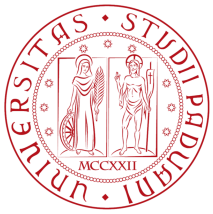

UNIVERSITÀ DEGLI STUDI DI PADOVA

Department of Information Engineering

Master's Degree in Computer Engineering

Master Thesis

A Local-Search Approach to Silhouette-Based Clustering

Supervisors

prof. Andrea Alberto Pietracaprina

prof. Geppino Pucci

prof. Fabio Vandin

dott. Ilie Sarpe

Candidate

Davide Peressoni

2008571

Academic Year 2021/2022

December 12, 2022

Abstract

Clustering aims to split a set of objects into k groups (called *clusters*), in such a way each cluster contains similar objects and dissimilar objects find place in different clusters. The silhouette coefficient is a metric which is widely used to evaluate the goodness of a clustering.

In this thesis are presented some methods to solve the clustering problem by optimizing the silhouette. The optimization is done by means of local search techniques.

The proposed algorithms underwent an exhaustive phase of testing, in comparison with the standard ones, on both real and synthetic datasets. From the results we can see how one of the proposed algorithms can be competitive with Lloyd's algorithm for k-means and also overcame other standard algorithms such as PAM for k-medoids.

Sommario

L'obiettivo del clustering è suddividere un insieme di oggetti in k partizioni (dette *cluster*), in modo che ogni cluster contenga oggetti simili e oggetti dissimili trovino posto in cluster diversi. La silhouette è una metrica molto usata per validare la bontà di un clustering.

In questa tesi sono presentati alcuni metodi per risolvere il problema del clustering, ottimizzando la silhouette. L'ottimizzazione viene effettuata grazie all'uso di tecniche di ricerca locale.

Gli algoritmi proposti sono stati sottoposti ad un'esaustiva fase di verifica sperimentale, comparandone i risultati con quelli degli algoritmi standard, sia su dataset reali che artificiali. Dai risultati possiamo vedere come uno degli algoritmi proposti può essere competitivo con l'algoritmo di Lloyd per k-means e addirittura battere altri algoritmi standard come PAM per k-medoids.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Clustering

Clustering is a common technique of *unsupervised machine learning*, which comes from statistical data analysis. The purpose of clustering is to split a set of objects into k groups (called *clusters*), in such a way each cluster contains similar objects and dissimilar objects find place in different clusters [AB84]. Clustering is very useful when we want to classify a set of objects without knowing the categories (*labels*) in advance. Hence, it is an important tool used in a wide spectrum of applications areas: e.g. image processing, object recognition, data compression, information retrieval, data mining, bioinformatics, network analysis, wireless sensor networks. [AR14]

1.1.1 Silhouette

There is a metric which is widely used [HPK11; TSK16] to evaluate the quality of a clustering: the *silhouette* coefficient [Rou87]. This is an *internal* measure, which means it does not depend on external information, but only on the clustering itself. The silhouette ranges from -1 to 1, whereas -1 indicates a low-quality clustering, whilst value towards 1 are a sign of a good clustering where close (similar) objects are in the same cluster and distant (dissimilar) objects in different clusters. In fact, taking into account not only the distance of a point from its cluster, but also with other clusters, the silhouette is an indicator of both intra-cluster similarity and inter-cluster dissimilarity.

1.1.2 Clustering strategies

A clustering method whose objective is to maximize the silhouette, which is the aim of this thesis, would be interesting since this metric captures both intra-cluster similarity and inter-cluster dissimilarity. However, it is difficult to achieve because the actual computation of the silhouette (i.e., of the objective function) is inefficient. For this reason classical clustering strategies usually involves the optimization of some simpler metrics. We will review the most common of them in Section 2.3.

An alternative approach to the clustering problem is hierarchical clustering. In these methods we typically start from the trivial clustering composed of n single-point clusters. Then we merge the *closest* clusters (according to the specified metric) until we found the desired number k of clusters, or another termination condition is met [Nie16].

1.2 Local search

Local search is a generic method with the aim of solving complex optimization problems [RN10]. Local search algorithms require the definition of a *neighborhood* relation among the possible states of the problem. In fact, they start from a feasible solution and move step-by-step to a neighbor solution until a goal state is found. Unfortunately local search does not always return the optimal solution, but it usually returns a good-enough one.

The local search strategy could be visualized as a connected graph whose nodes are the states of the problem and the edges represent the neighborhood relation. The local search starts from a node and moves along the edges to find a goal state. There are several strategies to choose the neighbor solution to move to, but all use only the information about neighbor states to make the choice (hence the search is local).

1.2.1 Hill climbing

The hill climbing is a local search technique which has a greedy approach to the selection of the next neighbor solution [SG06]. It computes the objective function for all the neighbor states and moves to the one which maximizes (or minimizes, for minimization problems) its value. Hill climbing would naturally end when no improving neighbor is found, but many times it is useful to reduce the number of iterations setting a termination condition such as the improvement is under a desired threshold or a fixed maximum number of iterations is reached.

It is easy to see how the main weakness of hill climbing is that it could possibly get stuck in points which are only locally optimum. An example can be seen in Fig. 1.1, where the height of each node represents the objective function value of

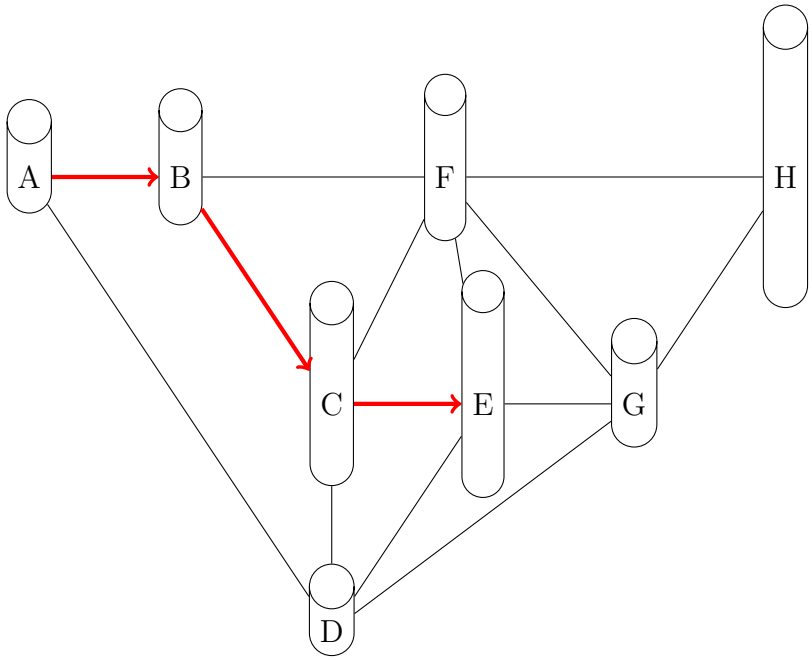

Figure 1.1: Hill climbing halts in local maximum.

the corresponding solution (in a maximization problem). If we start local search from node A we will move to B because the objective function has a greater value in B rather than in D. Then we will move, with the same reasoning, to C and then to E. Since all the neighbors of E (C, D, G and F) have a lower value of the objective function, hill climbing terminates. However, E is only a local maximum: the global maximum is H, which could be reached only staring the local search from F or G.

Many modifications of hill climbing which try to solve this problem are known: the most famous are restarts, randomization, iterated local search [LMS10] and simulated annealing [KGV83]. In particular simulated annealing allows, with a certain probability, to move to a state which worsens the objective function, with the aim to avoid local optima.

1.3 Previous work

The silhouette, as mentioned before, is widely used to validate a clustering, but it is not considered during the clustering optimization itself. There is one exception: when we do not known in advance the number k of clusters a very common technique is to make some clusterings with various values of k and then take the one with the

best silhouette [Lle+04]. But even if we use the silhouette to choose a clustering, it is not involved in the making of that clustering.

An alternative for hierarchical clustering is proposed in [NNA20]. We start with a cluster for each point, then we compute the *ExtASI* metric for each possible merge of two clusters. The ExtASI metric is a convex combination of the silhouette of the clustering and the ratio of points having positive silhouette. The merge giving the best ExtASI is retained and the algorithm is reapplied. At the end a dendogram of the merges is returned; each level has a different value of k and of ExtASI. The level with the highest ExtASI is selected. Like other hierarchical clustering techniques, this method is inefficient.

[LPB03] studies a variation of PAM (*Partitioning Around Medoids*) where the silhouette is optimized (PAMSIL). Since the resulting algorithm is extremely slow they tried using the medoid-based silhouette (MEDSIL) instead of the traditional silhouette. The resulting algorithm is called PAMMEDSIL. MEDSIL differs from the classical silhouette because it replaces the distance from a point to a cluster (i.e. the mean distance between a point and all the points of a cluster) with the distance between that point and the medoid of the desired cluster. Using the medoid-based silhouette enhances the performances but reduces the optimization power of the algorithm. Indeed, simplified silhouettes have been proven to be not a so good estimator of the real silhouette [Alt+21]. To further reduce the complexity of PAMMEDSIL, FastMSC and FasterMSC [LS22] were recently proposed. They exploit some tricks from FastPAM and FasterPAM [SR21] to reduce the time required computing the MEDSIL improvement by memorizing some partial results.

There are some studies in which the silhouette is used to guide the relocation of the points with the aim of improve an existing clustering. For example in [LRB21] and [Rob15] we could see two clusterings techniques, applied to the specific task of vegetation classification, which relocate misclassified points in a way which tries to maximize the silhouette coefficient. In particular the REMOS algorithms [LRB21] relocate the points which have a negative silhouette in the nearest cluster (REMOS1 relocates only the point with the worst silhouette, while REMOS2 relocates all points with negative silhouette). The other algorithm showed is OPTSIL [Rob15] which moves every point in every cluster and puts the corresponding silhouette in a priority queue. After each iteration the relocation which is atop of the queue is chosen as new starting point. This method converges very slowly. Also in [AT07] it is suggested to use the silhouette values to move misclassified objects in categorical clustering.

In [SJB17] a strategy is presented to optimize the silhouette using group search optimizations techniques. These optimization heuristics take inspiration

from animal searching behavior [HWS09]. Taking a group of different clusterings $\{\mathcal{C}^{(1)}, \mathcal{C}^{(m)}\}$ from the same set of points, we call producer ($\mathcal{C}^\star$) the one which gives the highest silhouette and divide the remaining into scroungers and rangers. For each clustering $\mathcal{C}^{(i)}$ in the group, a Poisson sample of its points is moved to another cluster: if the point $x \in C_a^{(i)}$ is sampled in a scrounger clustering $\mathcal{C}^{(i)}$, it is moved to the cluster $C_b^{(i)}$, of the same clustering, with the same label of the cluster $C_b^\star$ containing the point in the producer clustering; if instead it is in a ranger clustering the point is relocated to a random cluster $C_c^{(i)}$ of this clustering. At the end of each iteration the new producer is selected.

Moreover, there are recent studies focused on applying other animal-inspired optimization methods to the clustering problem. For example in [Gha+22] two *swarm intelligence* optimization techniques are investigated.

Local search has been also used to obtain a bounded approximated solution to the k-means problem [Kan+04]. Here the local step is the removal of a point from the set of centers, replacing it with a non-center point. The objective function is the same as k-means: the sum of all squared distance from each point to its center.

Furthermore, there are also some studies on applying the coreset technique to solve k-means. This technique, which we will use in Section 3.3, consists of solving the problem on a small sample (called *coreset*) of the input and then exploit this solution to generate a solution for the whole input [Aga+05]. Obviously the sample (coreset) must be a good representative of the full input for solving that specific problem. The strategy presented in [FS06] runs Lloyd's algorithm on the coreset and after some iterations repetitively doubles the size of the coreset finally reaching the entire instance; in this way the majority of iterations is done on a small set of points. A more classical approach is taken by the following algorithms where, after running a clustering algorithm on the coreset, the centers found on the coreset are used to find the centers for the whole point set. In [HM04] local search is used to swap the centers with other points, in [HK05] the points in the coreset are the center of mass of some chunks in which the dataset was divided. [FMS07] presents instead an approach were the points are sampled into the coreset with a probability depending on their distance to current centers: a similar PPS (Probability Proportional to Size, [Ski16]) strategy is used in [Alt+21] to build the sample used to approximate the silhouette. A distributed (MapReduce) coreset approach, for both k-means and k-median, is presented in [MPP19].

1.4 Summary of contributions

This thesis studies a new clustering method which, using a local-search approach, aims at finding a good solution to the silhouette-based clustering problem, whose

objective is to compute a clustering with maximum silhouette. We present several algorithms which implement this method trying to improve performance by deploying techniques such *memoization* and the usage of *coresets*. In particular we show as, to build the coreset, using a PPS (*probability proportional to size*) sample is more effective than a uniform sampling technique. For all the proposed algorithms we analyzed the theoretical time complexity, and compared their efficiency and effectiveness (regarding the silhouette of the resulting clustering) with standard algorithms not explicitly optimizing silhouette (i.e. k-means++, Lloyd's and PAM) via an extensive experimental analysis.

Among the proposed algorithms, the one exploiting both memoization and coresets is indeed competitive with the most common algorithm for center-based clustering. Indeed it always provides better solutions with respect to those computed by k-means++. It also gives results similar to Lloyd's algorithm both in time and in silhouette. But more importantly it works also with non-Euclidean distances: for example with Manhattan distance it reaches the results of PAM, but in much less time.

Another advantages of out approach, with respect to center-based clusterings, are that it does not require the definition of centers and that it is more scalable. Indeed, our best algorithm has complexity $O(nkt + rk^4t^2)$, where n is the number of points, k the number of clusters, t the expected sample size for each cluster, and r the number of local-search iterations. It can be noticed the term of the time complexity which includes the number of iterations r is separated by the term which includes the dataset size n: this means that for large datasets the number of iterations is not relevant. This is a great advantage over the previously cited algorithms, in which n and r are instead multiplied. For reference the complexity of Lloyd's algorithm is $O(rnk)$ and FastPAM $O(rn^2)$.

