Systematically improving existing $k$-means initialization algorithms at nearly no cost, by pairwise-nearest-neighbor smoothing

Carlo Baldassi$^1$:*

$^1$Department of Computing Sciences, Bocconi University, via Sarfatti 25, 20136 Milan, Italy

We present a meta-method for initializing (seeding) the $k$-means clustering algorithm called PNN-smoothing. It consists in splitting a given dataset into $J$ random subsets, clustering each of them individually, and merging the resulting clusterings with the pairwise-nearest-neighbor (PNN) method. It is a meta-method in the sense that when clustering the individual subsets any seeding algorithm can be used. If the computational complexity of that seeding algorithm is linear in the size of the data $N$ and the number of clusters $k$, PNN-smoothing is also almost linear with an appropriate choice of $J$, and in fact only at most a few percent slower in most cases in practice. We show empirically, using several existing seeding methods and testing on several synthetic and real datasets, that this procedure results in systematically better costs. It can even be applied recursively, and easily parallelized. Our implementation is publicly available at https://github.com/carlobaldassi/KMeansPNNSmoothing.jl.

I. INTRODUCTION

The classical $k$-means algorithm is one of the most well-known and widely adopted clustering algorithms [1, 2]. Given $N$ data points $\mathcal{X} = (x_i)_{i=1..N}$, where each point is $D$-dimensional, $x_i \in \mathbb{R}^D$, and given an integer $k \geq 2$, the algorithm aims at minimizing the sum-of-squared-errors (SSE) cost, defined as a function of $k$ centroids $\mathcal{C} = (c_a)_{a=1..k} \in (\mathbb{R}^D)^k$, and of a partition of the data points $\mathcal{P} = (p_i)_{i=1..N} \in \{1, \ldots, k\}^N$, as such:

$$\text{SSE}(\mathcal{C}, \mathcal{P}; \mathcal{X}) = \sum_{i=1}^{N} \| x_i - c_{p_i} \|^2$$

For fixed $\mathcal{C}$, the optimal partition is obtained by associating each point to its nearest centroid, and conversely, for fixed $\mathcal{P}$, the optimal centroids are given by the barycenter of each cluster. The $k$-means algorithm starts from an initial guess for the configuration and alternates optimizing $\mathcal{C}$ and $\mathcal{P}$ until a fixed point is reached. This alternating procedure, due to Lloyd [3], is greedy and converges to a local minimum. Its computational cost is $O(kND)$ per iteration; both optimization steps can be straightforwardly parallelized over $N$. It is notoriously sensitive to the choice of the initial configuration, i.e. the seeding, both in terms of the final value of the SSE and of the number of Lloyd’s iterations required to converge. Several schemes have been proposed, with various degrees of complexity. An extremely basic and cheap option is to sample the initial centroids uniformly at random from $\mathcal{X}$ [4]. Other popular methods typically produce considerable improvement and also scale like $O(kND)$, e.g. $k$-means++ [5] and maxmin [6, 7], among several others. Yet other methods have worse scalings and thus tend to dominate the computational time, e.g. the pairwise-nearest-neighbors (PNN) method [8] which scales like $\Omega(N^2)$.

In this paper, we propose a novel scheme, PNN-smoothing, based on randomly splitting the dataset $\mathcal{X}$ into $J$ subsets, clustering them individually with $k$-means, and then merging the resulting $Jk$ clusters following the PNN procedure until only $k$ clusters remain: these constitute the new seed for Lloyd’s algorithm. It is a meta-method, in the sense that it can use any seeding procedure for the subsets. We denote this with $\text{PNNs(INIT)}$ where INIT is any seeding algorithm. If INIT is $O(kN)$, then $\text{PNNs(INIT)}$ is also almost linear, as long as we set $J = O\left(\sqrt{N/k}\right)$. Our empirical tests indicate that $\text{PNNs(INIT)}$ gives systematically better SSEs compared to INIT, at a very small computational cost, even in a parallel implementation. Its results are surprisingly good even when INIT is one of the worst seeding methods, random uniform initialization. Also, since $\text{PNNs(INIT)}$ is itself a seeding algorithm, it can even be applied recursively, e.g. $\text{PNNs(PNNs(INIT))}$, to any desired depth.

Throughout the paper, we focus exclusively on the effect of seeding on basic $k$-means, which can be regarded as a basic tool in the optimization of the SSE cost. The rest of the paper is organized as follows. In sec. II we review prior literature and describe a few seeding methods that will be considered in the tests. In sec. III we describe and discuss in detail the $\text{PNNs(INIT)}$ scheme. In sec. IV we present and analyze detailed numerical results on several challenging synthetic and non-synthetic datasets. Sec. V has a final discussion.

II. RELATION TO PRIOR WORKS

As mentioned in the introduction, a large number of seeding schemes for $k$-means have been proposed. Extensive reviews and benchmarks can be found in refs. [9, 10]. Here, we only cover a few selected ones.

UNIF. Uniformly sampling $k$ centroids from the...
dataset [4] is arguably the most popular method. We’ll call this seeding method UNIF. It’s computationally very cheap, but its performance is generally very poor, even in moderately hard circumstances, as it often leads to poor local minima and long convergence times. Note that, in our definition, although the cost of sampling the centroids is \( O(k) \), the overall cost of UNIF is still \( O(kN^D) \) since we include the computation of the corresponding optimal partition, which is half of a Lloyd’s iteration. This choice is justified by the fact that the partition is necessary to compute the SSE, and other seeding algorithms include this computation as a byproduct; this allows to compare all seeding algorithms consistently.1 Note also that we’ll assume that the sampling is done without replacement.

**MAXMIN.** Another simple method [6, 7] goes under the name of “furthest point”, or “maxmin”, or “maximin”. We will refer to the same variant that was used in [9, 10], and call it MAXMIN. It consists in selecting the first centroid at random from the dataset, after which the process is iterative and deterministic: at each step, each successive centroid is chosen as the furthest point from the centroids selected so far. More precisely, it’s the point that maximizes the distance from its nearest centroid: 

\[
c_a = \arg\max_{x \in X} \min_{b < a} \| x - c_b \|^2.
\]

This algorithm scales as \( O(kN^D) \); it also computes the optimal partition (with respect to the chosen centroids) as a byproduct of the selection procedure. The results of ref. [10], on synthetic datasets, report this method as being among the best of those that were tested. On the other hand, in ref. [9], in which an array of real datasets was also tested, the authors advise against this method. They suggest instead, among the algorithms in the same complexity class, to use greedy-\(k\text{-}means++\) or Bradley and Fayyad’s “refine” (both described below).

**[g]KM++.** The “\(k\text{-}means++\)” seeding method [5] can be regarded as a more stochastic version of MAXMIN. While the first centroid is also selected at random from the dataset, the remaining centroids are sampled from the dataset with a probability proportional the squared distance from the closest centroids. More precisely, the probability of selecting a point \( x \) as the next centroid \( c_a \) when \( a \geq 2 \) is 

\[
P(c_a = x \mid (c_b)_{b < a}) \propto \min_{b < a} \| x - c_b \|^2.
\]

The procedure can be further extended in a greedy manner: at each step \( a \geq 2 \), \( s \) candidates are sampled according to the previous probability distribution, the new SSE (with \( a \) clusters) is computed, and the candidate with the lowest SSE is chosen. The number of candidates per step \( s \) is usually logarithmic in the number of clusters \( k \); in all our tests, we have used \( s = \lfloor 2 + \log k \rfloor \).2 We refer to the original variant as KM++ and to the greedy one as GKM++. The computational complexity of KM++ is \( O(kN^D) \), like MAXMIN, while GKM++ scales like \( O(kN\text{log}k) \). For both, the optimal partition is also computed as a byproduct, like for MAXMIN.

