improves when larger values of t are considered. PPS-Silhouette also features a very low variance, which is $< 10^{-3}$ in all cases. These results show that our PPS-Silhouette algorithm provides a very accurate approximation of the silhouette even with a limited number of samples.

On the other hand, for $t = 64$ the uniform sampling algorithm provides estimates with considerably larger average and maximum absolute errors for most values of k, with an average error that is is one order of magnitude larger than the average error of PPS-Silhouette. The variance of the uniform sampling algorithm (0.05 on average) is also larger than the one of our algorithm. These results show that the use of PPS sampling is crucial to obtain good estimates of the silhouette while keeping the sample size small.

The second column of Table 1 shows the error that occurs when the simplified silhouette is used to approximate the exact value of the silhouette. Such error is always larger than twice the maximum error provided by PPS-

[5]`https://archive.ics.uci.edu/ml/datasets/`
[6]`https://github.com/AbeelLab/javaml`

Table 2: Comparison between exact silhouette, simplified silhouette, and Monte Carlo estimates obtained by averaging 100 estimates returned by PPS-Silhouette (label "PPS") and the uniform sampling algorithm (label "uniform"), for $k = 2, 3, \dots 10$ and $t = 64, 128$ and 1024. The highest value identified by each algorithm (one for each t for PPS and uniform) is highlighted in boldface. Observe that the "correct" value of k corresponds to the value with highest exact silhouette.

k	Silhouette							
	Exact	**Simplified**	**PPS**			**uniform**		
			t=64	**t=256**	**t=1024**	**t=64**	**t=256**	**t=1024**
2	0.064	0.347	0.064	0.064	0.064	**0.192**	**0.176**	0.074
3	-0.072	0.332	-0.070	-0.072	-0.072	0.151	0.081	-0.191
4	**0.343**	0.369	**0.343**	**0.344**	**0.343**	0.049	0.039	**0.183**
5	-0.228	0.333	-0.223	-0.228	-0.226	0.158	0.031	-0.244
6	0.034	0.348	0.022	0.032	0.034	0.121	0.071	0.040
7	-0.143	0.306	-0.148	-0.145	-0.143	0.033	-0.030	-0.130
8	0.059	**0.380**	0.038	0.055	0.058	0.134	0.094	0.057
9	-0.087	0.368	-0.102	-0.091	-0.088	0.102	0.021	-0.087
10	-0.104	0.353	-0.116	-0.017	-0.104	0.073	0.005	-0.103

Silhouette (even for the smallest values of the sample size t) and is approximately 0.36 on average, which is not surprising considering the weak relation between simplified silhouette and the exact value of the silhouette quantified by [33]. Therefore, the simplified silhouette does not provide accurate estimates of the exact value of the silhouette.

In the second experiment, we assessed the impact of relying on estimated rather than exact values in the most typical scenario of use of the silhouette, suggested in the original paper [29], that is, identifying the best granularity k when the same clustering algorithm is applied to a dataset with different values of k. Specifically, we ran k-medoids clustering on the synthetic dataset described above, for all values of k in an interval $[2, 10]$, and checked whether the best clustering (according to the exact silhouette) was identified using the silhouette estimations returned by PPS-Silhouette and by the uniform sampling algorithm with expected sample size t, considering values of $t \in \{64, 256, 1024\}$. Since uniform sampling displays a large variance affecting its silhouette estimates, we adopted a Monte Carlo approach, where the average of 100 independent estimates is used in lieu of a single estimate, to reduce the variance of the estimates. We also computed the deterministic value of the simplified silhouette [20, 33] to compare its effectiveness against the two sampling strategies. The results are shown in Table 2, where the exact value of the silhouette is also shown.

As expected, PPS-Silhouette always identifies the correct value already for $t = 64$, and, for all k and t, it provides extremely accurate approximations. On the other hand, the uniform sampling algorithm identifies the correct value of k only for $t = 1024$, and, for smaller values of t, it provides much weaker approximations (indeed, $t = 512$ is needed merely to hit the correct sign of the silhouette value for all k). Also, the table shows that the simplified silhouette is unable to identify the correct value of k.