1.5 Structure of the thesis

In Chapter 2 we will briefly review the main theoretical concepts that are used in the thesis. We will provide the formal definition of clustering, of the silhouette metric, of the relevant clustering problems, and discuss the most popular algorithms to solve them.

Then, in Chapter 3, we will present our proposed method to solve the silhouette-based clustering problem using a local-search approach. We present an algorithm based on a straightforward implementation of the method, and several enhanced algorithms. All the algorithms are accompanied by a rigorous analysis of their time complexity and space usage.

After that, in Chapter 4, we will present and comment the results of a suite of experiments which have been carried out to evaluate our algorithms and to

compare them against some state-of-the-art ones.

Finally, in Chapter 5, we will resume our work and provide some suggestions for future research.

Chapter 2

Preliminaries

In this chapter we will introduce the clustering problem and discuss some techniques to solve it and evaluate the quality of the solutions. We will start by formally defining a clustering, then we will illustrate the most used validation metric: the silhouette index, along with an algorithm for its approximation. Finally, we will define what is a silhouette-based clustering, we will talk about a widespread family of clustering problems, known as the center-based clustering, and we will quickly review the most famous algorithms for this family.

2.1 Clustering

▷ **Definition 2.1** (Metric space)**.** A metric space (U, d) is a set U with a distance function $d\colon U \times U \to \mathbb{R}_{\geqslant 0}$.

The distance function must fulfill these four properties:

Symmetry $d(x, y) = d(y, x) \quad \forall\, x, y \in U$;

Null self distance $d(x, x) = 0 \quad \forall\, x \in U$;

Positivity $d(x, y) > 0 \quad \forall\, x \neq y \in U$;

Triangular inequality $d(x, y) + d(y, z) \geqslant d(x, z) \quad \forall\, x, y, z \in U$.

▷ **Definition 2.2** (Clustering)**.** Given a set $V \subseteq U$ from a metric space (U, d), we can formally define the clustering task as finding a partition $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ of V, that is

- $\bigcup_{i=1}^{k} C_i = V$;
- $C_i \cap C_j = \emptyset \;\forall i \neq j$.

The objective of the partition is to gather similar objects in the same cluster and put dissimilar objects in different clusters.

The distance function has the purpose to represent the dissimilarity between objects. As can be seen in Fig. 2.1, different distance functions may lead to different clusterings. In the example, using a distance function focused on color similarity, we obtain two clusters: one for blue objects, the other for green objects. If we supply instead a distance function associated to the shape dissimilarity, we expect a clustering in which a group contains squared objects, and the other circles.

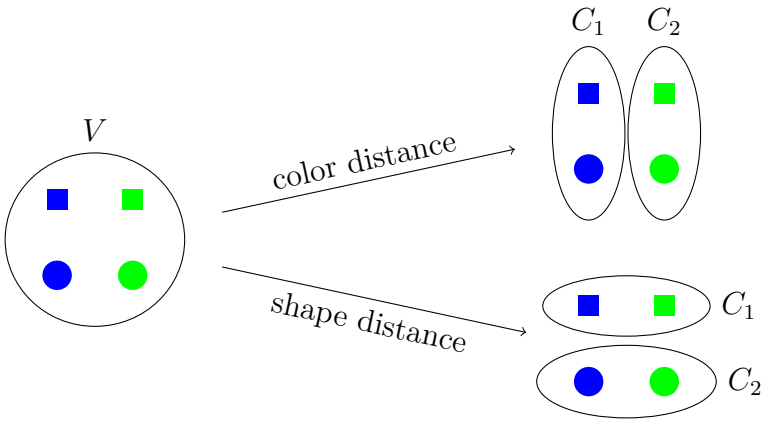

Figure 2.1: Different distance functions give different clusterings.

When objects are points in Euclidean space, and Euclidean distance is used, the interpretation of a clustering becomes quite intuitive as a grouping into well-separated sets of contiguous points (see Fig. 2.2).

2.2 Silhouette

As said in the introduction (see Section 1.1.1), the silhouette is the most used metric to validate a clustering. Its relevance relies on the fact that it takes care of both intra-cluster and inter-cluster distances and, being an internal measure, does not rely on additional information, other than the clustering itself. We will now provide the formal definition of this metric (Definition 2.3) and the algorithm to compute it (Algorithm 1).

▷ **Definition 2.3** (Silhouette)**.** Given a clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$ of V, the silhouette of a point $x \in C_i$ is

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

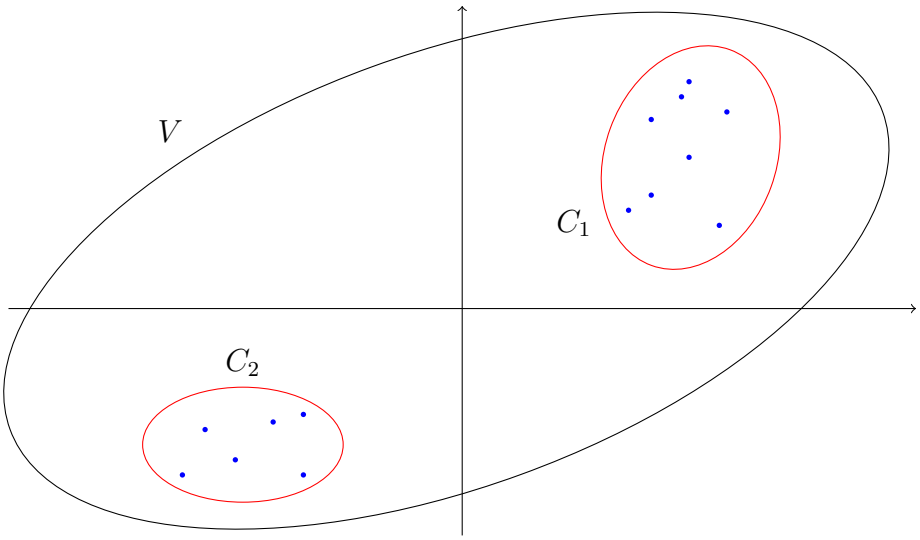

Figure 2.2: Example of clustering of points in a bi-dimensional Euclidean space.

where $a(x)$ is the mean distance from x to the objects of its cluster, and $b(x)$ the minimum mean distance of x from the points of another cluster:

$$a(x) = \frac{\sum_{y \in C_i} d(x,y)}{|C_i| - 1},$$
$$b(x) = \min_{i \neq j} \frac{\sum_{y \in C_j} d(x,y)}{|C_j|}.$$

Finally, the silhouette of a clustering is the average of the silhouettes of all points:

$$s(\mathcal{C}) = \frac{\sum_{x \in V} s(x)}{|V|}.$$

It is easy to see, from Algorithm 1, how all mutual distances between objects in V have to be computed in order to obtain the silhouette coefficient. From this we can say that the silhouette computation has a quadratic complexity $\Theta(n^2)$ in the cardinality $n = |V|$ of the point set.

2.2.1 Silhouette approximation

In [Alt+21] a method is shown to compute the silhouette with an approximation error bounded in high probability, whose complexity is $O(nk\varepsilon^{-2}\log(nk/\delta))$, where ε and δ are the parameters governing the approximation error. The algorithm is also

Algorithm 1: Silhouette computation.

Input: clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$
Output: silhouette of the clustering $\mathcal{C}$
$s \leftarrow 0$
let $V \coloneqq \bigcup_{i=1}^{k} C_i$ be the set of all points
for $i \leftarrow 1$ **to** k **do**
  if $|C_i| > 1$ **then**
    foreach $x \in C_i$ **do** // For each point x in the clustering
      $a \leftarrow \sum_{y \in C_i} d(x, y)/(|C_i| - 1)$
      $b \leftarrow \min_{j \neq i}\{\sum_{y \in C_j} d(x, y)/|C_j|\}$
      $s \mathrel{+}= (b - a)/\max(a, b)$
return $s/|V|$

easily distributable. The idea starts from breaking down the silhouette computation in two steps: the first step computes the *weight* of all points (Definition 2.4), the second computes the silhouette starting from the weights (Algorithm 2).

▷ **Definition 2.4** (Weight of a point)**.** Define the weight of a point x with respect to a cluster C_i as

$$W_i(x) \coloneqq \sum_{y \in C_i} d(x, y).$$

We can then compute $a(x)$ and $b(x)$ as

$$a(x) = \frac{W_i(x)}{|C_i| - 1} = \frac{\sum_{y \in C_i} d(x, y)}{|C_i| - 1},$$
$$b(x) = \min_{i \neq j} \frac{W_j(x)}{|C_j|} = \min_{i \neq j} \frac{\sum_{y \in C_j} d(x, y)}{|C_j|}.$$

[Alt+21] provides a way to compute an unbiased estimation of the weights. Providing the approximated weights to Algorithm 2 (sil_memo), we obtain an approximated silhouette $\hat{s}(\mathcal{C})$ such that $|\hat{s}(\mathcal{C}) - s(\mathcal{C})| \leqslant \frac{4\varepsilon}{1-\varepsilon}$.

Algorithm 2: Silhouette computed exploiting weights. (sil_memo)

Input: clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$, weights W of the clustering
Output: silhouette of the clustering $\mathcal{C}$ exploiting memoized weights W
$s \leftarrow 0$
let $V \coloneqq \bigcup_{i=1}^{k} C_i$ be the set of all points
for $i \leftarrow 1$ **to** k **do**
  if $|C_i| > 1$ **then**
    foreach $x \in C_i$ **do**
      $a \leftarrow W_i(x)/(|C_i| - 1)$
      $b \leftarrow \min_{j \neq i}\{W_j(x)/|C_j|\}$
      $s \mathrel{+}= (b - a)/\max(a, b)$
return $s/|V|$

Algorithm 3: Probability Proportional to Size sampling. (sample_pps)

Input: cluster C
Data: clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$, expected sample size t, required probability δ
Output: A PPS sample S_C of the given cluster. Each point $x \in S_C$ is associated with its probability p_x.
if $|C| \leqslant t$ **then return** C with $p_x = 1$
$u \leftarrow 1/|C|$
$p \leftarrow 2u \ln(2k/\delta)$
$S_C^{(0)} \leftarrow$ Poisson sample of points $x \in C$, with probability p
foreach $x \in S_C^{(0)}$ **do**
  $W_C(x) \leftarrow 0$
  foreach $y \in C$ **do**
    $W_C(x) \mathrel{+}= d(x, y)$
foreach $x \in C$ **do**
  $\gamma \leftarrow \max_{y \in S_C^{(0)}} \{d(x, y)/W_C(y)\}$
  $p_x \leftarrow \min(1, t \cdot \max(u, \gamma))$
$S_C \leftarrow$ Poisson sample of points $x \in C$, with probability p_x
return S_C with p_xs

The weight estimation is done by computing them on a PPS (Probability Proportional to Size) sample [Ski16]. The sampling technique is illustrated in Algorithm 3 (sample_pps). For each sample S_{C_i} (of the cluster C_i), the approximated weight

$\hat{W}_i(x)$ is the sum of the distances between x and each point y of the sample S_{C_i}, divided by the probability p_y with which y was sampled:

$$\hat{W}_i(x) = \sum_{y \in S_{C_i}} \frac{d(x,y)}{p_y}.$$

Finally, in Algorithm 4, we can see all the components assembled to compute the approximated silhouette.

Algorithm 4: Approximated silhouette computation.

Input: clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$
Output: approximated silhouette $\hat{s}$ of clustering $\mathcal{C}$
let $V \coloneqq \bigcup_{i=1}^{k} C_i$ be the set of all points
for $i \leftarrow 1$ **to** k **do**
    $S_{C_i} \leftarrow$ sample_pps(C_i)
    foreach $x \in V$ **do** $\hat{W}_i(x) \leftarrow \sum_{y \in S_{C_i}} \frac{d(x,y)}{p_y}$
return sil_memo($\mathcal{C}, \hat{W}$)

2.3 Clustering problems

Clustering problems are optimization problems which aim at finding a clustering of a set optimizing a given objective function. For example the silhouette-based clustering (Definition 2.5) goal is to find a clustering which maximizes the silhouette. In this thesis we will explore some algorithms which make use of local search techniques to obtain an approximate solution of the silhouette-based clustering problem.

▷ **Definition 2.5** (Silhouette-based clustering)**.** Given a set $V \subseteq U$ from a metric space (U, d), solving the silhouette-based clustering problem is to find a clustering (Definition 2.2) $\mathcal{C} = \{C_1, \ldots, C_k\}$ of V which maximizes the silhouette (Definition 2.3):

$$\mathcal{C}^\star = \operatorname*{argmax}_{\mathcal{C}} s(\mathcal{C}).$$

The benefit of silhouette-based clustering is that not only does it minimize the distances of a point from those of the same cluster (intra-cluster distance), but also maximizes the distances from a point to the points of other clusters (inter-cluster distance). Despite this benefit, silhouette-based clustering is not used yet, aside from some preliminary work (see Section 1.3). In fact, simpler metrics, which

involve only intra-cluster distances, are commonly used as objective function. The most famous family is that of center-based clustering (Definition 2.6).

▷ **Definition 2.6** (Center-based clustering)**.** Given a set $V \subseteq U$ from a metric space (U, d), the center-based clustering problem is the family of optimization problems with the following objective functions, which capture intra-cluster dissimilarity:

k**-means** $\min_{\mu_1,\mu_2,\ldots,\mu_k \in U, \mathcal{C}} \sum_{i=1}^{k} \sum_{x \in C_i} d(x, \mu_i)^2$;

k**-medians** $\min_{\mu_1,\mu_2,\ldots,\mu_k \in U, \mathcal{C}} \sum_{i=1}^{k} \sum_{x \in C_i} d(x, \mu_i)$;

k**-medoids** $\min_{\mu_1,\mu_2,\ldots,\mu_k \in V, \mathcal{C}} \sum_{i=1}^{k} \sum_{x \in C_i} d(x, \mu_i)$;

k**-center** $\min_{\mu_1,\mu_2,\ldots,\mu_k \in U, \mathcal{C}} \max_{i=1}^{k} \max_{x \in C_i} d(x, \mu_i)$

where $\mathcal{C} = \{C_1, \ldots, C_k\}$ is a clustering of V.