In ref. [9] GKM++ was reported as superior to KM++ and overall as one among the best linear (in \( N \)) stochastic methods; in ref. [10] the results of KM++ were considered comparable to or slightly worse than MAXMIN; however, GKM++ was not tested.

**REF(INIT).** Bradley and Fayyad’s “refine” seeding algorithm [13] is the one that most resembles our proposed scheme. Indeed, the initial step of the two methods is basically the same, i.e. it consists in splitting the dataset into \( J \) random subsets and clustering them individually with \( k\)-means, thus obtaining \( J \) groups of \( k \) centroids. The crucial difference relies in the way in which these \( J \) solutions are merged, which in ref. [13] is referred to as a “smoothing” procedure: in the refine method, the whole pool of \( Jk \) centroids is used as a new dataset and clustered for \( J \) times. Each time, one of the previous centroid configurations is used as seed for the Lloyd algorithm. Out of the resulting \( J \) configurations, the one with the smallest cost (computed on the pooled dataset) is finally chosen. Originally, the authors used UNIF as the seeding method for clustering the subsets, but this can be trivially generalized to other methods. We thus consider it a meta-method like PNN-smoothing, and denote it with REF(INIT) where INIT is the seeding method for the initial step. If the computational cost of INIT is linear, \( O(kN^D) \), then REF(INIT) scales as \( O((kN + Jk^2)^D) \). The algorithm requires \( J \leq N/k \) in order to perform the first step, and thus REF(INIT) is always at most \( O(kN^D) \). In the original publication, ref. [13], only REF(UNIF) with \( J = 10 \) was tested. This was also the setting used in refs. [9, 10]. We also used the same value of \( J \) in our tests (the only exception being a dataset for which \( N/k = 10 \), in which case we used \( J = 5 \)), but we tested more INIT algorithms. As mentioned above, in ref. [9] REF(UNIF) was found to be among the best methods, whereas in ref. [10] it was shown to perform rather poorly on synthetic datasets.

**PNN.** Our method starts out identically to refine, but it uses the PNN procedure to merge the resulting \( J \) clusterings. This procedure was originally introduced in ref. [8] as a seeding algorithm. The original algorithm, which we call PNN, is hierarchical. It starts with \( N \) clusters, one per data point, and then merges pairs of clusters iteratively until only \( k \) clusters remain. Merging two clusters means that the partition \( P \) is updated by substituting the two clusters with their union. The centroids set \( C \) is also updated, by substituting the two starting centroids \( c_a \) and \( c_b \) with the centroid of the new cluster, \( c_{\text{new}} = (z_a c_a + z_b c_b) / (z_a + z_b) \), where \( z_a \) and \( z_b \) are the number of elements in each of the two original clusters. The algorithm is deterministic and greedy: the two clusters to be merged at each step are those whose merging will result in the smallest increase in the SSE cost. It is easy to see from eq. 1 that the cost increment of merging two clusters of sizes \( z_a \) and \( z_b \) and with centroids \( c_a \) and \( c_b \) is 

\[
\Delta_{ab} = z_a z_b \| c_a - c_b \|^2 / (z_a + z_b).
\]
Thus, thanks to the fact that the SSE uses the squared Euclidean distance, both the pairwise merging costs and the new centroid can be computed using only the centroids and the cluster sizes. The initial computation of the new centroid can be computed using only the Euclidean distance, both the pairwise merging costs and thus, thanks to the fact that the seeding scheme and discuss its properties.

The computational complexity of a merging step, assuming that we are going from \( k + 1 \) clusters to \( k \) clusters, would be \( O(Dk^2) \) if performed straightforwardly, due to the need to update the \( \Delta_{ab} \) after each merge. However, in ref. [14] it was shown that a significant speedup can be obtained by considering that most cluster pairs are unaffected by individual merges, and the complexity can be reduced to \( O(Dk\tau_k) \) where \( \tau_k \) is essentially the number of (potentially) affected clusters and is generally much smaller than \( \hat{k} \). Furthermore, the residual full-update operations can be straightforwardly parallelized over \( \hat{k} \), which proves advantageous above a certain threshold. In the original PNN algorithm the merging step must be performed \( N-k \) times with \( \hat{k} \) ranging from \( N-1 \) to \( k \). Thus, the overall computational complexity is \( \Omega(DN^2) \). This is quite expensive for large datasets. On the other hand, the results in terms of the SSE objective are generally very good.

The PNN scheme was also used in the genetic algorithm of ref. [15], where however it was used as a crossover step rather than a seeding procedure. In that algorithm, two given configurations with \( k \) clusters each are first merged into a single configuration with \( 2k \) clusters, which is then used as the starting point for the PNN iterative merging, until \( k \) clusters remain. The resulting cost is \( O(Dk^2\tau_k) \), which (crucially) does not involve a factor \( N \) thanks to the fact that only centroid computations are involved in the merge. Our seeding scheme is similar to this, in the sense that we also use the iterative PNN merging procedure with an initial number of clusters much smaller than \( N \).

### III. THE PNN-SMOOTHING SCHEME

In this section we describe in detail the \textsc{pnns}(init) seeding scheme and discuss its properties.

The inputs of the procedure are the same as for any other algorithm (the dataset \( \mathcal{X} \) and the number of clusters \( k \)), plus the subset-seeding algorithm \textsc{init}, and one extra parameter \( \rho \), used to determine the number of subsets \( J \), for which we use a default value of 0.5. The output is a configuration to be used as a starting point for local optimization. The high-level summary (see also the illustration of each step in fig. 1) is as follows:

**Inputs:** \( \mathcal{X}, \ k, \ \text{init}, \ \rho \quad [=0.5] \)

1. Set \( J = \left\lceil \sqrt{\rho N/k} \right\rceil \). Cap the result at \( \lfloor N/k \rfloor \).

2. Split \( \mathcal{X} \) into \( J \) random subsets \( \left( \tilde{X}_a \right)_{a=1...J} \).

3. Cluster each \( \tilde{X}_a \) independently, using \text{init} for seeding followed by Lloyd’s algorithm; obtain \( J \) configurations \( \left( \tilde{C}_a, \tilde{P}_a \right) \), with \( k \) clusters each.

4. Collect the \( J \) configurations \( \left( \tilde{C}_a, \tilde{P}_a \right) \) into a single configuration \( (C_0, P_0) \) for the entire \( \mathcal{X} \), with \( k \) centroids and clusters.

5. Merge the clusters of \((C_0, P_0)\), two at a time, using the PNN procedure, until \( k \) clusters remain; obtain a set of \( k \) centroids \( C \).

6. Compute the optimal partition \( P \) associated to \( C \).

**Output:** \((C, P)\)

Next, we describe and discuss in more detail each step.

1. The parameterization in terms of \( \rho \) rather than \( J \) ensures that the asymptotic behavior of the algorithm is almost linear in \( N \) and \( k \), as discussed in the introduction (cf. point 5 below). The upper bound to \( J \) is necessary for the following steps 2 and 3. The effective range of \( \rho \), assuming for simplicity that \( N \) is divisible by \( k \), is \([k/N, N/k] \). The lower bound leads to \( J = 1 \) and \textsc{pnns}(\text{init}) = \text{init}.