In third experiment, we compared the accuracy of the silhouette estimates computed by PPS-Silhouette, uniform sampling, and simplified silhouette, on the real datasets HIGGS and Covertype, considering $k = 5, 10$ and $t = 64, 256, 1024$. Table 3 reports the the maximum and average absolute errors of the silhouette estimates, over 100 runs, together with the values of the exact silhouette and of the simplified silhouette. Similarly to the case of the synthetic dataset, PPS-Silhouette provides very accurate estimates already for $t = 64$, with an average error smaller than 0.03, and for $t \geq 256$ it features an average error below 0.01. The variance is below 10^{-3} in all cases. On these datasets, the uniform sampling algorithm also provides fairly accurate estimates (with variance $< 10^{-3}$ in all cases). However, for no combination of k and t the average error of the uniform sampling algorithm is lower than the one of PPS-Silhouette, while its maximum error is slightly better only in three out of twelve configurations. Also, the simplified silhouette displays poor accuracy, worse than the one of PPS-Silhouette even with $t = 64$. Once again, this experiment confirms the superiority of our PPS-Silhouette algorithm with respect to the two competitors.

In [3], we report on two additional experiments. One experiment, run on the synthetic dataset for various combinations of t and ranges of k, shows that out of 100 trials PPS-Silhouette always identifies the best value of k while the uniform sampling algorithm succeeds only a fraction of the times (very small in many cases). The other experiment provides evidence that PPS-Silhouette yields accurate estimates also when squared Euclidean distances (which do not satisfy the triangle inequality) are used.

Overall, the results show that PPS-Silhouette exhibits a high level of accuracy even for relatively small values of the expected sample size t. Since the total work performed by PPS-Silhouette is $O(nkt)$, this implies that reliable estimates can be obtained in much less than the $\Theta(n^2)$ work required by the exact computation.

Table 3: Comparison between exact silhouette, simplified silhouette, and maximum and average absolute error of the silhouette estimates returned by PPS-Silhouette (label "PPS") and the uniform sampling algorithm, (label "uniform"), relative to the k-medoid clustering of the real datasets Covertype and HIGGS, with $k = 5, 10$ and $t = 64, 256$ and 1024.

Dataset	**k**			**Absolute error**											
				PPS						**Uniform**					
		Silhouette		t = **64**		t = **256**		t = **1024**		t = **64**		t = **256**		t = **1024**	
		Exact	**Simpl.**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**
Covertype	5	0.344	0.474	.084	.016	.042	.006	.007	.002	.108	.019	.027	.007	.014	.003
	10	0.266	0.394	.120	.023	.025	.006	.014	.002	.160	.031	.040	.009	.014	.002
HIGGS	5	0.244	0.368	.114	.016	.024	.006	.008	.003	.110	.017	.045	.008	.019	.003
	10	0.151	0.289	.107	.028	.047	.010	.020	.004	.155	.033	.081	.012	.019	.005

3.5 Performance Evaluation

In this section, we report the results of several experiments aimed at comparing the distributed performance of the MapReduce implementation of PPS-Silhouette against the one of MapReduce implementations of the uniform sampling and the Frahling-Sohler optimization baselines, and of the optimized Spark routine for squared distances, on very large datasets.

In the first experiment, we assessed the scalability of the MapReduce version of PPS-Silhouette. For this experiment, we used four instances of the synthetic dataset (see Section 3.3) with $n = 10^6, 10^7, 10^8, 10^9$ elements and $k = 5$ clusters. We ran MapReduce implementation of PPS-Silhouette with $t = 64$ using $w = 1, 2, 4, 8, 16$ workers (each on a single core). For the dataset with $n = 10^9$ elements, we considered only $w = 4, 8, 16$ since for lower levels of parallelism the local memory available to each core was not sufficient to process the n/w elements assigned to each worker. The results are shown in Figure 1. We observe that for the two largest datasets PPS-Silhouette exhibits linear scalability. For $n = 10^7$, the algorithm still exhibits linear scalability for up to 8 workers, while there is a limited gain in using 16 workers. For $n = 10^6$, scalability is still noticeable, however, due to the relatively small size of the dataset, the behavior is less regular due to the stronger incidence of communication overheads and caching effects. Also, for a fixed number of workers, we observe that the algorithm scales linearly with the number of elements, in accordance with our theoretical analysis.