In summary the goal is to find a center μ_i for each cluster C_i, and assign each point to the best cluster in such a way the sum of squared distances (for k-means), the sum of distances (for k-medians and k-medoids) or the maximum distance (for k-center) between a point and the center of its cluster is minimized. The difference between k-medians and k-medoids is that in the former the centers are part of the metric space set U, whilst in the latter they must be elements of the set V. So, in k-medoids, the center of the cluster is indeed a point of the cluster, and it is called *medoid*.

All of these problems are NP-hard, but fortunately there are several approximation algorithms for them which give a good solution. In particular for the k-means problem in Euclidean space, we have the well known Lloyd's heuristic [Llo82; WH07] which, given a good selection of initial centers μ_i, usually returns a good clustering doing $O(rnk)$ distance computations, where r is the required number of iterations of the algorithm. It is not guaranteed that Lloyd's algorithm would return the optimal solution, or a solution with a specified approximation guarantee, since it could get trapped in a local minimum [SI84]. The algorithm (5) is divided in two parts: the first computes the centroid for each cluster $\mu_i = \sum_{x \in C_i} \frac{x}{|C_i|}$, whilst the second redistributes the points assigning each point to the cluster with the nearest centroid. The two parts are repeated until a local minimum is found (the objective function does not improve anymore).

For what concerns the computation of the initial clustering $\mathcal{C}$, the k-means++ algorithm (6) is widely used [VA06]. It starts by taking the first center randomly from the set of elements V, and then it picks up the next $k-1$ centers from the remaining elements of V. Each element has probability to be selected as center proportional to the sum of its squared distances with the already chosen centers. The obtained centers could be used as a starting point for other center-based

Algorithm 5: Lloyd's algorithm for k-means.

Input: clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$
Output: clustering $\mathcal{C}^\star = \{C_1^\star, \ldots, C_k^\star\}$ optimized for k-means
let $V \coloneqq \bigcup_{i=1}^{k} C_i$ be the set of all points
$\mathcal{C}^\star \leftarrow \mathcal{C}$ // Current optimal clustering
$\phi^\star \leftarrow \infty$ // Objective function value for the current optimal clustering
do
  $\phi \leftarrow \phi^\star$
  $\phi^\star \leftarrow 0$
  for $i \leftarrow 1$ **to** k **do** // Compute centers
    $\mu_i \leftarrow \sum_{x \in C_i^\star} x / |C_i^\star|$
  for $i \leftarrow 1$ **to** k **do** $C_i^\star \leftarrow \{\}$
  foreach $x \in V$ **do** // Assign points to the nearest cluster
    $i \leftarrow \operatorname{argmin}_j d(x, \mu_j)$
    $\phi^\star \mathrel{+}= d(x, \mu_i)^2$
    $C_i^\star \leftarrow C_i^\star \cup \{x\}$
while $\phi^\star < \phi$
return $\mathcal{C}^\star$

Algorithm 6: k-means++.

Input: set V, number of clusters k
Output: clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$ over V. Each cluster C_i is associated with its center $\mu_i \in C_i$.
let $M = \{\mu_1, \ldots, \mu_k\}$ be the (initially empty) set of centers chosen so far
$\mu_1 \leftarrow \mathrm{Unif}(V)$ // Pick first center at random
for $i \leftarrow 2$ **to** k **do**
  $\alpha = \sum_{x \in V \setminus M} \sum_{\mu \in M} d(x, \mu)^2$ // Probability normalization factor
  foreach $x \in V \setminus M$ **do** // Compute probabilities
    $p_x = \sum_{\mu \in M} d(x, \mu)^2 / \alpha$
  sample new center μ_i from $V \setminus M$ according to probability distribution p
foreach $x \in V \setminus M$ **do**
  $i_x \leftarrow \operatorname{argmin}_j d(x, \mu_j)$ // Find the center which minimizes the distance
for $j \leftarrow 1$ **to** k **do**
  $C_j \leftarrow \{\mu_j\} \cup \{x \in V \setminus M : i_x = j\}$ // Assign points to cluster of the nearest center
return $\mathcal{C}$ with μ_is

algorithms, or instead to already build a clustering simply assigning each point to the cluster with the nearest center (as in the second part of Lloyd's algorithm). We remark how the centers returned by k-means++ are *medoids* (the centers are elements of the set V), and so they can be used as a starting point also for algorithms solving the k-medoids problem.

Algorithm 7: Partitioning Around Medoids (PAM) for k-medoids.

Input: set V, initial medoids $M = \{\mu_1, \ldots, \mu_k\} \subset V$
Output: clustering $\mathcal{C}^\star = \{C_1^\star, \ldots, C_k^\star\}$ over V, optimized for k-medoids

```
φ* ← ∞
do
    φ ← φ*
    i* ← 0                                          // To save the best medoid swap
    for i ← 1 to k do
        foreach x ∈ V \ M do
            μ̄_i ← μ_i
            μ_i ← x                                 // Swap medoid μ_i and point x
            // Compute new objective function value
            φ ← 0
            for j ← 1 to k do C_j ← {μ_j}
            foreach y ∈ V \ M do
                j ← argmin_l d(y, μ_l)
                φ += d(y, μ_j)
                C_j ← C_j ∪ {y}
            if φ < φ* then
                φ* ← φ
                C* ← C = {C_1, ..., C_k}
                i* ← i
                x* ← x
            μ_i ← μ̄_i                               // Restore medoid μ_i
    if i* ≠ 0 then μ_{i*} ← x*                      // Apply medoid swap
while φ* < φ
return C*
```

The principal algorithm for k-medoids is Partitioning Around Medoids (PAM) [KR90], which could indeed be used also with other objective functions involving the selection of medoids. As it can be seen in Algorithm 7, it computes a local search swapping a medoid with another point of the set and evaluating the new objective function to move to the best configuration. The main drawback of PAM

is its time complexity: in the naïve implementation it is $O(rn^2k^2)$. Storing the old distances allows simplifying the computation of the new objective function value, reducing the cost by a factor $O(k)$. Finally the FastPAM algorithm exploits a trick derived from splitting in parts the computation of the cost and achieves a complexity of $O(rn^2)$ [SR21], which is $O(k^2)$ faster, but is still squared in the number of elements of V.

Chapter 3

Algorithms for Silhouette-Based Clustering

The approaches here presented aims to solve the clustering problem by optimizing the silhouette coefficient using the local search technique of hill climbing. The neighborhood of a solution is the moving of each point in each cluster: each neighbor differs from the current state only for a point which is relocated in another cluster.

3.1 The naïve algorithm

The naïve way to achieve such local search is presented in Algorithm 8 (Naïve). It begins stating that the input clustering $\mathcal{C}$ is the so on best clustering $\mathcal{C}^\star$, then, at each iteration, it tries to move each point x of the current reference clustering $\mathcal{C}$ from its cluster C_i to all other clusters C_j. If one of these new clusterings $\mathcal{C}'$ has a better silhouette score it becomes the new optimal clustering $\mathcal{C}^\star$. Finally the optimal clustering becomes the reference clustering for the next iteration. When the improvement on the silhouette is lower than a threshold η the so on optimal clustering is returned.

It is easy to see that the number of silhouettes to be computed is $1+r\cdot n\cdot(k-1)$, where the clustering is on a set of $n = |V|$ points and r iterations are required. So, computing the silhouette in the naïve way, the complexity is $\Theta\left(rn^3k\right)$.

To reduce the complexity we could instead use an approximated silhouette. For example, as early said in Section 2.2.1, [Alt+21] presents a way to compute an approximated silhouette with bounded error $\frac{4\varepsilon^2}{1-\varepsilon}$ with probability at least $1-\delta$, in time $O(nk\varepsilon^{-2}\log(nk/\delta))$. Using this approximated silhouette, the overall complexity of our local search would lower to $O\left(rn^2k^2\varepsilon^{-2}\log(nk/\delta)\right)$. The use of the approximated silhouette could lead to some false positives (a clustering with

Algorithm 8: Naïve implementation of the local search.

Input: clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$
Output: clustering $\mathcal{C}^\star = \{C_1^\star, \ldots, C_k^\star\}$, optimized for silhouette

```
C* ← C                                   // Current optimal clustering
s* ← sil(C)                              // Silhouette of the current optimal clustering
do
    s ← s*                               // Silhouette of the reference clustering
    for i ← 1 to k do
        foreach x ∈ C_i do               // For each point x in the clustering
            C'_i ← C_i \ {x}
            for j ← 1 to k, j ≠ i do     // For each cluster, except the one containing
                x
                C'_j ← C_j ∪ {x}
                C' ← C \ {C_i, C_j} ∪ {C'_i, C'_j}      // Move x from C_i to C_j
                s' ← sil(C')
                if s' > s* then
                    C* ← C'
                    s* ← s'
    C ← C*                               // Update the reference clustering for the next iteration
while s* − s > η                         // Repeat until there is no significative improvement
return C*
```

lower silhouette being chosen as optimal) and false negatives (a clustering with a higher silhouette being discarded). To take care of this when two silhouettes are compared the approximation error must be taken into consideration: the condition $s' > s^\star$ must become $\hat{s'} > \hat{s^\star} + \bar{\varepsilon}$, and the ending condition $s^\star - s > \eta$ should be substituted by $\hat{s'} - \hat{s^\star} > \eta + \bar{\varepsilon}$.

▷ **Lemma 3.1.** *Fixing $\bar{\varepsilon} \geqslant \frac{8\varepsilon}{1-\varepsilon}$ the false positive probability is*

$$\mathbb{P}\left(\hat{s'} > \hat{s^\star} + \bar{\varepsilon} \;\middle|\; s' < s^\star\right) \leqslant 2\delta + \delta^2$$

Proof.

$$\mathbb{P}\left(\hat{s'} > \hat{s^\star} + \bar{\varepsilon} \;\middle|\; s' < s^\star\right) = \mathbb{P}\left(\hat{s'} - \hat{s^\star} > \bar{\varepsilon} \;\middle|\; s' < s^\star\right) =$$
$$= \mathbb{P}\left(\left(\hat{s'} - s'\right) - \left(\hat{s^\star} - s^\star\right) > \underbrace{(s^\star - s')}_{>0} + \bar{\varepsilon} \;\middle|\; s^\star - s' > 0\right) \leqslant$$
$$\leqslant \mathbb{P}\left(\underbrace{\left(\hat{s'} - s'\right) - \left(\hat{s^\star} - s^\star\right)}_{<|\hat{s'}-s'|+|\hat{s^\star}-s^\star|} > \bar{\varepsilon} \;\middle|\; s^\star - s' > 0\right) \leqslant \mathbb{P}\left(|\hat{s'} - s'| + |\hat{s^\star} - s^\star| > \bar{\varepsilon}\right) =$$

Calling $e' = |\hat{s'} - s'|$ and $e^\star = |\hat{s^\star} - s^\star|$

$$\begin{aligned}
=&\mathbb{P}\left(e' + e^\star > \bar{\varepsilon} \;\middle|\; e' < \frac{4\varepsilon}{1-\varepsilon}, e^\star < \frac{4\varepsilon}{1-\varepsilon}\right)\mathbb{P}\left(e' < \frac{4\varepsilon}{1-\varepsilon}\right)\mathbb{P}\left(e^\star < \frac{4\varepsilon}{1-\varepsilon}\right) + \\
&\mathbb{P}\left(e' + e^\star > \bar{\varepsilon} \;\middle|\; e' > \frac{4\varepsilon}{1-\varepsilon}, e^\star > \frac{4\varepsilon}{1-\varepsilon}\right)\mathbb{P}\left(e' > \frac{4\varepsilon}{1-\varepsilon}\right)\mathbb{P}\left(e^\star > \frac{4\varepsilon}{1-\varepsilon}\right) + \\
&\mathbb{P}\left(e' + e^\star > \bar{\varepsilon} \;\middle|\; e' < \frac{4\varepsilon}{1-\varepsilon}, e^\star > \frac{4\varepsilon}{1-\varepsilon}\right)\mathbb{P}\left(e' < \frac{4\varepsilon}{1-\varepsilon}\right)\mathbb{P}\left(e^\star > \frac{4\varepsilon}{1-\varepsilon}\right) + \\
&\mathbb{P}\left(e' + e^\star > \bar{\varepsilon} \;\middle|\; e' > \frac{4\varepsilon}{1-\varepsilon}, e^\star < \frac{4\varepsilon}{1-\varepsilon}\right)\mathbb{P}\left(e' > \frac{4\varepsilon}{1-\varepsilon}\right)\mathbb{P}\left(e^\star < \frac{4\varepsilon}{1-\varepsilon}\right)
\end{aligned}$$

Recalling that

$$\mathbb{P}\left(|\hat{s} - s| > \frac{4\varepsilon}{1-\varepsilon}\right) \leqslant \delta, \quad \mathbb{P}(\bullet) \leqslant 1$$
$$\mathbb{P}\left(e' + e^\star > \bar{\varepsilon} \;\middle|\; e' < \frac{4\varepsilon}{1-\varepsilon}, e^\star < \frac{4\varepsilon}{1-\varepsilon}\right) \leqslant \mathbb{P}\left(\frac{8\varepsilon}{1-\varepsilon} > \bar{\varepsilon}\right) = 0$$

we can conclude

$$\mathbb{P}\left(\hat{s'} > \hat{s^\star} + \bar{\varepsilon} \;\middle|\; s' < s^\star\right) \leqslant 0 \cdot 1 \cdot 1 + 1 \cdot \delta \cdot \delta + 1 \cdot 1 \cdot \delta + 1 \cdot \delta \cdot 1 = 2\delta + \delta^2$$

□

▷ **Corollary 3.1.1.** *Keeping $\bar{\varepsilon} > \frac{8\varepsilon}{1-\varepsilon}$ the false negative probability, in the bad case where the discarded silhouette is higher of more than $2\bar{\varepsilon}$, is:*

$$\mathbb{P}\left(\hat{s'} < \hat{s^\star} + \bar{\varepsilon} \;\middle|\; s' > s^\star + 2\bar{\varepsilon}\right) \leqslant 2\delta + \delta^2$$

Proof.

$$\mathbb{P}\left(\hat{s'} < \hat{s^\star} + \bar{\varepsilon} \;\middle|\; s' > s^\star + 2\bar{\varepsilon}\right) = \mathbb{P}\left(\hat{s'} - \hat{s^\star} < \bar{\varepsilon} \;\middle|\; s' - s^\star > 2\bar{\varepsilon}\right) =$$

$$= \mathbb{P}\left(\left(\hat{s'} - s'\right) - \left(\hat{s^\star} - s^\star\right) < \bar{\varepsilon} \underbrace{-(s' - s^\star)}_{<-2\bar{\varepsilon}} \;\middle|\; s' - s^\star > 2\bar{\varepsilon}\right) \leqslant$$

$$\leqslant \mathbb{P}\left(\left(\hat{s'} - s'\right) - \left(\hat{s^\star} - s^\star\right) < -\bar{\varepsilon}\right) = \mathbb{P}\left(\left(s' - \hat{s'}\right) - \left(s^\star - \hat{s^\star}\right) > \bar{\varepsilon}\right) \leqslant$$

$$\leqslant \mathbb{P}\left(|\hat{s'} - s'| + |\hat{s^\star} - s^\star| > \bar{\varepsilon}\right) \leqslant 2\delta + \delta^2$$

□

3.2 Introducing memoization

In Algorithm 8 (Naïve) for each relocation we are recomputing the whole silhouette, which is an expensive task. Many times frequent expensive computations can be mitigated using memoization. This caching strategy consist in saving some results which will not change in some contiguous iteration and reuse their saved values instead of recomputing them [Mic68]. Memoization exchanges so time complexity for memory occupation.