2. The upper bound corresponds to \( J = N/k \) and \textsc{pnns}(\text{init}) = \text{PNN} (under the assumption that \text{init} invoked on \( \hat{k} \) points will assign each one to its own cluster, which is the case in all our tests, including \text{unif} since we perform sampling without replacement). In other words, by changing \( \rho \) we can interpolate between any seeding algorithm \text{init} and the pairwise-nearest-neighbor algorithm. This explains why, as a rule of thumb, increasing \( \rho \) improves quality at the cost of performance, although this is not strictly true in all cases.

The allowed range for \( \rho \) is quite wide under normal circumstances, in which \( k \) is much smaller than \( N \). Throughout our tests, we used the value \( \rho = 0.5 \), which seems to result in a good trade-off in all cases and for all \text{init} algorithms, without requiring any fine-tuning. Some results on this are shown in Appendix A. This is a valid (i.e. non-degenerate) choice whenever \( k < N/2 \), which is arguably always the case in realistic scenarios.

2. We perform the splitting by assigning each point of the dataset to a subset uniformly at random, but under the global constraint that each subset has at least \( k \) points. This is easy to implement efficiently by just creating a list of length \( N \) in which the first \( kJ \) entries are repetitions of \( 1...J \) for \( k \) times, followed in the remaining \( N-kJ \) entries by random uniform integers extracted from \( 1...J \); the list is then randomly shuffled to obtain the final subset assignments.

3. The computational cost of each individual clustering of one of the subsets depends on the choice of
Figure 1. Example of pnns(init) in action. Here $D = 2$, $N = 72$, $k = 4$ and init = unif. The numbering of the steps follows the text (step 1 simply yields $J = 3$). The crosses represent centroids. After step 3 the subsets clusterings contain some clear mistakes, but after the merge (4) and the PNN procedure (5) the remaining 4 centroids are close to the correct positions. After step 6 the seeding procedure is completed; the final local optimization solves the problem.

4. In order to obtain the new configuration, we just take the union of the centroids, i.e. $C_0 = \bigcup_a \tilde{C}_a$; the partition $P_0$ would also be simply the union of the $\left(\tilde{P}_a\right)_a$ with remapped indices, but since it is not even needed for the algorithm (only the cluster sizes are used) it can be skipped. It is also interesting to note that the new configuration will, in general, be nowhere near optimal for a problem with $kJ$ clusters, since points near each other will likely be assigned to clusters coming from different subsets. Nevertheless, at least under some favorable scenarios, we can expect that the centroids in $C_0$ may themselves be approximately clustered into $k$ groups (see fig. 1). This is the same intuition at the root of the refine method. The question then becomes how to best find a consensus configuration among the $J$ different results.
5. The PNN procedure will start, as the name implies, by merging the closest centroids (accounting for their associated cluster size). If the centroids in \( C_0 \) are mostly clustered already, the procedure will likely pick first the centroids that appeared in multiple subset clusterings. Each time two centroids are merged, their associated size (and thus weight) increases, such that even if in the last stages some very sub-optimal partitions are merged with a large cluster, the centroid will be heavily skewed toward the latter.

The computational cost of the merging scales like \( O\left(D(Jk)^2\tau_{jk}\right) = O(DN\kappa\tau_{jk}) \), as per the analysis of the PNN procedure of the previous section. This is indeed a consequence of our choice for the scaling of \( J \). The additional factor \( \tau_{jk} \) is hard to estimate; it is bounded by \( Jk = \sqrt{\rho NK} \) but in practice it appears to be quite small. This is the step that dominates the computational complexity of the whole algorithm.

6. At the end of the procedure the partition is recomputed. This is done to compute the SSE and provide a consistent starting point for Lloyd’s algorithm (cf. the note about UNIF in the previous section), but it also means that outlier points in the partitions that could be (virtually) produced during the PNN merge are eliminated (like the blue points in the bottom-right cluster and the green points in the top-left cluster in fig. 1).

As mentioned in the introduction, the algorithm can be applied recursively. We use the notation PNN\(^n\)(init) to denote the \( n \)-th level recursive algorithm, e.g. PNN\(^3\)(init) = PNN(PNN(PNN(init))).

We have also explored a fully-recursive scheme in which we keep splitting the dataset until \( N \leq \kappa k \), in which case we use PNN. We call this algorithm PNNSR. By the master theorem, its computational complexity in terms of the input size \( N \) is bound between \( O(N\log N) \) in the best-case scenario \( \tau_k = O(1) \), and \( O(N^{3/2}) \) in the worst-case scenario \( \tau_k = O\left(\frac{1}{\kappa}\right) \).

### IV. NUMERICAL EXPERIMENTS

#### A. Experimental setup

We performed a series of tests comparing PNNs with all the algorithms mentioned in sec. II, namely: UNIF, MAXMIN, KM++, GKM++, PNN, and REF(init) with INIT = \{UNIF, MAXMIN, KM++, GKM++\}. For PNNs, we tested PNN\(^n\)(init) with INIT = \{UNIF, MAXMIN, KM++, GKM++\} and \( n \in \{1, 2, 3\} \); plus we tested the fully-recursive PNNSR. We used the same data structures and programming language (Julia v1.6.2) for all of them. For the local optimization part, i.e. Lloyd’s algorithm, we used the technique introduced in ref. [16], which is quite effective in saving some distance computations in the partition update after the first iteration, at the cost of keeping track of which centroids remain unchanged between one iteration and the next. This implies that the overall number of Lloyd’s iterations is not proportional to the running time, since iterations become cheaper as the clustering stabilizes. As a criterion for detecting convergence of Lloyd’s algorithm we checked whether the SSE ceased to improve within a relative tolerance of \( 10^{-5} \). Our code is available at ref. [17].

All the timings that we report refer to tests performed on the same hardware (Intel Core i7-9750H 2.60GHz CPU with 6 physical cores, 64Gb DDR4 2666MHz RAM, running Ubuntu Linux 18.04 with 5.3.0 kernel) with no other computationally intensive processes running while testing. All codes were carefully optimized and can run in parallel with multi-threading; most of our results are shown for the single-threaded case except where otherwise noted.

We also report average numbers of Lloyd’s iterations. In order to make comparisons as uniform as possible, when computing the total number of iterations of REF(init) and PNNs(init) we include the ones spent in the seeding stage, but we reweight them by a factor \( 1/J \), since each of them operates on a subset of the data. The reweighting is applied recursively for PNNSR and for PNN\(^n\)(init) with \( n \geq 2 \).

We tested a number of synthetic and real-world datasets whose characteristics are shown in table I. The synthetic datasets are mainly intended to measure the ability of the algorithms to find the solution when one can be clearly identified, and for direct comparison with the results of ref. [10].