In successive experiments, we compared the running times of the MapReduce implementations of PPS-Silhouette (with $t = 64$) and of the baselines. We compared PPS-Silhouette with uniform sampling algorithm and times showed that, as expected, uniform sampling is slightly faster (by 10−20% in all experiments), due to the absence of the probability calculation phase. This mild advantage of uniform sampling is however counterbalanced by its much lower level of accuracy, quantified in the previous section, for equal sample size. We compared PPS-Silhouette with the exact computation based on the definition (DEF), and of the exact computation based on the Frahling and Sohler optimization (FS), with fixed, maximum degree of parallelism $w = 16$, and using the smallest dataset of the previous experiment, namely the one with $n = 10^6$ elements. While DEF was stopped after 6000 seconds, FS completed the execution in 1135 seconds, and PPS-Silhouette in 14 seconds, thus (more than) two orders of magnitude faster than DEF and FS. Also, we point out that PPS-Silhouette was able to estimate the silhouette for a three orders of magnitude larger dataset ($n = 10^9$) in 2433 seconds, which is about twice the time required by FS on the dataset with 10^6 elements. Considering the accuracy of the estimation provided by PPS-Silhouette, assessed in the previous subsection, these comparisons provide evidence of its practicality.

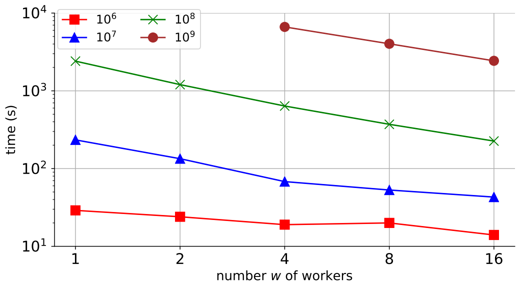

Figure 1: Running time (median over 5 runs) of our PPS-Silhouette algorithm as function of the number w of workers, for datasets of different sizes.

Finally, we compared the performance of PPS-Silhouette under squared Euclidean distances with the one of the optimized exact method available in Apache Spark for such distances, using our synthetic dataset with $n = 10^8$ elements and two values of k, namely $k = 5, 10$. Since our goal in this experiment was to compare the best performance achievable by the two implementations, and since our PPS-Silhouette algorithm provides very accurate estimates already with a limited number t of samples, we fixed $t = 32$ and $w = 16$ workers. For both values of k the estimates from PPS-Silhouette are very precise (average error 0.007 for $k = 5$ and 0.087 for $k = 10$), while the running time of PPS-Silhouette is comparable to, even if higher than (up to approximately two times), the running time of the optimized Apache Spark method.

4 Conclusions

In this work, we developed the first efficient, sampling-based algorithm to estimate the silhouette coefficient for a give clustering with provable approximation guarantees, whose running time dramatically improves over the quadratic complexity required, in the general case, by the exact computation. We provided a distributed implementation of our algorithm in Map-Reduce which runs in constant rounds, and requires only sublinear space at each worker. The experimental evaluation conducted on real and synthetic datasets, demonstrates that our algorithm enables an accurate estimation of the silhouette for clusterings of massive datasets for which the exact computation is out of reach. Future research should target a more extensive experimentation on real-world datasets. Also, it would be interesting to devise a more flexible, dynamic version of the algorithm where the desired accuracy can be incrementally refined by reusing (part of) the previously sampled points.

References

[1] T. Abeel, Y. V. d. Peer, and Y. Saeys. Java-ml: A machine learning library. *JMLR*, 2009.

[2] C. C. Aggarwal and C. K. Reddy. *Data Clustering: Algorithms and Applications*, 2014.

[3] F. Altieri, A. Pietracaprina, G. Pucci, and F. Vandin. Scalable distributed approximation of internal measures for clustering evaluation. ArXiv:2003.01430, 2020.

[4] P. Awasthi and M. Balcan. Center based clustering: A foundational perspective. In *Handbook of cluster analysis*, 2014.