Even if the change in the clustering given by a local step is minimal, potentially the silhouette of all points could change. Obviously the ones of the two clusters involved in the move, but also the silhouettes of other points could be affected since the cluster involved in the computation of $b(x)$ could change. For this reason we have to recompute from scratch the silhouette each time. Fortunately, as we saw in Definition 2.4, we can write the silhouette formula in a different way, as suggested by [Alt+21], which allows us to memoize part of its computation. Thus, given the weights W, we can easily compute the silhouette in time $O(nk)$, as can be seen in Algorithm 2 (sil_memo). This computation has a smaller complexity than the usual one, but requires in input the weights of the clustering.

Since, in our local search, a lot of weights in the computation of silhouettes do not change (for example those of the clusters not interested in the moving of the point x), we can memoize them, as can be seen in Algorithm 9 (ExactMemoization). The new part of the algorithm is highlighted in blue: at the beginning all weights are computed and stored in a matrix W of size $k \times n$. The weights W' of a temporary clustering $\mathcal{C}'$ are essentially the weights W of the current reference cluster $\mathcal{C}$, except for clusters C_i and C_j: for each i (j) the new weights are computed. Before passing to a new cluster C_i (C_j) the previous weights are restored. This update operation is very easy since to recompute the weights of a cluster after the moving of a point, it is sufficient to only add/subtract the distances involving that point. In fact

Algorithm 9: Optimize exact silhouette with memoization. (ExactMemoization)

Input: clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$
Output: clustering $\mathcal{C}^\star = \{C_1^\star, \ldots, C_k^\star\}$, optimized for silhouette
let $V := \bigcup_{i=1}^{k} C_i$ be the set of all points
for $i \leftarrow 1$ **to** k **do**
  foreach $x \in V$ **do** $W_i(x) \leftarrow \sum_{y \in C_i} d(x, y)$ // Memoize weights
$\mathcal{C}^\star \leftarrow \mathcal{C}$, $W^\star \leftarrow W$
$s^\star \leftarrow \text{sil\_memo}(\mathcal{C}, W)$
do
  $s \leftarrow s^\star$
  $W' \leftarrow W$ // The weights of each move will be based on those of the new reference clustering
  for $i \leftarrow 1$ **to** k **do**
    foreach $x \in C_i$ **do**
      $C_i' \leftarrow C_i \setminus \{x\}$
      foreach $y \in V$ **do** $W_i'(y) \mathrel{-}= d(x, y)$
      for $j \leftarrow 1$ **to** k, $j \neq i$ **do**
        $C_j' \leftarrow C_j \cup \{x\}$
        $\mathcal{C}' \leftarrow \mathcal{C} \setminus \{C_i, C_j\} \cup \{C_i', C_j'\}$
        foreach $y \in V$ **do** $W_j'(y) \mathrel{+}= d(x, y)$
        $s' \leftarrow \text{sil\_memo}(\mathcal{C}', W')$
        if $s' > s^\star$ **then**
          $\mathcal{C}^\star \leftarrow \mathcal{C}'$, $W^\star \leftarrow W'$
          $s^\star \leftarrow s'$
        $W_j' \leftarrow W_j$ // Restore weights of C_j
    $W_i' \leftarrow W_i$ // Restore weights of C_i
  $\mathcal{C} \leftarrow \mathcal{C}^\star$, $W \leftarrow W^\star$
while $s^\star - s > \eta$
return $\mathcal{C}^\star$

$$W_i'(y) = \sum_{z \in C_i'} d(y, z) = \sum_{z \in C_i \setminus \{x\}} d(y, z) = \sum_{z \in C_i} d(y, z) - d(x, y) = W_i(y) - d(x, y);$$
$$W_j'(y) = \sum_{z \in C_j'} d(y, z) = \sum_{z \in C_j \cup \{x\}} d(y, z) = \sum_{z \in C_j} d(y, z) + d(x, y) = W_j(y) + d(x, y).$$

The algorithm always computes the exact silhouette.

The computational complexity is

$$\sum_{i=1}^{k} O(n|C_i|) + T_{sil} + r\sum_{i=1}^{k}|C_i|\left(O(n) + \sum_{j=0,j\neq i}^{k}(O(n) + T_{sil})\right) =$$

$$= O\left(n^2\right) + O(nk) + r\sum_{i=1}^{k}|C_i|\left(O(n) + O\left(nk\right) + O\left(nk^2\right)\right) = O\left(rn^2k^2\right)$$

where $T_{sil} = O(nk)$.

The memory required for the memoization is $O(nk)$.

▷ **Lemma 3.2** (Number of iterations). *The theoretical maximum number of required iterations r is $O\left(\min\left(\eta^{-1}, k^n\right)\right)$*

Proof. In the worst case the starting silhouette could be -1. At each iteration the silhouette grows at least of η, so in $\frac{2}{\eta}$ iterations it reaches at least 1, which is the maximum reachable value.

On the other side the clustering combinations are finite, in particular they are $O(k^n)$. We never go to a state we had been before since it would not bring any enhancement in silhouette. □

3.3 Using coresets

We could think that using approximated silhouette in Algorithm 9 (ExactMemoization), as we suggested for Algorithm 8 (Naïve), would reduce the time complexity, but this is not the case. The time complexity would indeed increase to $O\left(rn^2k\left(\frac{\log(nk/\delta)}{\varepsilon^2} + k\right)\right)$. This is due to the fact that the approximated silhouette presented in [Alt+21] uses a sample $S_{C_1}, \ldots, S_{C_k}$ of the clusters to estimate the weights W. So we cannot use the trick of Algorithm 9 (ExactMemoization) that to update the silhouette we can add/subtract one distance, but we have to recompute the samples of the clusters involved in the point move.

The samples are build using a PPS (Probability Proportional to Size) sampling scheme [Ski16], which is resumed in Algorithm 3 (sample_pps). The resulting samples, of expected size $t = \Theta\left(\varepsilon^{-2}\log(nk/\delta)\right)$, contain the representatives of the cluster points. The cost of a clustering on the representatives has been shown to be a good approximation of the real cost of the clustering [CCK18], making the samples a *coreset* of the clustering. A coreset is in fact a subset of the original instance on which it is easy to obtain a good approximation of a measure on the whole original instance [Aga+05].

We could exploit the fact samples are coresets of the clusters to reduce the time complexity of our local search reducing the number of point relocations.

Algorithm 10: Optimize exact silhouette with coreset and memoization. (ExactCoresetMemo)

Input: clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$
Output: clustering $\mathcal{C}^\star = \{C_1^\star, \ldots, C_k^\star\}$, optimized for silhouette
let $V := \bigcup_{i=1}^{k} C_i$ be the set of all points
for $i \leftarrow 1$ **to** k **do**
  $S_{C_i} \leftarrow$ sample_pps(C_i) // Compute samples
  foreach $x \in V$ **do** $W_i(x) \leftarrow \sum_{y \in C_i} d(x, y)$
$\mathcal{C}^\star \leftarrow \mathcal{C}$, $W^\star \leftarrow W$
$s^\star \leftarrow$ sil_memo$(\mathcal{C}, W)$
do
  $s \leftarrow s^\star$
  $W' \leftarrow W$
  $i^\star \leftarrow j^\star \leftarrow 0$ // To save the clusters involved in the moving
  for $i \leftarrow 1$ **to** k **do**
    foreach $x \in S_{C_i}$ **do** // Move only points from samples
      $C_i' \leftarrow C_i \setminus \{x\}$
      foreach $y \in V$ **do** $W_i'(y) \mathrel{-}= d(x, y)$
      for $j \leftarrow 1$ **to** k, $j \neq i$ **do**
        $C_j' \leftarrow C_j \cup \{x\}$
        $\mathcal{C}' \leftarrow \mathcal{C} \setminus \{C_i, C_j\} \cup \{C_i', C_j'\}$
        foreach $y \in V$ **do** $W_j'(y) \mathrel{+}= d(x, y)$
        $s' \leftarrow$ sil_memo$(\mathcal{C}', W')$
        if $s' > s^\star$ **then**
          $\mathcal{C}^\star \leftarrow \mathcal{C}'$, $W^\star \leftarrow W'$
          $s^\star \leftarrow s'$
          $i^\star \leftarrow i$, $j^\star \leftarrow j$
        $W_j' \leftarrow W_j$
    $W_i' \leftarrow W_i$
  $\mathcal{C} \leftarrow \mathcal{C}^\star$, $W \leftarrow W^\star$
  if $i^\star \neq 0$ **then** // Update samples
    $S_{C_{i^\star}} \leftarrow$ sample_pps$(C_{i^\star})$
    $S_{C_{j^\star}} \leftarrow$ sample_pps$(C_{j^\star})$
while $s^\star - s > \eta$
return $\mathcal{C}^\star$

In Algorithm 10 (ExactCoresetMemo) only the points of the samples are moved. Obviously not all solutions are explored, but the exact silhouette is always computed. The intuition behind this approach is that the points in the coreset are the most influential on the silhouette (in fact they are used to approximate the weights of the clustering); so moving them we can somewhat maximize the silhouette improvement.

In blue are highlighted the differences with Algorithm 9 (ExactMemoization). The major difference is that only the points which are selected by the PPS sampling are moved, the other will remain where they are. Using the samples we are required to recompute them after changing the clusters; for this reason we have to recompute samples S_{C_i} and S_{C_j} after moving a point from C_i to C_j.

The computational complexity is

$$\begin{aligned}
&\sum_{i=1}^{k}\left(T_{sample_i} + O(n|C_i|)\right) + T_{sil} + \\
&\qquad + r\left(2T_{sample} + \sum_{i=1}^{k}|S_{C_i}|\left(O(n) + \sum_{j=0,j\neq i}^{k}\left(O(n) + T_{sil}\right)\right)\right) = \\
&= O\left(n(n + \log(nk/\delta))\right) + O(nk) + \\
&\qquad + r\left(n\log(nk/\delta) + \sum_{i=1}^{k}|S_{C_i}|\left(O(n) + O\left(nk\right) + O\left(nk^2\right)\right)\right) = \\
&= O\left(n^2 + rnk^3\varepsilon^{-2}\log(nk/\delta)\right) = O\left(n^2 + rnk^3 t\right)
\end{aligned}$$

where

- $T_{sil} = O(nk)$
- $T_{sample_i} = O\left(|C_i|\log(nk/\delta)\right) = O\left(n\log(nk/\delta)\right)$
- $|S_{C_i}| = O\left(\varepsilon^{-2}\log(nk/\delta)\right) = O(t)$

The memory required for the memoization is $O(nk)$.