#### B. Synthetic datasets

In ref. [10], several synthetic datasets with different characteristics were chosen and tested in order to probe the strengths and weaknesses of several seeding algorithms with respect to properties of the data. We picked 5 of the most challenging ones, all obtained from ref. [18]: A3,

| dataset  | \( D \) | \( N \) | \( k \) |
|----------|------|------|------|
| USCensus | 68   | 2458285 | 100 |
| House    | 3    | 34112  | 256  |
| Bridge   | 16   | 4096   | 256  |
| Miss America | 16 | 6480  | 256  |
| UrbanGB  | 2    | 360177 | 469  |
| Olivetti | 4096 | 400    | 40   |
| Isolalet | 617  | 7792   | 26   |

| name     | \( D \) | \( N \) | \( k \) |
|----------|------|------|------|
| A3       | 2    | 7500 | 50   |
| Birch1   | 2    | 100000  | 100  |
| Birch2   | 2    | 100000  | 100  |
| Unbalance | 2    | 6500  | 8    |
| Dim1024  | 1024 | 1024  | 16   |


Overall, from our results it emerges that the

PNNs(INIT) scheme is able to improve the success rate of any INI algorithm (even of \text{INIT}=\text{PNNs(INIT)}) with at most a modest time penalty, sometimes even a time gain.

Birch1, Birch2, Unbalance and Dim1024 (see Table I). We scaled each dataset uniformly in order to make them span the range \( [0,1] \). The difficulty for A3, Birch1 and Birch2 is in their relatively large size and abundance of local minima; for Unbalance, it’s the fact that some clusters are small and far from the bigger ones; for Dim1024 it’s the large dimensionality. All algorithms tested in [10] showed poor results in at least some of these datasets; in particular, the authors report a 0% success rate (as defined below) on Birch1 for all algorithms.

All of these datasets were generated from isotropic Gaussians centered around ground-truth centroids, and for all of them the global optimum of the SSE is very close to the ground truth. Under these circumstances, it is reasonable to classify the local minima configurations that the algorithms produce by their “centroid index” (CI), as defined in ref. [19]. The CI is computed by matching each centroid of a clustering with its closest one from the ground truth, and counting the number of unmatched ground truth centroids. It can be interpreted as the number of mistakes in the resulting clustering, and therefore we define the success rate of an algorithm as the frequency with which it finds a solution, i.e., a configuration with CI = 0.

We present our most representative results in Table II; the complete results are reported in Appendix B.1. All algorithms except for PNN and those in the PNNs family fail badly in at least some dataset. Conversely, basically all the PNNs(INIT) algorithms solve all the datasets in 100% of the cases (98% for PNNs(KM++) on A3), with the only notable exception being INIT=UNI. Even in that case, however, the performance is much better than UNI and REF(UNI), and at times even better than UNI and REF (KM++) which is the variant in the refine family that gives the best overall results. Furthermore, PNNs(INIT) again has 100% success rate. This demonstrates that (at least for this scenario) PNN-smoothing is a considerably better smoothing technique than refine.

The timings, in particular those of PNNs(MAXMIN) and PNNs(KM++), are generally comparable with, sometimes even better than, those of the other linear algorithms, even the best ones; on the other hand, the only other algorithm capable of solving all the datasets, PNN, is \( \Omega(N^2) \) and indeed orders of magnitude slower. The fully-recursive PNNs is, unsurprisingly, the slowest in the PNN family: it is not necessary for these datasets. In the tables we also report the number of Lloyd’s iterations, which (as expected) is generally comparable with, some-

Table II. Results on synthetic datasets

| dataset | UNI | MAXMIN | GMM++ | PNN | REF(UNI) | REF(GMM++) | PNNs(UNI) | PNNs(GMM++) | PNNs2(UNI) | PNNs2(GMM++) | PNNs(maxMIN) | PNNs(maxMIN) |
|---------|-----|--------|-------|----|----------|------------|-----------|-------------|------------|-------------|-------------|-------------|
| A3      | 0.003 | 0.06 | 0.0 | 0.0 | 0.224 | 0.689 | 1 | 1 | 1 |
| Birch1  | 0 | 0 | 0.04 | 0 | 0.02 | 1 | 1 | 1 | 1 |
| Birch2  | 0 | 0 | 0.08 | 0 | 0.41 | 0.24 | 1 | 1 | 1 |
| Unbalance | 0.215 | 0.94 | 1 | 0.001 | 1 | 0.667 | 1 | 1 | 1 |
| Dim1024 | 0.001 | 1 | 1 | 0.058 | 1 | 0.91 | 1 | 1 | 1 |

success rate

| dataset | UNI | MAXMIN | GMM++ | PNN | REF(UNI) | REF(GMM++) | PNNs(UNI) | PNNs(GMM++) | PNNs2(UNI) | PNNs2(GMM++) | PNNs(maxMIN) | PNNs(maxMIN) |
|---------|-----|--------|-------|----|----------|------------|-----------|-------------|------------|-------------|-------------|-------------|
| A3      | (10.1 ± 2.2) · 10^−3 | (7.7 ± 1.1) · 10^−3 | (9.3 ± 1.1) · 10^−3 | (558 ± 4) · 10^−3 | (63 ± 5) · 10^−3 |
| Birch1  | 0.97 ± 0.36 | 0.71 ± 0.17 | 0.56 ± 0.11 | 92.1 ± 0.5 | 1.33 ± 0.06 |
| Birch2  | 0.26 ± 0.04 | 0.21 ± 0.04 | 0.242 ± 0.021 | 98.87 ± 0.11 | 1.073 ± 0.029 |
| Unbalance | (1.9 ± 0.8) · 10^−3 | (0.66 ± 0.16) · 10^−3 | (0.97 ± 0.38) · 10^−3 | (421 ± 12) · 10^−3 | (14.2 ± 1.1) · 10^−3 |
| Dim1024 | (61 ± 10) · 10^−4 | (63 ± 5) · 10^−4 | (166 ± 14) · 10^−4 | (16260 ± 130) · 10^−4 | (481 ± 12) · 10^−4 |

Lloyd’s iterations (including during seeding where applicable)

| dataset | UNI | MAXMIN | GMM++ | PNN | REF(UNI) | REF(GMM++) | PNNs(UNI) | PNNs(GMM++) | PNNs2(UNI) | PNNs2(GMM++) | PNNs(maxMIN) | PNNs(maxMIN) |
|---------|-----|--------|-------|----|----------|------------|-----------|-------------|------------|-------------|-------------|-------------|
| A3      | 26 ± 11 | 17 ± 6 | 12 ± 5 | 5 | 26 ± 11 | 12 ± 6 | 18 ± 6 | 17 ± 4 | 11.6 ± 2.8 | 10.6 ± 2.2 |
| Birch1  | 77 ± 32 | 56 ± 19 | 36 ± 13 | 6 | 61 ± 31 | 31 ± 18 | 31 ± 6 | 26 ± 5 | 24 ± 5 | 16.7 ± 2.8 |
| Birch2  | 27 ± 11 | 26 ± 11 | 13 ± 6 | 2 | 36 ± 16 | 9 ± 6 | 26 ± 11 | 18 ± 5 | 12.2 ± 3.9 | 7.4 ± 1.2 |
| Unbalance | 11 ± 5 | 4.1 ± 0.8 | 3.2 ± 3.3 | 2 | 13 ± 7 | 4.2 ± 1.1 | 14 ± 11 | 11 ± 4 | 4.1 ± 0.6 | 7.6 ± 2.3 |
| Dim1024 | 2.94 ± 0.33 | 2 ± 0 | 2 ± 0 | 1 | 6.1 ± 2.1 | 4.001 ± 0.026 | 4.9 ± 0.4 | 6.8 ± 0.8 | 3 ± 0 | 5 ± 0 |
Table III. Results on real-world datasets