[5] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii. Scalable k-means++. *PVLDB* 2012.

[6] M.-F. F. Balcan, S. Ehrlich, and Y. Liang. Distributed k-means and k-median clustering on general topologies. *NIPS* 2013.

[7] A. Barger and D. Feldman. k-means for streaming and distributed big sparse data. *SDM* 2016.

[8] P. Beame, P. Koutris, and D. Suciu. Communication steps for parallel query processing. *PODS* 2013.

[9] M. Ceccarello, C. Fantozzi, A. Pietracaprina, G. Pucci, and F. Vandin. Clustering uncertain graphs. *PVLDB* 2017.

[10] M. Ceccarello, A. Pietracaprina, and G. Pucci. Solving k-center clustering (with outliers) in MapReduce and streaming, almost as accurately as sequentially. *PVLDB* 2019.

[11] S. Chechik, E. Cohen, and H. Kaplan. Average distance queries through weighted samples in graphs and metric spaces: high scalability with tight statistical guarantees. *APPROX/RANDOM* 2015.

[12] E. Cohen, S. Chechik, and H. Kaplan. Clustering small samples with quality guarantees: adaptivity with one2all PPS. *AAAI* 2018.

[13] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Comm. ACM.*

[14] A. Ene, S. Im, and B. Moseley. Fast clustering using MapReduce. *KDD* 2011.

[15] D. Feldman and M. Langberg. A unified framework for approximating and clustering data. *STOC* 2011.

[16] G. Frahling and C. Sohler. A fast k-means implementation using coresets. *Int. J. Comp. Geom. Appl.* 2008.

[17] J. Han, J. Pei, and M. Kamber. *Data mining: concepts and techniques*, 2011.

[18] C. Hennig, M. Meila, F. Murtagh, and R. Rocci. *Handbook of cluster analysis*, 2015.

[19] M. S. Hossain and R. A. Angryk. Gdclust: A graph-based document clustering technique. *ICDMW 2007.*

[20] E. R. Hruschka, L. N. de Castro, and R. J. Campello. Evolutionary algorithms for clustering gene-expression data. *ICDM* 2004.

[21] H. J. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for mapreduce. *SODA* 2010.

[22] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of Massive Datasets*, 2020.

[23] S. P. Lloyd. Least squares quantization in PCM. *IEEE Trans. on Information Theory*, 1982.

[24] G. Malkomes, M. Kusner, W. Chen, K. Weinberger, and B. Moseley. Fast Distributed k-Center Clustering with Outliers on Massive Data. *NIPS* 2015.

[25] A. Mazzetto, A. Pietracaprina, and G. Pucci. Accurate mapreduce algorithms for k-median and k-means in general metric spaces. *ISAAC* 2019.

[26] D. Moulavi, P. A. Jaskowiak, R. J. G. B. Campello, A. Zimek, and J. Sander. Density-based clustering validation. *SDM* 2014.

[27] R. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. *VLDB* 1994.

[28] A. Pietracaprina, G. Pucci, M. Riondato, F. Silvestri, and E. Upfal. Space-round tradeoffs for mapreduce computations. *ICS* 2012.

[29] P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *J. of Comp. and App. Mathematics*, 1987.

[30] T. Sellam, R. Cijvat, R. Koopmanschap, and M. Kersten. Blaeu: mapping and navigating large tables with cluster analysis. *PVLDB* 2016.

[31] P. N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*, 2016.

[32] C. Tomasini, L. R. Emmendorfer, E. N. Borges, and K. S. Machado. A methodology for selecting the most suitable cluster validation internal indices. *SAC* 2016.

[33] F. Wang, H.-H. Franco-Penya, J. D. Kelleher, J. Pugh, and R. Ross. An analysis of the application of simplified silhouette to the evaluation of k-means clustering validity. *MLDM* 2017.

[34] C. Wiwie, J. Baumbach, and R. Röttger. Comparing the performance of biomedical clustering methods. *Nature methods*, 2015.

[35] H. Xiong and Z. Li. Clustering validation measures. *Data Clustering: Algorithms and Applications*, 2014.