In the Algorithm 10 (ExactCoresetMemo) the initial exact silhouette computation dominates the complexity, for this in Algorithm 11 (ApproxCoresetMemo) we use approximated weights in the initial phase. The initial silhouette is the same approximated one proposed in [Alt+21]. Then, since the approximated weights are an unbiased estimator of the real weights, we can treat them as the real weights and simply sum/subtract one distance as in previous algorithms, without recompute the samples. It would be indeed an error to sum/subtract the distance divided by

Algorithm 11: Optimize approximated silhouette with coreset and memoization. (ApproxCoresetMemo)

Input: clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$
Output: clustering $\mathcal{C}^\star = \{C_1^\star, \ldots, C_k^\star\}$, optimized for silhouette
let $V \coloneqq \bigcup_{i=1}^{k} C_i$ be the set of all points
for $i \leftarrow 1$ **to** k **do**
  $S_{C_i} \leftarrow$ sample_pps(C_i)
  foreach $x \in V$ **do** $\hat{W}_i(x) \leftarrow \sum_{y \in S_{C_i}} \frac{d(x,y)}{p_y}$ // Approximated weights
$\mathcal{C}^\star \leftarrow \mathcal{C}$, $\hat{W}^\star \leftarrow \hat{W}$
$\hat{s}^\star \leftarrow$ sil_memo$(\mathcal{C}, \hat{W})$ // Approximated silhouette
do
  $\hat{s} \leftarrow \hat{s}^\star$
  $\hat{W}' \leftarrow \hat{W}$
  $i^\star \leftarrow j^\star \leftarrow 0$
  for $i \leftarrow 1$ **to** k **do**
    foreach $x \in S_{C_i}$ **do**
      $C_i' \leftarrow C_i \setminus \{x\}$
      foreach $y \in V$ **do** $\hat{W}_i'(y) \mathrel{-}= d(x,y)$
      for $j \leftarrow 1$ **to** k, $j \neq i$ **do**
        $C_j' \leftarrow C_j \cup \{x\}$
        $\mathcal{C}' \leftarrow \mathcal{C} \setminus \{C_i, C_j\} \cup \{C_i', C_j'\}$
        foreach $y \in V$ **do** $\hat{W}_j'(y) \mathrel{+}= d(x,y)$
        $\hat{s}' \leftarrow$ sil_memo$(\mathcal{C}', \hat{W}')$
        if $\hat{s}' > \hat{s}^\star$ **and** $\hat{s}' > \hat{s} + \bar{\varepsilon}$ **then** // Take care of approximation error
          $\mathcal{C}^\star \leftarrow \mathcal{C}'$, $\hat{W}^\star \leftarrow \hat{W}'$
          $\hat{s}^\star \leftarrow \hat{s}'$
          $i^\star \leftarrow i$, $j^\star \leftarrow j$
        $\hat{W}_j' \leftarrow \hat{W}_j$
    $\hat{W}_i' \leftarrow \hat{W}_i$
  $\mathcal{C} \leftarrow \mathcal{C}^\star$, $\hat{W} \leftarrow \hat{W}^\star$
  if $i^\star \neq 0$ **then**
    $S_{C_{i^\star}} \leftarrow$ sample_pps$(C_{i^\star})$
    $S_{C_{j^\star}} \leftarrow$ sample_pps$(C_{j^\star})$
while $\hat{s}^\star - \hat{s} > \eta + \bar{\varepsilon}$ // Take care of approximation error
return $\mathcal{C}^\star$

its related probability: in this way also the points represented by the moved point would be moved.

The computational complexity is

$$\begin{aligned}
&\sum_{i=1}^{k}\left(T_{sample_i}+O(n|S_{C_i}|)\right)+T_{sil}+\\
&\qquad\qquad +r\left(2T_{sample}+\sum_{i=1}^{k}|S_{C_i}|\left(O(n)+\sum_{j=0,j\neq i}^{k}\left(O(n)+T_{sil}\right)\right)\right)=\\
&=O\left(n\log(nk/\delta)(1+k\varepsilon^{-2})\right)+O(nk)+\\
&\qquad\qquad +r\left(n\log(nk/\delta)+\sum_{i=1}^{k}|S_{C_i}|\left(O(n)+O\left(nk\right)+O\left(nk^2\right)\right)\right)=\\
&=O\left(rnk^3\varepsilon^{-2}\log(nk/\delta)\right)=O\left(rnk^3t\right)
\end{aligned}$$

where

- $T_{sil}=O(nk)$
- $T_{sample_i}=O\left(|C_i|\log(nk/\delta)\right)=O\left(n\log(nk/\delta)\right)$
- $|S_{C_i}|=O\left(\varepsilon^{-2}\log(nk/\delta)\right)=O(t)$

The memory required for the memoization is $O(nk)$.

A common approach to take advantage by the coresets in problems which are solved by expensive algorithms, is to run the algorithm on the coreset, and then extend the obtained result to the whole instance. In our case we can run the expensive local search of Algorithm 9 (ExactMemoization) on the small coreset of the cluster samples, and then assign the points not in the coreset to the nearest cluster. This is what Algorithm 12 (SampleOptimization) does. We first compute all cluster PPS samples, then apply Algorithm 9 (ExactMemoization) to our coresets. We thus obtain a clustering of the coreset, optimized for its silhouette, which should be a good representative of an optimization on the entire clustering. Finally, we assign the remaining points to the nearest cluster in a way similar to the second part of Lloyd's algorithm: we find for each point the cluster which minimizes the mean distance with the point (the equivalent of finding the nearest center in Lloyd's) and then, at the end, assign each point to its chosen cluster.

Algorithm 12: Optimize on the sample. (SampleOptimization)

Input: clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$
Output: clustering $\mathcal{C}^\star = \{C_1^\star, \ldots, C_k^\star\}$, optimized for silhouette
for $i \leftarrow 1$ **to** k **do**
  $S_{C_i} \leftarrow$ sample_pps(C_i) // Compute samples
let $S_\mathcal{C} \coloneqq \{S_{C_1}, \ldots, S_{C_k}\}$
$S_\mathcal{C}^\star \leftarrow$ exact_memoization($S_\mathcal{C}$) // Apply Algorithm 9 (ExactMemoization) to the sample
let $\bar{S} \coloneqq \bigcup_{i=1}^k C_i \setminus S_{C_i}$ be the set of not sampled points
foreach $x \in \bar{S}$ **do** $i_x \leftarrow \text{argmin}_i \left\{ \sum_{y \in S_{C_i}^\star} \frac{d(x,y)}{|S_{C_i}^\star|} \right\}$ // Find the cluster which minimizes the mean distance
foreach $x \in \bar{S}$ **do** $S_{C_{i_x}}^\star \leftarrow S_{C_{i_x}}^\star \cup \{x\}$ // Assign not sampled points to the nearest cluster
return $S_\mathcal{C}^\star$

The computational complexity is

$$\begin{gathered}
\sum_{i=1}^{k} T_{sample_i} + T_{\text{exact_memoization}} + |\bar{S}| T_{argmin} + |\bar{S}| = \\
= O(n \log(nk/\delta)) + O(rm^2k^2) + O(nm - m^2) + O(n - m) = \\
= O(n \log(nk/\delta)) + O(rm^2k^2) + O(nm) + O(n) = \\
= O(nm + rm^2k^2) = O(nkt + rk^4t^2)
\end{gathered}$$

where

- $T_{sample_i} = O\left(|C_i| \log(nk/\delta)\right)$
- $|S_{C_i}| = O\left(\varepsilon^{-2} \log(nk/\delta)\right) = O(t)$
- $m = \sum_{i=1}^k |S_{C_i}| = O\left(k\varepsilon^{-2} \log(nk/\delta)\right) = O\left(kt\right)$
- $|\bar{S}| = n - m$
- $T_{argmin} = \sum_{i=1}^k |S_{C_i}^\star| = \sum_{i=1}^k |S_{C_i}| = m$
- $T_{\text{exact\_memoization}} = O(rm^2k^2)$

The memory required for the memoization is $O(mk) = O\left(k^2\varepsilon^{-2} \log(nk/\delta)\right) = O\left(k^2 t\right)$.

As we can see we have a great improvement on the time complexity, and also, for the first time, in the memory required for the memoization.

Chapter 4

Experimental Evaluation

To validate the proposed approach, a suite of experiments has been performed. A first set of experiments (illustrated in Section 4.1) was directed to discover the differences, in performances, among the proposed local search algorithms, reported in Table 4.1. Since Algorithm 9 (ExactMemoization) returns the same solution of Algorithm 8 (Naïve), but, thanks to memoization, requiring less distance computations, the latter algorithm was not tested. Its results would be indeed the same of the former for what concerns silhouette, but a lot worse in time. Then another set of experiments (discussed in Section 4.2) has been executed to compare the best proposed local search algorithm (which resulted to be Algorithm 12, SampleOptimization) with the most common algorithms for clustering, namely, k-means++, Lloyd's algorithm and PAM k-medois. After that, in Section 4.3, we compared two variants of Algorithm 12 (SampleOptimization): the standard one, which uses the PPS sample of [Alt+21], and an alternative which uses a sample extracted through Poisson Sampling with uniform probability. Finally, in Section 4.4, we conclude the evaluation with an experiment directed to understand if a second pass of Algorithm 12 (SampleOptimization) could refine the result of the first execution.

Algorithm 9	ExactMemoization
Algorithm 10	ExactCoresetMemo
Algorithm 11	ApproxCoresetMemo
Algorithm 12	SampleOptimization

Table 4.1: Proposed algorithms used in the experiments.

For the experiments both real and artificial datasets were used. They are reported in Table 4.2. Real case datasets were taken from UCI Machine Learning Repository [DG17].

Dataset *Radius* is the combination of *Radius Queries* and *Radius Queries Count* (without the count component), from the *Query Analytics Workloads* dataset[1][SAT18; AST18]. It contains 60 000 (1 000 in the truncated version used in Section 4.1) circles represented by triplets composed by the bi-dimensional coordinate of the center, and the radius. The circles are taken from Gaussian distributions over a real dataset.

From dataset *Selfback*[2] [San+16] we took 1 000 points of participant number 26. The data is collected by two accelerometers positioned on the wrist and on the thigh of the participant, during exercise activities. The points contain the three-dimensional coordinates of both accelerometers.

Dataset *Cloud*[3] is made of 1024 vectors of parameters obtained from AVHRR images of clouds (raging both visible and infrared spectrum).

The artificial *Gaussian* dataset contains 10 000 points on a bi-dimensional plane. The points are generated according to a Gaussian distribution around 5 centers. There is also an 1% of noisy points uniformly distributed.

The *Higgs*[4] [BSW14] dataset is the result of Monte Carlo simulations of signal processes producing Higgs bosons. Each point is composed by 21 measurements and 7 summary features, but we used a reduced version with only 10 000 points containing the 7 summary attributes. For this dataset the Manhattan distance was used, and so Lloyd's algorithm was replaced by PAM k-medoids [KR90].

Name	Real / Artificial	number of points (n)	dimension of points
Radius	Synthesized from Real	1 000 (truncated)	3
Selfback	Real	1 000 (truncated)	6
Cloud	Synthesized from Real	1 024	10
Radius	Synthesized from Real	60 000	3
Gaussian	Artificial	10 000	2
Higgs	Artificial	10 000 (truncated)	7

Table 4.2: Datasets used in the experiments. The first three are used in Section 4.1, the last three in other experiments.

For each experiment k-means++ was initially run to obtain a starting clustering from the list of points. Then, from this clustering, the proposed algorithms and Lloyd's were run. All the experiments use Euclidean distances, except for dataset

[1] `https://archive.ics.uci.edu/ml/datasets/Query+Analytics+Workloads+Dataset`
[2] `https://archive.ics.uci.edu/ml/datasets/selfBACK`
[3] `https://archive.ics.uci.edu/ml/datasets/cloud`
[4] `https://archive.ics.uci.edu/ml/datasets/HIGGS`

Higgs for which Manhattan distance was used.[5] Recall that Euclidean distance is $d_2(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$ and Manhattan distance is $d_1(x, y) = \sum_i |x_i - y_i|$.

The data collected in the experiments are the execution time, the iterations performed by the local-search algorithms (r) and the final exact silhouette computed on the whole clustering. This silhouette is the optimized metric only for ExactMemoization and ExactCoresetMemo: Lloyd's optimizes the sum of squared distances, PAM the sum of distances, ApproxCoresetMemo the approximated silhouette (on the whole clustering) and SampleOptimization the exact silhouette on the sample. The final clustering for SampleOptimization is composed, as described in Section 3.3, assigning each point not in the sample to the closest cluster.

The tables in the following sections also report the improvements in the silhouette obtained by the various algorithms with respect to the one obtained by k-means++ and k-means++ combined with Lloyd (i.e. the difference between current algorithm silhouette and reference silhouette), and the speedup with respect to Lloyd's execution time (i.e. the ratio between the time required by Lloyd's algorithm and that required by the current algorithm). In all the executions the probability δ was set to 0.1 and the threshold η to 0.001; the other parameters are reported in the performances tables: the number of clusters k, and the sample expected size t.

All the experiments have been run single-threaded on a machine with *Intel Core i5-4210U* @ 1.70 GHz and 8 GiB of RAM running *GNU/Linux NixOS* 22.11 with kernel *Linux* 5.15.67 `x86_64` and *OpenJDK* `17.0.4+8-nixos`.

4.1 Differences among proposed local search algorithms

As said, in this section we will compare the local search algorithms proposed in Chapter 3 to understand their optimization power and to assess their time performance. The Tables 4.3 to 4.5 report the results of these experiments. For each algorithm we can see the silhouette of the obtained clustering, the number of local-search iterations performed and the required time.

A first observation that derives from the results reported in Tables 4.3 to 4.5, is that generally all the proposed algorithms achieve an improvement in silhouette on the clustering returned by k-means++; further evidence of this fact will be provided in Section 4.2.

Among the local search algorithms we can easily see how SampleOptimization returns the best improvement in silhouette and takes the less time to achieve it. For what concerns the running time, it is clearly the winner, since it takes

[5]With *Higgs* dataset PAM replaced Lloyd's since the latter is suited only for the standard Euclidean distance.

Optimization algorithm	Exact silhouette	r	Time [ms]	Improves k-means++	Improves Lloyd's	Speedups Lloyd's
k-means++	0.32208	–	63	–	–	–
Lloyd's k-means	0.36942	30	118	0.04734	–	–
ExactMemoization	0.32506	3	4773	0.00298	-0.04436	0.02472
ExactCoresetMemo	0.3225	1	158	0.00042	-0.04692	0.74684
ApproxCoresetMemo	0.3219	1	91	-0.00018	-0.04752	1.2967
SampleOptimization	0.36153	2	16	0.03945	-0.00789	7.375

Table 4.3: Comparison of the proposed algorithms on dataset *Radius*, using $k = 5$ and $t = 5$.

Optimization algorithm	Exact silhouette	r	Time [ms]	Improves k-means++	Improves Lloyd's	Speedups Lloyd's
k-means++	0.1765	–	19	–	–	–
Lloyd's k-means	0.4651	28	41	0.2886	–	–
ExactMemoization	0.26931	49	182110	0.09281	-0.19579	0.00023
ExactCoresetMemo	0.24814	34	7107	0.07164	-0.21696	0.00577
ApproxCoresetMemo	0.2425	40	8778	0.066	-0.2226	0.00467
SampleOptimization	0.56987	40	1090	0.39337	0.10477	0.03761

Table 4.4: Comparison of the proposed algorithms on dataset *Selfback*, using $k = 9$ and $t = 5$.