| dataset   | UNIF | MAXMIN | GKM++ | PNN | REF(UNIF) | REF(GKM++) | PNNs(UNIF) | PNNs(GKM++) | PNNsR |
|-----------|------|--------|-------|-----|-----------|------------|------------|-------------|-------|
| Bridge    | 11793 ± 9 | 11383 ± 4 | 11236 ± 34 | 10827.9 | 11622 ± 5 | 11589 ± 4 | 11231 ± 4 | 10951 ± 24 | 10911 ± 22 |
| House     | 1089 ± 13 | 1019 ± 6 | 953.8 ± 2.9 | 949.857 | 996 ± 11 | 956.4 ± 3.1 | 956.2 ± 3.3 | 947.4 ± 1.9 | 946.6 ± 1.9 |
| Miss America | 607 ± 6 | 579.7 ± 2.7 | 550.8 ± 1.9 | 531.611 | 584 ± 4 | 583.1 ± 3.1 | 561 ± 4 | 534 ± 0.9 | 533.6 ± 0.8 |
| UrbanGB   | 693 ± 134 | 299 ± 6 | 243.1 ± 1.9 | 231.562 | 533 ± 4 | 242.3 ± 1.9 | 269 ± 4 | 229.9 ± 0.4 | 229.59 ± 0.29 |
| Olivetti  | 12981 ± 261 | 12543 ± 143 | 12233 ± 136 | 11623.8 | 12963 ± 203 | 12593 ± 196 | 12159 ± 137 | 11891 ± 7 | 11809 ± 6 |
| Isolet    | 119669 ± 8 | 123447 ± 1506 | 119081 ± 6 | 117693 | 119048 ± 7 | 118358 ± 381 | 117966 ± 193 | 117893 ± 168 | 117936 ± 188 |

SSE cost (mean over 100 repetitions)

| dataset   | UNIF | MAXMIN | GKM++ | PNN | REF(init) | REF(GKM++) | PNNs(init) | PNNs(GKM++) | PNNsR |
|-----------|------|--------|-------|-----|-----------|------------|------------|-------------|-------|
| Bridge    | 11575.5 | 11286.9 | 11129.6 | 10827.9 | 11455.9 | 11421.2 | 11127.2 | 10891 | 10864.5 |
| House     | 980.161 | 1002.81 | 947.956 | 949.857 | 975.093 | 950.761 | 947.169 | 942.332 | 942.079 |
| Miss America | 590.809 | 573.246 | 546.44 | 531.611 | 574.243 | 575.463 | 552.914 | 531.905 | 531.185 |
| UrbanGB   | 501.889 | 285.497 | 238.387 | 231.562 | 442.297 | 237.643 | 257.816 | 228.972 | 228.838 |
| Olivetti  | 12286.4 | 12167.9 | 11941.9 | 11623.8 | 12415.2 | 12221.8 | 11906.7 | 11695.2 | 11657.2 |
| Isolet    | 118018 | 120656 | 117859 | 117693 | 117698 | 117856 | 117690 | 117687 | 117692 |

Convergence time

| dataset   | UNIF | MAXMIN | GKM++ | PNN | REF(init) | REF(GKM++) | PNNs(init) | PNNs(GKM++) | PNNsR |
|-----------|------|--------|-------|-----|-----------|------------|------------|-------------|-------|
| Bridge    | 0.0329 ± 0.0031 | 0.0367 ± 0.0037 | 0.0587 ± 0.0019 | 0.364 ± 0.013 |
| House     | 0.99 ± 0.13 | 0.94 ± 0.12 | 0.78 ± 0.07 | 14.01 ± 0.38 |
| Miss America | 0.079 ± 0.005 | 0.139 ± 0.024 | 0.123 ± 0.007 | 0.946 ± 0.008 |
| UrbanGB   | 9.1 ± 0.9 | 6.6 ± 1.1 | 10.2 ± 0.8 | 1389 ± 31 |
| Olivetti  | 0.063 ± 0.008 | 0.078 ± 0.011 | 0.139 ± 0.008 | 0.48 ± 0.05 |
| Isolet    | 0.49 ± 0.12 | 0.41 ± 0.09 | 0.66 ± 0.12 | 46.42 ± 0.19 |

D. Real-world datasets

We tested 7 challenging real-world datasets (see table I). The first three, Bridge, House and Miss America, were obtained from [18]: UrbanGB, Isolet and US Census are from [20] and Olivetti from [12]. All of these are comparatively large and quite challenging for SSE optimization. The Isolet dataset was chosen as the hardest one among those tested in ref. [9] (based on the results reported there); the US Census dataset was chosen for its large size, to provide a test of parallelization efficiency; the other 5 datasets were also tested in ref. [21], where it was shown that even sophisticated, state-of-the-art evolutionary algorithms cannot easily find their global minima (and indeed it is not even clear if they can find them at all). We did not scale the datasets, except UrbanGB for which we scaled the longitude by a factor of 1.7 to make distances roughly proportional to geographical distances, and US Census for which we linearly transformed each dimension individually to make them fit into the range [-1, 1]. For all datasets, except UrbanGB and Isolet, the choice of $k$ follows existing literature and is otherwise arbitrary.

As for the previous section, we report our most representative results here; the full results can be found in Appendices B.2 and B.3. We first present the results for the non-parallel case (on all datasets except US Census). Some results are presented in table III, confirming that the PNN family of algorithms attains results superior to all other linear algorithms, in comparable times, both in terms of the average and of the minimum SSE cost. More specifically, PNNs(INIT) consistently achieves better costs than both INIT and REF(INIT), for all the tested INIT. The number of Lloyd’s iterations (reported in the Appendix), is also comparable across algorithms, as in the previous case. The best SSEs are achieved in 4 cases out of 6 by PNN, which however belongs to a different computational class and is considerably slower.

In fig. 2, we plot the mean SSE vs convergence time for the best linear “plain” methods, KM++, GKM++ and MAXMIN, their REF(INIT) versions, and their PNNs(INIT) and PNNs(INIT) versions, as well as PNNsR. They all have comparable timings, but the PNNs(INIT) family is consistently below the others. Within the family, recursion can generally help producing some minor improve-

3 For the Olivetti dataset $N/k = 10$ and thus we used $J = 5$ instead of the default $J = 10$. 
Figure 2. SSE cost vs convergence time, averages over 100 samples (only 20 for \textit{USCensus}), for the real-world datasets. The meta-methods symbols enclose the symbol for their \textsc{init} algorithm. The dashed lines are the costs achieved by \textsc{pnn}, whose timings would be off-scale. The top 6 panels are non-parallel test; the bottom row are tests performed with 6 threads in parallel.
The dashed lines show for reference the costs achieved by PNN (its timings would all be off-scale, cf. table III), showing that even when it performs better than the algorithms in the PNNs family, the difference is generally relatively small.

In order to test the effect of parallelization on the performance of the algorithm, we also performed some tests with multi-threading enabled, using 6 threads. We tested the two largest datasets, US Census and UrbanGB. The results are shown in the last two rows of fig. 2 (complete data in Appendix B3); the case of UrbanGB allows a comparison with the non-parallel ones; note that we did not run PNN on US Census because it would be impractical. The results are qualitatively similar to the non-parallel versions.

Overall, the result do not highlight any clear best among the PNNs algorithms; PNNs(km++) is almost always the fastest, and it produces consistently good results.