Optimization algorithm	Exact silhouette	r	Time [ms]	Improves k-means++	Improves Lloyd's	Speedups Lloyd's
k-means++	0.31047	–	23	–	–	–
Lloyd's k-means	0.44944	13	22	0.13897	–	–
ExactMemoization	0.35513	8	35652	0.04466	-0.09431	0.00062
ExactCoresetMemo	0.35264	6	1621	0.04217	-0.0968	0.01357
ApproxCoresetMemo	0.32285	14	4434	0.01238	-0.12659	0.00496
SampleOptimization	0.41822	15	378	0.10775	-0.03122	0.0582

Table 4.5: Comparison of the proposed algorithms on dataset *Cloud*, using $k = 10$ and $t = 5$.

advantage of executing the heaviest part only on a small sample with fixed expected size $O(kt)$. The real surprising result is instead on the silhouette of the returned clustering. It is indeed comparable to the one of the clustering returned by Lloyd's, and always higher than those of the clustering returned by the others local search algorithms. While the worse approximation provided by ExactCoresetMemo and

ApproxCoresetMemo can be justified by the fact that these algorithms can swap only points of the samples, the surprisingly good results provided by SampleOptimization, even superior to those of ExactMemoization, which works on the entire datasets, were somewhat unexpected. In fact, this can be explained by observing that each point of the sample is a representative of more points, and so moving a single point in the sample means moving more points in the complete clustering. This permits our algorithm to converge faster.

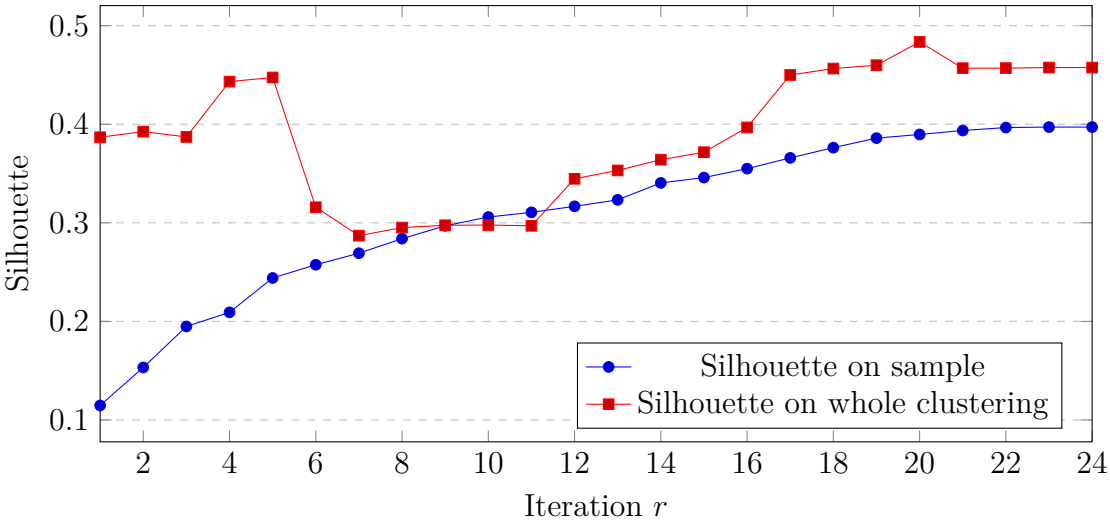

Figure 4.1: Evolution of the silhouettes on dataset *Selfback* with SampleOptimization.

Another interesting aspect can be seen in Fig. 4.1: the plot reports the silhouette computed on the sample and on the complete set of points[6] at the end of each iteration of SampleOptimization. As can be seen an improvement of the silhouette on the sample does not always correspond to an improvement on the silhouette on the entire clustering. This non-monotonicity property can be potentially due to a sort of simulating annealing effect, in the sense that SampleOptimization is able to skip some local maxima by letting some iterations to temporarily worsen the silhouette value. As it can be seen, in facts after the 5^{th} iteration we have a significant lowering of the final silhouette, but then it restarts to grow reaching higher values (with the 20^{th} iteration) than before.

[6]The final step of assigning each point outside the sample to the nearest cluster was done at each iteration in order to compute the silhouette on the final clustering.

4.2 Performance comparison of SampleOptimization with k-means++, Lloyd's algorithm and PAM

Now we have assessed SampleOptimization to be the best among the proposed local search strategies, we will analyze another batch of experiments directed to compare this algorithm to the known ones. The results of these experiments are reported in Tables 4.6 to 4.8, where values are medians over 5 runs (3 in the experiments with *Higgs* dataset). This was not needed in the previous analysis since the results were sufficiently well separated to conclude the difference in performance among the algorithms was not imputable to the noise. Instead in this case, where the results were closer each other, we opted in for a more robust analysis.

The datasets used in this phase are *Radius*, *Gaussian* and *Higgs* (with Manhattan distance), which were described in the beginning of this chapter. These datasets contain more points of those used in Section 4.1: n is one order of magnitude bigger. Even *Radius* was used as whole and not truncated as before.

Optimization algorithm	Exact silhouette	r	Time [ms]	Improves k-means++	Improves Lloyd's	Speedups Lloyd's
k-means++	0.32778	–	660	–	–	–
Lloyd's k-means	0.36126	63	2171	0.03348	–	–
SampleOpt $t = 5$	0.35114	19	978	0.02336	-0.01012	2.21984
SampleOpt $t = 10$	0.35607	29	3818	0.02829	-0.00519	0.56862
SampleOpt $t = 15$	0.36642	43	12699	0.03864	0.00516	0.17096
SampleOpt $t = 20$	0.35679	34	10975	0.02901	-0.00447	0.19781
SampleOpt $t = 25$	0.34816	19	12908	0.02038	-0.0131	0.16819
SampleOpt $t = 50$	0.34286	17	52026	0.01508	-0.0184	0.04173
SampleOpt $t = 100$	0.34277	4	35796	0.01499	-0.01849	0.06065

Table 4.6: Median performances on dataset *Radius*, using $k = 10$.

The first observation we can make from the results presented in Tables 4.6 to 4.8 is that, as in the previous set of experiments, SampleOptimization always obtains an improvement with respect to k-means++. Moreover, it is comparable with Lloyd's algorithm: they achieve somewhat similar silhouette values. The time required by these two last algorithms is more variable, because of a high variance of the number of iterations as we will discuss later on; however, with the largest dataset *Radius*, SampleOptimization achieved 2 times speed up on Lloyd's. It is important to remark that from the time complexity analysis done in Chapter 3, as SampleOptimization should perform a lot better than this with big datasets where $n \gg k^4$. This because in that case the number of iterations r will not be the dominant component of the time complexity $O(nkt + rk^4t^2)$. Another great

advantage of SampleOptimization on Lloyd's algorithm is that it does not require the selection of the centers, not being a center-based algorithm.

Optimization algorithm	Exact silhouette	r	Time [ms]	Improves k-means++	Improves Lloyd's	Speedups Lloyd's
k-means++	0.36528	–	105	–	–	–
Lloyd's k-means	0.68246	28	158	0.31718	–	–
SampleOpt $t = 5$	0.78467	20	675	0.41939	0.10221	0.23407
SampleOpt $t = 10$	0.80074	61	6998	0.43546	0.11828	0.02258
SampleOpt $t = 15$	0.78929	67	17608	0.42401	0.10683	0.00897
SampleOpt $t = 20$	0.78784	81	37964	0.42256	0.10538	0.00416
SampleOpt $t = 25$	0.7863	82	51952	0.42102	0.10384	0.00304
SampleOpt $t = 50$	0.64578	106	240210	0.2805	-0.03668	0.00066
SampleOpt $t = 100$	0.58311	99	717661	0.21783	-0.09935	0.00022

Table 4.7: Median performances on dataset *Gaussian*, using $k = 10$.

For what concerns the expected sample size, we can conclude that it is convenient to use small values of t. Besides the evident benefit in time given by the smaller sample, we usually obtain similar silhouette results to those given by higher values of t, and sometimes even higher. This last astonishing outcome can have the same reason of the fact SampleOptimization outperforms ExactMemoization (see Section 4.1).

Another interesting aspect could be seen in Fig. 4.2: there is a point (respectively $t = 80$, 75 and 50 for the three datasets) where the number of iterations executed by our algorithm drops. These values of t, for which a local step improving more than the threshold is harder to achieve, are such $1/tk \approx \eta$. Knowing the silhouette of the sample to be the mean of the silhouettes of its points (which are $O(tk)$), we can in fact suppose the possible improvement achievable in a step to be $O(1/tk)$. Indeed, from the data collected in the experiments, it can be observed how the mean step improvement (i.e. the silhouette improvement w.r.t. k-means++, divided by the number of iterations) is $O(1/tk)$.

With dataset *Gaussian* we have a great improvement on both k-means++ and Lloyd's. With $t = 10$ we reached even the silhouette of 0.80 (aided by the dataset which has natural clusters and very low noise).

A further interesting detail is the variance of the collected data: for each experiment and for each algorithm (i.e. a row in the previous tables) the variance of the silhouette is very small, instead the variance of the number of iterations r (and so the time) is very high. This means that the returned clustering should

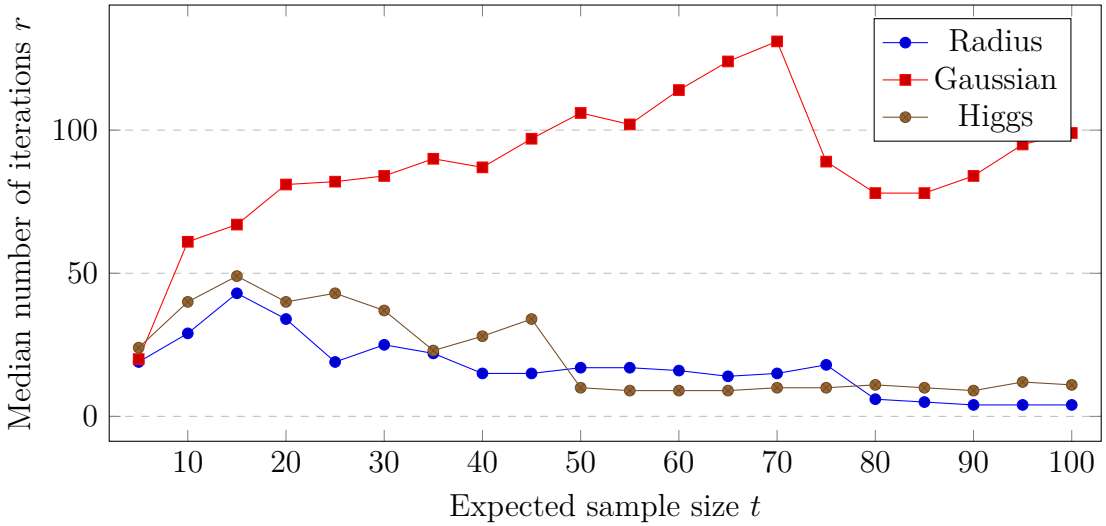

Figure 4.2: Dependence from t of the number of iterations r with SampleOptimization.

quite always be good, whilst it could be more probable the case of an unluckily slow run (requiring more iterations).

4.2.1 Using non-Euclidean distance functions

Optimization algorithm	Exact silhouette	r	Time [ms]	Improves k-means++	Improves PAM	Speedups PAM
k-means++	0.06879	–	151	–	–	–
PAM k-medoids	0.13114	18	12638785	0.06235	–	–
SampleOpt $t = 5$	0.12375	25	824	0.05496	-0.00739	15338.33131
SampleOpt $t = 10$	0.15331	45	4829	0.08452	0.02217	2617.26755
SampleOpt $t = 15$	0.22956	89	13224	0.16077	0.09842	955.74599
SampleOpt $t = 20$	0.1156	31	11553	0.04681	-0.01554	1093.98295
SampleOpt $t = 25$	0.31333	104	56215	0.24454	0.18219	224.8294
SampleOpt $t = 50$	0.08153	10	21440	0.01274	-0.04961	589.49557
SampleOpt $t = 100$	0.08753	9	73723	0.01874	-0.04361	171.43612

Table 4.8: Median performances on dataset *Higgs* with Manhattan distance, using $k = 10$.

One known Lloyd's algorithm weakness is that it works well only with Euclidean distances [SYR13]. Our proposed local search algorithms work with arbitrary

distance functions. To explore this scenario we can see the results in Table 4.8 where Manhattan distance was used instead of Euclidean one. Since Lloyd's is not indicated in this situation, the algorithm was replaced with PAM *(Partitioning Around Medoids)*: a popular approximation algorithm for solving the k-medoids problem.

In this case, k-means++ returns a poorly optimized clustering with silhouette values in the order of 10^{-2}. It can be seen how our proposed SampleOptimization can achieve a one order of magnitude step in silhouette improvement. This is an important observation because it tells us the algorithm is able not only to fine-tune already good solutions, but also to obtain a good solution from a not so good one.

SampleOptimization enhances the silhouette of the clustering returned by k-means++ only slightly less than PAM, except for $t = 10$, 15 and 25 where there is an improvement. In any case the speedup is really significative: PAM has a $O(rn^2k^2)$ time complexity which could not compete with the one of SampleOptimization. As said in Section 2.3, there are some enhancements of PAM, such as FastPAM which has complexity $O(rn^2)$ [SR21], but losing only a factor k^2 our proposed algorithm remains very competitive.

It is good to remark how our local search differs from PAM not only by the objective function (as does for example PAMSIL [LPB03], already cited in Section 1.3). As a matter of fact the swaps are completely different: in our strategy we swap a point between two clusters, while in PAM the swap is between a medoid and a non-medoid. Consequently, our strategy does not require the computation of centers. This indeed is a great advantage which, for example, allows us to apply the memoization trick exposed in Section 3.2, which saves a lot of computations. This memoization could not be applied to PAMSIL because the used swap technique could possibly move a lot of points to another cluster and so the weights must be recomputed.

Moreover, we adopted a coreset technique, on the top of the local search, which brings a great enhancement of performances both in quality and in execution time. This sampling methodology offers a well representative coreset, while the simplified silhouette adopted in PAMMEDSIL lacks representativeness.

4.3 Using a different sampling technique

A legitimate question would be if we really need a PPS sample to build the coreset for SampleOptimization, or instead a simpler uniform sample would suffice. To answer this question we executed a third batch of experiments running two variants of SampleOptimization: the standard variant based on the PPS sampling technique by [Alt+21], and a variant where PPS sampling is replaced by the simpler Poisson

sampling, where each point x from a cluster C_i is included in the sample with uniform probability $t/|C_i|$ independently of the other points.

The results are presented in Figs. 4.3 to 4.5. In the plots we can see, for each value of t, the (exact) silhouette on the sample, the exact silhouette on the whole clustering and the required number of iterations. In blue with the original PPS sampling, in red with the uniform sample.

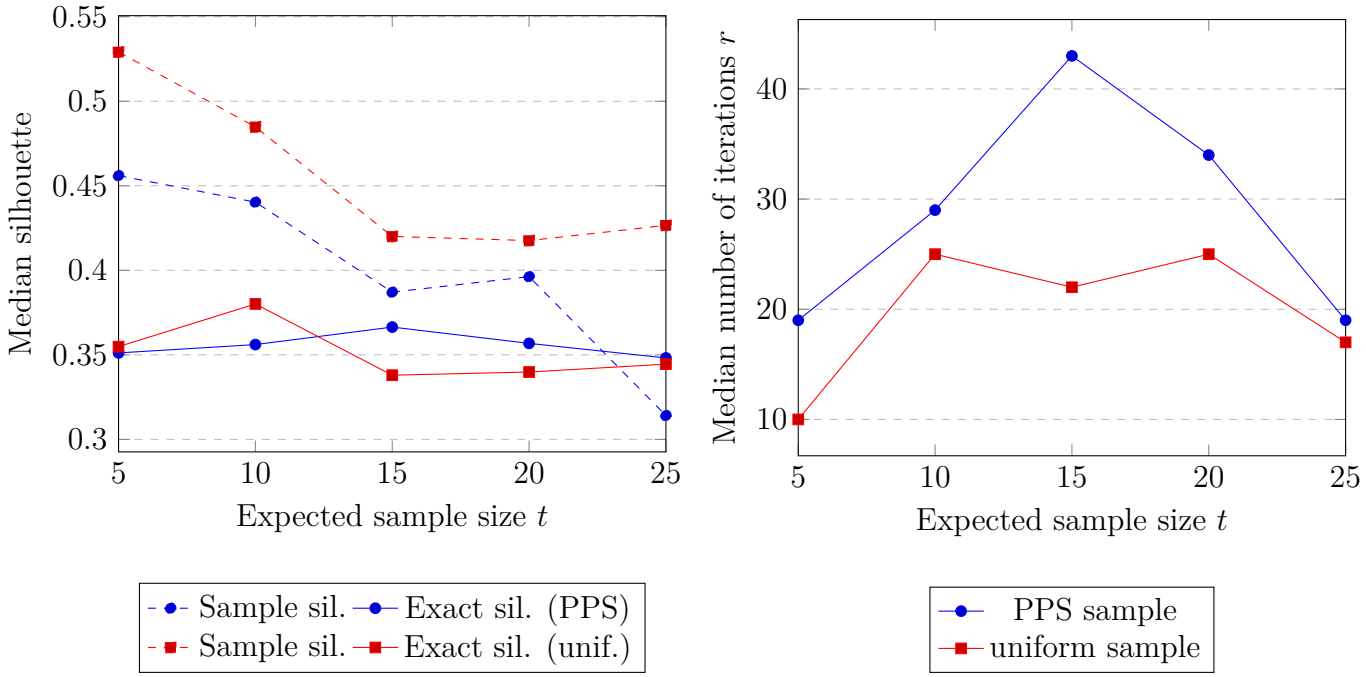

Figure 4.3: Median performances on dataset *Radius* with two sampling techniques.

We can easily notice how using a uniform sampling technique the number of iterations performed (and so the execution time) is lower than using a PPS sample. The difference is notably remarked for dataset *Gaussian*. This could be somewhat expected since a uniform sample has the effect of regularize the dataset. This regularized, and less representative, sample offers fewer possibilities of improvement since all the states of our local search tends to be similar, leading to early stops.

The greater representativeness of the PPS sample can be seen in the silhouette values of the experiments with *Radius* and *Higgs* datasets. In fact, we can notice how the silhouette on the sample is closer to the silhouette on the whole clustering using a PPS sample: the silhouette using a uniform sample has a greater variance if computed on the sample or on the complete dataset. Indeed, even if using a uniform sample usually brings to a better improvement of the silhouette internally

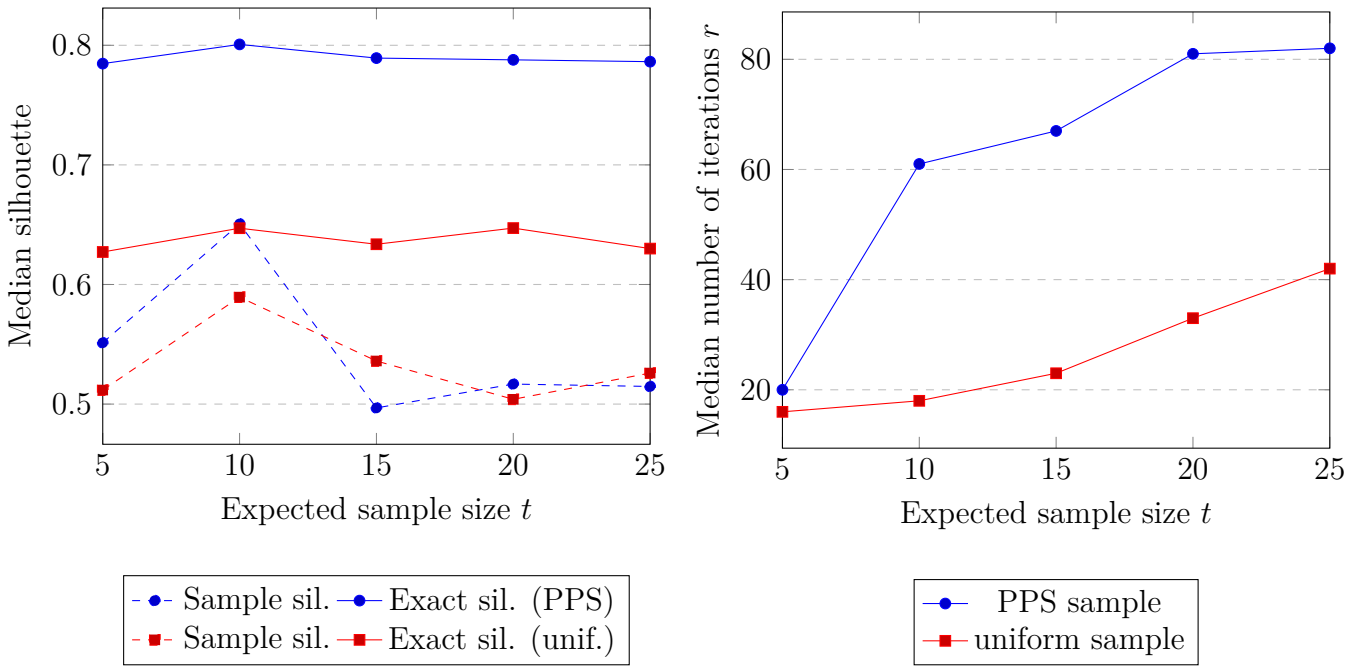

Figure 4.4: Median performances on dataset *Gaussian* with two sampling techniques.

used by the local search, this improvement does not correspond to a better final silhouette. As a matter of fact the PPS sample offers the best final silhouette. The only black sheep is the dataset *Radius*, for which the two sampling strategies offer a similar final result. This could be due to the fact that for this particular dataset a uniform sample would be enough representative.

4.4 Refinement with two-pass

Since the sample loses its representativeness going on with iterations, we devised an improved strategy based on running SampleOptimization twice and recomputing a new sample before the second run. More precisely, in a first pass, the algorithm is run to provide a clustering for the whole dataset, which, based on the experimental results of the previous subsections, is already rather good. Then, in the second pass, a new sample is computed for this clustering and the algorithm is run again using this new sample, which will hopefully refine the quality of the clustering. This technique could also reduce the probability of ending up in local maximum, since it restarts the search from a very different point.

In Table 4.9 we can see the results after the first pass and after the second pass

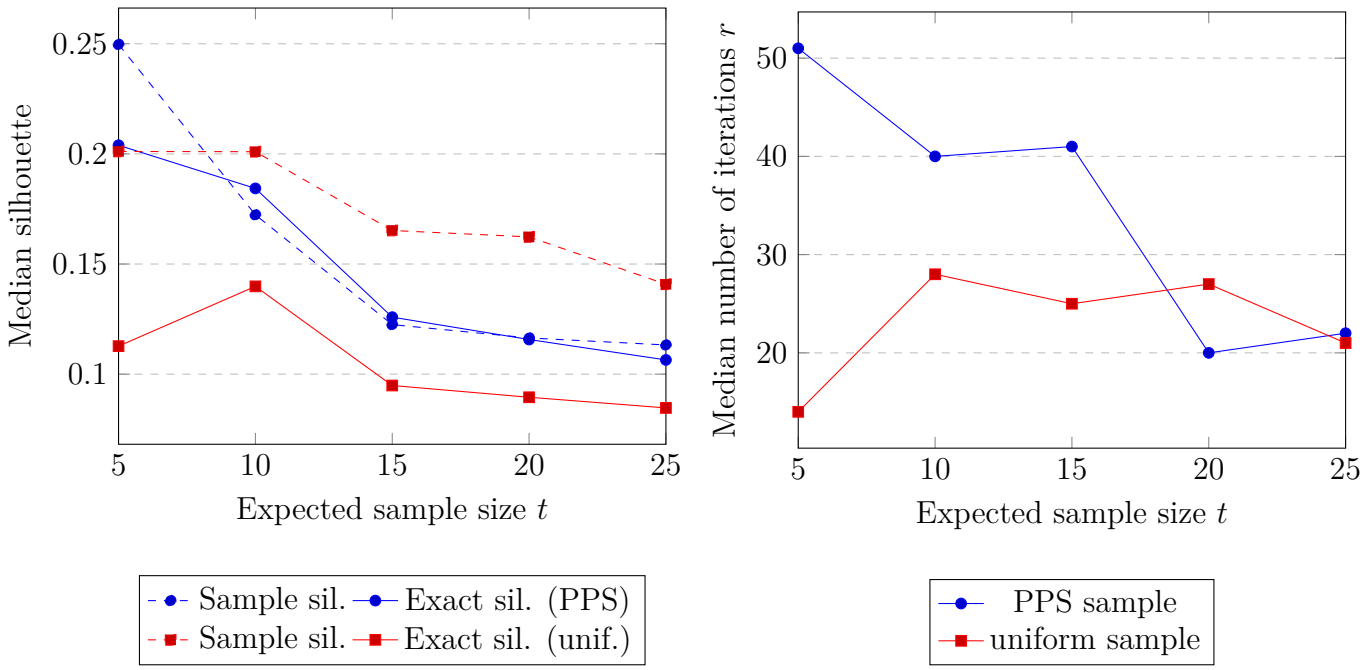

Figure 4.5: Median performances on dataset *Higgs* with two sampling techniques.

Optimization algorithm		Exact silhouette	r	Time [ms]
k-means++		0.32778	-	651
Lloyd's k-means		0.36126	63	1958
SampleOpt $t = 5$	first run	0.35114	19	925
	refinement	0.35897	9	444
SampleOpt $t = 10$	first run	0.35607	29	3107
	refinement	0.3633	14	1788
SampleOpt $t = 50$	first run	0.34286	17	42011
	refinement	0.35368	3	7875
SampleOpt $t = 100$	first run	0.34277	4	31511
	refinement	0.34257	4	33872

Table 4.9: Median performances on dataset *Radius* with two-pass, using $k = 10$.

of SampleOptimization. As it can be seen, even if the second pass usually improves the silhouette, the improvement is not significative and so the refinement does not seem to bring a substantial practical benefit.

Chapter 5

Conclusions

In this thesis we developed a local search algorithm for solving the clustering problem optimizing the silhouette. The local step is moving a point from a cluster to another. Moreover, we enhanced a straightforward implementation of the algorithm by exploiting memoization and combined it with the sampling strategy from [Alt+21] obtaining the coreset-based algorithm, dubbed SampleOptimization.

An extensive experimental analysis provided evidence SampleOptimization can be competitive with the standard Lloyd's algorithm for k-means, in the sense that our algorithm returns a clustering whose silhouette is comparable with the one of the clustering returned by Lloyd's algorithm, sometimes also in less time. When more time was required the resulting silhouette was indeed higher. From the theoretical time complexity analysis we can also notice how the speed-up would be higher for largest datasets where $n \gg k^3t^2$, since SampleOptimization affords greater scalability over Lloyd's approach. In addition, it can be notice how SampleOptimization would be easily distributable. The distributability of the first part, where the clusters are sampled, was already shown in [Alt+21]. Also, the last part of the algorithm is easily parallelizable being formed of loops having independent iterations.

Another strong feature of SampleOptimization is that it can be used also with non-Euclidean distances, for which Lloyd's algorithm is either not suited or not at all applicable (due to its use of centroids). For example, with Manhattan distances we saw how SampleOptimization outperforms the most popular algorithm for k-median, PAM, taking merely about $1/16\,000$ of the time. Moreover, SampleOptimization is likely to outperform also the most famous improvements of PAM: FastPAM and FasterPAM.