V. DISCUSSION

We have presented a scheme for k-means seeding called PNN-smoothing that can be applied to any existing linear (O(kND)) algorithm, with a limited impact on the scaling and very little impact in practice on the convergence times. Our experiments show clear and consistent improvements on challenging synthetic and real-world datasets, and superior results with respect to the similar “refine” smoothing scheme. The advantage is maintained in the context of a parallel implementation. The experiments also indicate that PNN-smoothing is not particularly susceptible to the original seeding scheme; applying the smoothing procedure to k-means++ (which is already among the most popular seeding algorithms) may be a sensible default option.

[1] Pavel Berkhin. A survey of clustering data mining techniques. In Grouping multidimensional data, pages 25–71. Springer, 2006. URL: https://doi.org/10.1007/3-540-28349-8_2.

[2] Xindong Wu, Vipin Kumar, J Ross Quinlan, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. Top 10 algorithms in data mining. Knowledge and information systems, 14(1):1–37, 2008. URL: https://doi.org/10.1007/s10115-007-0114-2.

[3] Stuart Lloyd. Least squares quantization in pcm. IEEE transactions on information theory, 28(2):129–137, 1982. URL: https://doi.org/10.1109/TIT.1982.1056489.

[4] James MacQueen. Some methods for classification and analysis of multivariate observations. In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, volume 1: statistics, volume 1, pages 281–297. Oakland, CA, USA, University of California Press, 1967. URL: https://projecteuclid.org/euclid.bsmsp/1200512992.

[5] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007. URL: http://theory.stanford.edu/~sergei/papers/kMeansPP-soda.pdf.

[6] Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. Theoretical Computer Science, 38:293–306, 1985. URL: https://doi.org/10.1016/0304-3975(85)90224-5.

[7] Ioannis Katsavounidis, C-C Jay Kuo, and Zhen Zhang. A new initialization technique for generalized lloyd iteration. IEEE Signal Processing Letters, 1(10):144–146, 1994. URL: https://doi.org/10.1109.97.329844.

[8] William H E雀itz. A new vector quantization clustering algorithm. IEEE transactions on acoustics, speech, and signal processing, 37(10):1568–1575, 1989. URL: https://doi.org/10.1109/29.35395.

[9] M Emre Celebi, Hassan A Kingravi, and Patricio A Vela. A comparative study of efficient initialization methods for the k-means clustering algorithm. Expert systems with applications, 40(1):200–210, 2013. URL: https://doi.org/10.1016/j.eswa.2012.07.021.

[10] Pasi Franti and Sami Sieranoja. How much k-means can be improved by using better initialization and repeats? Pattern Recognition, 2019. URL: https://doi.org/10.1016/j.patcog.2019.04.014.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011. URL: http://www.jmlr.org/papers/v12/pedregosa11a.html.

[12] scikit-learn. URL: https://scikit-learn.org.

[13] Paul S Bradley and Usama M Fayyad. Refining initial points for k-means clustering. In ICML, volume 98, pages 91–99. Citeseer, 1998. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.50.8528&rep=rep1&type=pdf.

[14] Pasi Franti and Timo Kaukoranta. Fast implementation of the optimal pnn method. In Proceedings 1998 International Conference on Image Processing. ICIP98 (Cat. No. 98CB36269), pages 104–108 vol.3. IEEE, 1998. URL: https://doi.org/10.1109/ICIP.1998.999001.

[15] Pasi Franti. Genetic algorithm with deterministic crossover for vector quantization. Pattern Recognition Letters, 21(1):61–68, 2000. URL: https://doi.org/10.1016/S0167-8655(99)00133-6.

[16] Timo Kaukoranta, P Franti, and Olli Nevalainen. Reduced comparison search for the exact gla. In Proceedings DCC’99 Data Compression Conference (Cat. No. PR00096), pages 33–41. IEEE, 1999. URL: https://doi.org/10.1109/DCC.1999.755651.
[17] Code repository for PNN-smoothing and other algorithms mentioned in the paper. URL: https://github.com/carlobaldassi/KMeansPNNSmoothing.jl.

[18] Pasi Fränti and Sami Sieranoja. K-means properties on six clustering benchmark datasets. Applied Intelligence, 48(12):4743–4759, 2018. URL: http://cs.uef.fi/sipu/datasets/.

[19] Pasi Fränti, Mohammad Rezaei, and Qinpei Zhao. Centroid index: cluster level similarity measure. Pattern Recognition, 47(9):3034–3045, 2014. URL: https://doi.org/10.1016/j.patcog.2014.03.017.

[20] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL: http://archive.ics.uci.edu/ml.

[21] Carlo Baldassi. Recombinator-k-means: An evolutionary algorithm that exploits k-means++ for recombination. IEEE Transactions on Evolutionary Computation, pages 1–1, 2022. URL: https://doi.org/10.1109/TEVC.2022.3144134.
Appendix A: The effect of varying $\rho$

![Graphs showing SSE cost vs convergence time varying $\rho$, averaged over 100 samples. The symbols correspond to $\rho = 0.5$ (same data as in fig. 2 of the main text). The lines passing through each symbol span $\rho \in \{0.2, 0.5, 1.0, 2.0, 5.0\}$ where $\rho = 5.0$ is always the rightmost point in each curve.]

In this section we report the results of experiments performed at different values of the parameter $\rho$. This value was set to $0.5$ throughout the text. In fig. 3 we show the results for two real-world datasets, Bridge and Miss America. As expected from the fact that in the PNNS(INIT) algorithm $\rho \in [k/N, N/k]$ interpolates between INIT and PNN, there is generally a trade-off, such that increasing $\rho$ results in smaller costs but longer times. The PNNSR algorithm behaves similarly, except for small $\rho = 0.2$. 
Appendix B: Complete numerical results

In this section we report the full data for all numerical tests of sec. IV of the main text. This complements tables II and III of the main text, and includes additional measures.

1. Synthetic datasets

   a. Convergence time

| dataset  | UNIF  | MAXMIN | KM++  | GKM++ | PNSS  |
|----------|-------|--------|-------|-------|-------|
| A3       | (10.1 ± 2.2) · 10⁻³ | (7.7 ± 1.4) · 10⁻³ | (8.2 ± 1.7) · 10⁻³ | (9.3 ± 1.1) · 10⁻³ | (558 ± 4) · 10⁻³ |
| Birch1   | 0.97 ± 0.36 | 0.71 ± 0.17 | 0.71 ± 0.17 | 0.56 ± 0.11 | 92.1 ± 0.5 |
| Birch2   | 0.26 ± 0.04 | 0.21 ± 0.04 | 0.21 ± 0.04 | 0.242 ± 0.021 | 98.87 ± 0.11 |
| Unbalance| (1.9 ± 0.8) · 10⁻³ | (0.66 ± 0.16) · 10⁻³ | (1.1 ± 0.9) · 10⁻³ | (0.97 ± 0.38) · 10⁻³ | (421 ± 12) · 10⁻³ |
| Dim1024  | (61 ± 9) · 10⁻⁴ | (63 ± 5) · 10⁻⁴ | (68 ± 7) · 10⁻⁴ | (166 ± 14) · 10⁻⁴ | (16260 ± 130) · 10⁻⁴ |
| ref(unif) | ref(maxmin) | ref(km++) | ref(gkm++) | | |