We wish to point out that our approach can also be employed for non-center based clusterings. This allowed us to define the local step in a different way from PAM-based algorithms aiming at optimizing the silhouette (e.g. PAMSIL), which

yielded us to better optimizations.

With another set of experiments, we provided evidence that the sampling technique plays an important role in the goodness of the final result. In fact our SampleOptimization highly benefits from the representativeness of the PPS sample adopted, with respect to a simpler uniform sample.

5.1 Future works

As always, there is still room for improvement. For example, a method to find at runtime the best value for t (the expected sample size in SampleOptimization) could be devised. In fact, we saw from the experiments that increasing the sample size raises its representativeness, but with too many points the local search cannot make any significative improvement, and gets stuck at local maxima, or does not fulfill the threshold of minimal improvement.

The experiments suggest that the silhouette of the clustering returned by SampleOptimization, expressed as a function of the expected sample size is a curve initially increasing, and then, after a global maximum, decreasing. If this is true and if the silhouette of the sample is indeed a good representative of the silhouette on the whole clustering[1], then a simple method to pick t could be applied, namely by running the local search on the sample for small values of t and then picking the result which maximizes the silhouette, to be passed to the last part of the algorithm.

Another interesting aspect to study would be the adjustment of the threshold η to the sample size to avoid early saturations.

Moreover, while the experiments in Section 4.4 indicate that repeating SampleOptimization twice with the same expected sample size t does not yield significant improvements, we could test the benefits of a second run of SampleOptimization with a different value of t. A smaller value of t could avoid some local maxima and permit a faster move across states, whilst a greater t (combined with a smaller η) would increase the possible routes towards the optimum, allowing ending up in better states.

We can also take inspiration from the optimizations done in PAM, and how they were adapted in [LS22]. In fact, we can save, at the beginning of each local search iteration, for each point, the three closest clusters. Sure enough one of the three cluster will be involved in the computation of $b(x)$. Then, in the computation of the memoized silhouette, to obtain $b(x)$ we have to take the minimum among the weights with those three clusters and the two clusters involved in the moving of the point[2],

[1] Actually, it is sufficient that there is a monotonicity relation between the two silhouettes. From our experiments we can say this could quite be possible.

[2] Excluded the cluster of x in the case we are computing $b(x)$ for the moved point.

instead of comparing the weights of all clusters. This, for large values of $k \gg 5$, would reduce of a factor $O(k)$ both the computation of the memoized silhouette and of ExactMemoization, bringing the complexity of SampleOptimization from $O(nkt + rk^4t^2)$ to $O(nkt + rk^3t^2)$. We really need to save the three nearest clusters because the two clusters involved in the moving of the point could possibly be the two closest. In this case, where the weights of these two clusters have changed, we could be sure the distances with the other clusters (which did not change weights) to be greater only of the third nearest.

References

[AB84] Mark S. Aldenderfer and R. Blashfield. *Cluster analysis*. Sage, 1984. ISBN: 0803923767.

[Aga+05] Pankaj K. Agarwal, Sariel Har-peled, Kasturi, and R. Varadarajan. "Geometric approximation via coresets". In: *Combinatorial and computational geometry* 52.1 (2005). URL: http://sarielhp.org/p/04/survey/.

[Alt+21] Federico Altieri, Andrea Pietracaprina, Geppino Pucci, and Fabio Vandin. "Scalable distributed approximation of internal measures for clustering evaluation". In: *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*. SIAM. 2021, pp. 648–656. DOI: 10.1137/1.9781611976700.73.

[AR14] Charu C. Aggarwal and Chandan K. Reddy. "Data clustering". In: *Algorithms and applications. Chapman&Hall/CRC Data mining and Knowledge Discovery series, Londra* (2014).

[AST18] Christos Anagnostopoulos, Fotis Savva, and Peter Triantafillou. "Scalable aggregation predictive analytics". In: *Applied Intelligence* 48.9 (2018), pp. 2546–2567.

[AT07] S. Aranganayagi and K. Thangavel. "Clustering Categorical Data Using Silhouette Coefficient as a Relocating Measure". In: *International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007)*. Vol. 2. 2007, pp. 13–17. DOI: 10.1109/ICCIMA.2007.328.

[BSW14] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. "Searching for exotic particles in high-energy physics with deep learning". In: *Nature communications* 5.1 (2014), pp. 1–9.

[CCK18] Edith Cohen, Shiri Chechik, and Haim Kaplan. "Clustering Small Samples With Quality Guarantees: Adaptivity With One2all PPS". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1 (Apr. 2018). DOI: 10.1609/aaai.v32i1.11772.

[DG17] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[FMS07] Dan Feldman, Morteza Monemizadeh, and Christian Sohler. "A PTAS for K-Means Clustering Based on Weak Coresets". In: *Proceedings of the Twenty-Third Annual Symposium on Computational Geometry*. SCG '07. Gyeongju, South Korea: Association for Computing Machinery, 2007, pp. 11–18. ISBN: 9781595937056. DOI: 10.1145/1247069.1247072.

[FS06] Gereon Frahling and Christian Sohler. "A Fast K-Means Implementation Using Coresets". In: *Proceedings of the Twenty-Second Annual Symposium on Computational Geometry*. SCG '06. Sedona, Arizona, USA: Association for Computing Machinery, 2006, pp. 135–143. ISBN: 1595933409. DOI: 10.1145/1137856.1137879.

[Gha+22] Kareem Kamal A. Ghany, Amr Mohamed AbdelAziz, Taysir Hassan A. Soliman, and Adel Abu El-Magd Sewisy. "A hybrid modified step Whale Optimization Algorithm with Tabu Search for data clustering". In: *Journal of King Saud University - Computer and Information Sciences* 34.3 (2022), pp. 832–839. ISSN: 1319-1578. DOI: 10.1016/j.jksuci.2020.01.015.

[HK05] Sariel Har-Peled and Akash Kushal. "Smaller Coresets for K-Median and k-Means Clustering". In: *Proceedings of the Twenty-First Annual Symposium on Computational Geometry*. SCG '05. Pisa, Italy: Association for Computing Machinery, 2005, pp. 126–134. ISBN: 1581139918. DOI: 10.1145/1064092.1064114.

[HM04] Sariel Har-Peled and Soham Mazumdar. "On Coresets for K-Means and k-Median Clustering". In: *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*. STOC '04. Chicago, IL, USA: Association for Computing Machinery, 2004, pp. 291–300. ISBN: 1581138520. DOI: 10.1145/1007352.1007400.

[HPK11] J. Han, J. Pei, and M. Kamber. *Data mining: concepts and techniques*. 2011.

[HWS09] S. He, Q. H. Wu, and J. R. Saunders. "Group Search Optimizer: An Optimization Algorithm Inspired by Animal Searching Behavior". In: *IEEE Transactions on Evolutionary Computation* 13.5 (2009), pp. 973–990. DOI: 10.1109/TEVC.2009.2011992.

[Kan+04] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. "A local search approximation algorithm for k-means clustering". In: *Computational Geometry* 28.2 (2004). Special Issue on the 18th Annual Symposium on Computational Geometry - SoCG2002, pp. 89–112. ISSN: 0925-7721. DOI: 10.1016/j.comgeo.2004.03.003.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by simulated annealing". In: *Science* 220.4598 (1983), pp. 671–680. DOI: 10.1126/science.220.4598.671.

[KR90] L. Kaufman and P.J. Rousseeuw. "Partitioning Around Medoids (Program PAM)". In: *Finding Groups in Data*. John Wiley & Sons, Ltd, 1990. Chap. 2, pp. 68–125. ISBN: 9780470316801. DOI: 10.1002/9780470316801.ch2.

[Lle+04] R. Lletí, M.C. Ortiz, L.A. Sarabia, and M.S. Sánchez. "Selecting variables for k-means cluster analysis by using a genetic algorithm that optimises the silhouettes". In: *Analytica Chimica Acta* 515.1 (2004), pp. 87–100. ISSN: 0003-2670. DOI: 10.1016/j.aca.2003.12.020.

[Llo82] S. Lloyd. "Least squares quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137. DOI: 10.1109/TIT.1982.1056489.

[LMS10] Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. "Iterated Local Search: Framework and Applications". In: *Handbook of Metaheuristics*. Ed. by Michel Gendreau and Jean-Yves Potvin. Boston, MA: Springer US, 2010, pp. 363–397. ISBN: 978-1-4419-1665-5. DOI: 10.1007/978-1-4419-1665-5_12.

[LPB03] Mark Van der Laan, Katherine Pollard, and Jennifer Bryan. "A new partitioning around medoids algorithm". In: *Journal of Statistical Computation and Simulation* 73.8 (2003), pp. 575–584. DOI: 10.1080/0094965031000136012.

[LRB21] Attila Lengyel, David W. Roberts, and Zoltán Botta-Dukát. "Comparison of silhouette-based reallocation methods for vegetation classification". In: *Journal of Vegetation Science* 32.1 (2021), e12984. DOI: 10.1111/jvs.12984.

[LS22] Lars Lenssen and Erich Schubert. "Clustering by Direct Optimization of the Medoid Silhouette". In: *Similarity Search and Applications*. Ed. by Tomáš Skopal, Fabrizio Falchi, Jakub Lokoč, Maria Luisa Sapino, Ilaria Bartolini, and Marco Patella. Cham: Springer International Publishing, 2022, pp. 190–204. DOI: 10.1007/978-3-031-17849-8_15.

[Mic68] Donald Michie. ""Memo" Functions and Machine Learning". In: *Nature* 218.5136 (Apr. 1968), pp. 19–22. ISSN: 1476-4687. DOI: 10.1038/218019a0.

[MPP19] Alessio Mazzetto, Andrea Pietracaprina, and Geppino Pucci. "Accurate MapReduce Algorithms for k-median and k-means in General Metric Spaces". In: *30th International Symposium on Algorithms and Computation (ISAAC 2019)*. Ed. by Pinyan Lu and Guochuan Zhang. Vol. 149. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 34:1–34:16. ISBN: 978-3-95977-130-6. DOI: 10.4230/LIPIcs.ISAAC.2019.34.

[Nie16] Frank Nielsen. "Hierarchical Clustering". In: Feb. 2016, pp. 195–211. ISBN: 978-3-319-21902-8. DOI: 10.1007/978-3-319-21903-5_8.

[NNA20] N. Nidheesh, K. A. Abdul Nazeer, and P. M. Ameer. "A Hierarchical Clustering algorithm based on Silhouette Index for cancer subtype discovery from genomic data". In: *Neural Computing and Applications* 32.15 (Aug. 2020), pp. 11459–11476. ISSN: 1433-3058. DOI: 10.1007/s00521-019-04636-5.

[RN10] Stuart Russel and Peter Norvig. "Beyond Classical Search". In: *Artificial Intelligence. A Modern Approach*. 3rd ed. Prentice Hall, 2010. Chap. 4.

[Rob15] David W. Roberts. "Vegetation classification by two new iterative reallocation optimization algorithms". In: *Plant Ecology* 216.5 (May 2015), pp. 741–758. ISSN: 1573-5052. DOI: 10.1007/s11258-014-0403-2.

[Rou87] Peter J. Rousseeuw. "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis". In: *Journal of Computational and Applied Mathematics* 20 (1987), pp. 53–65. ISSN: 0377-0427. DOI: 10.1016/0377-0427(87)90125-7.

[San+16] Sadiq Sani, Nirmalie Wiratunga, Stewart Massie, and Kay Cooper. "SELFBACK—activity recognition for self-management of low back pain". In: *International Conference on Innovative Techniques and Applications of Artificial Intelligence*. Springer. 2016, pp. 281–294.

[SAT18] Fotis Savva, Christos Anagnostopoulos, and Peter Triantafillou. "Explaining aggregates for exploratory analytics". In: *2018 IEEE International Conference on Big Data (Big Data)*. IEEE. 2018, pp. 478–487.

[SG06] Bart Selman and Carla P. Gomes. "Hill-climbing search". In: *Encyclopedia of cognitive science* 81 (2006), p. 82.

[SI84] Shokri Z. Selim and M. A. Ismail. "K-Means-Type Algorithms: A Generalized Convergence Theorem and Characterization of Local Optimality". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-6.1 (1984), pp. 81–87. DOI: 10.1109/TPAMI.1984.4767478.

[SJB17] Kim Sung-Soo, Baek Jun-Young, and Kang Bum-Soo. "Group Search Optimization Data Clustering Using Silhouette". In: *Journal of the Korean Operations Research and Management Science Society* 42.3 (Aug. 2017), pp. 25–34.

[Ski16] Chris J. Skinner. "Probability Proportional to Size (PPS) Sampling". In: *Wiley StatsRef: Statistics Reference Online*. John Wiley & Sons, Ltd, 2016, pp. 1–5. ISBN: 9781118445112. DOI: 10.1002/9781118445112.stat03346.pub2.

[SR21] Erich Schubert and Peter J. Rousseeuw. "Fast and eager k-medoids clustering: O(k) runtime improvement of the PAM, CLARA, and CLARANS algorithms". In: *Information Systems* 101 (2021), p. 101804. ISSN: 0306-4379. DOI: 10.1016/j.is.2021.101804.

[SYR13] Archana Singh, Avantika Yadav, and Ajay Rana. "K-means with Three different Distance Metrics". In: *International Journal of Computer Applications* 67.10 (2013).

[TSK16] P. N. Tan, M. Steinbach, and V. Kumar. *Introduction to data mining*. 2016.

[VA06] Sergei Vassilvitskii and David Arthur. "k-means++: The advantages of careful seeding". In: *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. 2006, pp. 1027–1035. URL: http://ilpubs.stanford.edu:8090/778/.

[WH07] Gregory A. Wilkin and Xiuzhen Huang. "K-Means Clustering Algorithms: Implementation and Comparison". In: *Second International Multi-Symposiums on Computer and Computational Sciences (IMSCCS 2007)*. 2007, pp. 133–136. DOI: 10.1109/IMSCCS.2007.51.