| dataset  | PNSS(unif) | PNSS(maxmin) | PNSS(km++) | PNSS(gkm++) |
|----------|------------|--------------|------------|-------------|
| A3       | (14.1 ± 2.3) · 10⁻³ | (12.7 ± 1.6) · 10⁻³ | (11.6 ± 1.3) · 10⁻³ | (13.6 ± 1.1) · 10⁻³ |
| Birch1   | 0.81 ± 0.14 | 0.58 ± 0.11 | 0.63 ± 0.11 | 0.56 ± 0.06 |
| Birch2   | 0.36 ± 0.04 | 0.188 ± 0.017 | 0.249 ± 0.028 | 0.276 ± 0.016 |
| Unbalance| (2.7 ± 0.6) · 10⁻³ | (1.11 ± 0.27) · 10⁻³ | (1.51 ± 0.34) · 10⁻³ | (1.73 ± 0.36) · 10⁻³ |
| Dim1024  | (254 ± 14) · 10⁻⁴ | (211 ± 29) · 10⁻⁴ | (207 ± 14) · 10⁻⁴ | (298 ± 11) · 10⁻⁴ |
| ref(pnns) | ref(pnns(maxmin)) | ref(pnns(km++)) | ref(pnns(gkm++)) | |

| dataset  | PNSS²(unif) | PNSS²(maxmin) | PNSS²(km++) | PNSS²(gkm++) |
|----------|-------------|--------------|------------|-------------|
| A3       | (14.1 ± 6.1) · 10⁻³ | (12.7 ± 1.6) · 10⁻³ | (11.6 ± 1.3) · 10⁻³ | (13.6 ± 1.1) · 10⁻³ |
| Birch1   | 0.466 ± 0.011 | 0.418 ± 0.014 | 0.451 ± 0.013 | 0.507 ± 0.011 |
| Birch2   | 0.296 ± 0.019 | 0.226 ± 0.004 | 0.2496 ± 0.0038 | 0.352 ± 0.016 |
| Unbalance| (4.4 ± 3.1) · 10⁻³ | (3.12 ± 0.39) · 10⁻³ | (3.6 ± 0.6) · 10⁻³ | (4.1 ± 0.6) · 10⁻³ |
| Dim1024  | (176 ± 16) · 10⁻⁴ | (176 ± 11) · 10⁻⁴ | (174 ± 8) · 10⁻⁴ | (258 ± 11) · 10⁻⁴ |
| ref(pnns²) | ref(pnns²(maxmin)) | ref(pnns²(km++)) | ref(pnns²(gkm++)) | |

| dataset  | PNSS³(unif) | PNSS³(maxmin) | PNSS³(km++) | PNSS³(gkm++) |
|----------|-------------|--------------|------------|-------------|
| A3       | (22.2 ± 1.6) · 10⁻³ | (21.1 ± 2.4) · 10⁻³ | (23.8 ± 2.7) · 10⁻³ | (29.1 ± 3.4) · 10⁻³ |
| Birch1   | 0.513 ± 0.011 | 0.467 ± 0.009 | 0.537 ± 0.016 | 0.657 ± 0.019 |
| Birch2   | 0.399 ± 0.014 | 0.358 ± 0.016 | 0.376 ± 0.012 | 0.531 ± 0.004 |
| Unbalance| (6.7 ± 0.9) · 10⁻³ | (6.1 ± 0.6) · 10⁻³ | (6.7 ± 0.6) · 10⁻³ | (7.6 ± 0.6) · 10⁻³ |
| Dim1024  | (306 ± 11) · 10⁻⁴ | (297 ± 8) · 10⁻⁴ | (304 ± 9) · 10⁻⁴ | (388 ± 11) · 10⁻⁴ |
| ref(pnns³) | ref(pnns³(maxmin)) | ref(pnns³(km++)) | ref(pnns³(gkm++)) | |

| dataset  | PNSS⁴(unif) | PNSS⁴(maxmin) | PNSS⁴(km++) | PNSS⁴(gkm++) |
|----------|-------------|--------------|------------|-------------|
| A3       | (32 ± 4) · 10⁻³ | (31.1 ± 1.6) · 10⁻³ | (34.3 ± 1.7) · 10⁻³ | (29.1 ± 3.4) · 10⁻³ |
| Birch1   | 0.718 ± 0.031 | 0.676 ± 0.024 | 0.794 ± 0.031 | 0.657 ± 0.019 |
| Birch2   | 0.616 ± 0.031 | 0.583 ± 0.016 | 0.626 ± 0.019 | 0.531 ± 0.004 |
| Unbalance| (10.1 ± 1.6) · 10⁻³ | (9.6 ± 0.9) · 10⁻³ | (10.6 ± 1.2) · 10⁻³ | (7.6 ± 0.6) · 10⁻³ |
| Dim1024  | (441 ± 17) · 10⁻⁴ | (421 ± 11) · 10⁻⁴ | (428 ± 11) · 10⁻⁴ | (388 ± 11) · 10⁻⁴ |
| ref(pnns⁴) | ref(pnns⁴(maxmin)) | ref(pnns⁴(km++)) | ref(pnns⁴(gkm++)) | |

| dataset  | PNSS⁵(unif) | PNSS⁵(maxmin) | PNSS⁵(km++) | PNSS⁵(gkm++) |
|----------|-------------|--------------|------------|-------------|
| A3       | (63 ± 5) · 10⁻³ | | | |
| Birch1   | 1.33 ± 0.06 | | | |
| Birch2   | 1.073 ± 0.029 | | | |
| Unbalance| (14.2 ± 1.1) · 10⁻³ | | | |
| Dim1024  | (481 ± 12) · 10⁻⁴ | | | |
### b. SSE cost

| Dataset | UNIF | MAXMIN | KM++ | GKM++ | PNN |
|---------|------|--------|------|-------|-----|
| A3      | 11.4 ± 1.4 | 8.6 ± 0.6 | 9.3 ± 0.8 | 7.6 ± 0.4 | 6.7377 |
| Birch2  | 111 ± 5 | 106.6 ± 2.9 | 105.2 ± 3.6 | 99.8 ± 2.6 | 92.7731 |
| Birch2  | 1.56 ± 0.26 | 0.84 ± 0.11 | 0.86 ± 0.11 | 0.523 ± 0.034 | 0.456724 |
| Unbalance | 6.2 ± 1.1 | 6 ± 4 | 1.1 ± 0.5 | 0.68 ± 0.16 | 0.646935 |
| Dim1024 | 13792 ± 4 | 5.3938 ± 0 | 23 ± 258 | 5.3938 ± 0 | 5.3938 |

| Dataset | REF (UNIF) | REF (MAXMIN) | REF (KM++) | REF (GKM++) |
|---------|------------|--------------|------------|-------------|
| A3      | 9.7 ± 0.7  | 7.16 ± 0.29  | 8.3 ± 0.4  | 7.16 ± 0.26 |
| Birch1  | 104.4 ± 2.4 | 100.7 ± 1.9  | 101.4 ± 2.6 | 97.1 ± 1.6  |
| Birch2  | 1.23 ± 0.11 | 0.562 ± 0.038 | 0.71 ± 0.05 | 0.481 ± 0.021 |
| Unbalance | 4.4 ± 1.3 | 0.646935 ± 0 | 0.646935 ± 0 | 0.646935 ± 0 |
| Dim1024 | 5298 ± 2454 | 5.3938 ± 0 | 5.3938 ± 0 | 5.3938 ± 0 |

| Dataset | PNNs (UNIF) | PNNs (MAXMIN) | PNNs (KM++) | PNNs (GKM++) |
|---------|-------------|---------------|-------------|--------------|
| A3      | 6.91 ± 0.26 | 6.73786 ± 0.00012 | 6.74 ± 0.07 | 6.73783 ± 0.00011 |
| Birch1  | 92.77306 ± 0.00012 | 92.77305 ± 0.00011 | 92.77311 ± 0.00012 | 92.77302 ± 0.00009 |
| Birch2  | 0.497 ± 0.028 | 0.456724 ± 0 | 0.4567240 ± 1 · 10⁻⁷ | 0.4567243 ± 4 · 10⁻⁷ |
| Unbalance | 1.1 ± 0.5 | 0.646935 ± 0 | 0.646935 ± 0 | 0.646935 ± 0 |
| Dim1024 | 328 ± 1027 | 5.3938 ± 0 | 5.3938 ± 0 | 5.3938 ± 0 |

| Dataset | PNNs³ (UNIF) | PNNs³ (MAXMIN) | PNNs³ (KM++) | PNNs³ (GKM++) |
|---------|-------------|---------------|-------------|--------------|
| A3      | 6.73785 ± 0.00012 | 6.73784 ± 0.00011 | 6.73784 ± 0.00011 | 6.73783 ± 0.00011 |
| Birch1  | 92.77305 ± 0.00009 | 92.77307 ± 0.00012 | 92.77302 ± 0.00009 | 92.77306 ± 0.00011 |
| Birch2  | 0.4567243 ± 4 · 10⁻⁷ | 0.45672412 ± 3.2 · 10⁻⁷ | 0.4567240 ± 1 · 10⁻⁷ | 0.45672418 ± 3.8 · 10⁻⁷ |
| Unbalance | 0.646935 ± 0 | 0.646935 ± 0 | 0.646935 ± 0 | 0.646935 ± 0 |
| Dim1024 | 5.3938 ± 0 | 5.3938 ± 0 | 5.3938 ± 0 | 5.3938 ± 0 |
c. Success rate

| Dataset | UNIF | MAXMIN | KM++ | GKM++ | PNN |
|---------|------|--------|------|-------|-----|
| A3      | 0    | 0.003  | 0    | 0.06  | 1   |
| Birch1  | 0    | 0      | 0    | 0.004 | 1   |
| Birch2  | 0    | 0      | 0    | 0.08  | 1   |
| Unbalance | 0   | 0.215  | 0.524| 0.941 | 1   |
| Dim1024 | 0.001| 1      | 0.995| 1     | 1   |

| Dataset | REF(UNIF) | REF(MAXMIN) | REF(KM++) | REF(GKM++) |
|---------|------------|-------------|------------|------------|
| A3      | 0          | 0.283       | 0          | 0.224      |
| Birch1  | 0          | 0           | 0          | 0.02       |
| Birch2  | 0          | 0           | 0          | 0.41       |
| Unbalance | 0.001 | 1          | 1          | 1          |
| Dim1024 | 0.058      | 1           | 1          | 1          |

| Dataset | PNNS(UNIF) | PNNS(MAXMIN) | PNNS(KM++) | PNNS(GKM++) |
|---------|------------|-------------|------------|------------|
| A3      | 0.689      | 1           | 0.977      | 1          |
| Birch1  | 1          | 1           | 1          | 1          |
| Birch2  | 0.24       | 1           | 1          | 1          |
| Unbalance | 0.667 | 1          | 1          | 1          |
| Dim1024 | 0.91       | 1           | 1          | 1          |

| Dataset | PNNS²(UNIF) | PNNS²(MAXMIN) | PNNS²(KM++) | PNNS²(GKM++) |
|---------|------------|-------------|------------|------------|
| A3      | 1          | 1           | 1          | 1          |
| Birch1  | 1          | 1           | 1          | 1          |
| Birch2  | 1          | 1           | 1          | 1          |
| Unbalance | 1     | 1          | 1          | 1          |
| Dim1024 | 1          | 1           | 1          | 1          |

| Dataset | PNNS³ | PNNS³(MAXMIN) | PNNS³(KM++) | PNNS³(GKM++) |
|---------|-------|--------------|------------|------------|
| A3      | 1     | 1            | 1          | 1          |
| Birch1  | 1     | 1            | 1          | 1          |
| Birch2  | 1     | 1            | 1          | 1          |
| Unbalance | 1    | 1          | 1          | 1          |
| Dim1024 | 1     | 1            | 1          | 1          |

| Dataset | PNNSr |
|---------|-------|
| A3      | 1     |
| Birch1  | 1     |
| Birch2  | 1     |
| Unbalance | 1    |
### d. Lloyd's iterations

| Dataset | UNIF | MAXMIN | KM++ | GKM++ | PNS |
|---------|------|--------|------|-------|-----|
| A3      | 26 ± 11 | 17 ± 6 | 19 ± 8 | 12 ± 5 | 5   |
| Birch1  | 77 ± 32 | 56 ± 19 | 56 ± 18 | 36 ± 13 | 6   |
| Birch2  | 27 ± 11 | 26 ± 11 | 26 ± 11 | 13 ± 6 | 2   |
| Unbalance | 11 ± 5 | 4.1 ± 0.8 | 8 ± 8 | 3.2 ± 3.3 | 2   |
| Dim1024 | 2.94 ± 0.33 | 2 ± 0 | 2 ± 0 | 2 ± 0 | 1   |

| Dataset | REF(UNIF) | REF(MAXMIN) | REF(KM++) | REF(GKM++) |
|---------|-----------|-------------|------------|------------|
| A3      | 26 ± 11   | 12 ± 6      | 19 ± 9     | 12 ± 6     |
| Birch1  | 61 ± 31   | 44 ± 26     | 48 ± 27    | 31 ± 18    |
| Birch2  | 36 ± 16   | 19 ± 11     | 28 ± 14    | 9 ± 6      |
| Unbalance | 13 ± 7  | 4.2 ± 0.5   | 5.8 ± 2.8  | 4.2 ± 1.1  |
| Dim1024 | 6.1 ± 2.1 | 4.001 ± 0.029 | 4.001 ± 0.027 | 4.001 ± 0.026 |

| Dataset | PNS(UNIF) | PNS(MAXMIN) | PNS(KM++) | PNS(GKM++) |
|---------|-----------|-------------|------------|------------|
| A3      | 18 ± 4    | 1.16 ± 2.8  | 14 ± 4     | 10.7 ± 2.6 |
| Birch1  | 31 ± 6    | 24 ± 5      | 26 ± 5     | 21 ± 4     |
| Birch2  | 26 ± 11   | 12.2 ± 3.9  | 14.3 ± 3.9 | 8.9 ± 3.6  |
| Unbalance | 14 ± 11  | 4.1 ± 0.6   | 6.7 ± 3.2  | 4.7 ± 2.1  |
| Dim1024 | 4.9 ± 0.4 | 3 ± 0       | 3.02 ± 0.17 | 3 ± 0      |

| Dataset | PNS\(^3\)(UNIF) | PNS\(^3\)(MAXMIN) | PNS\(^3\)(KM++) | PNS\(^3\)(GKM++) |
|---------|------------------|--------------------|------------------|------------------|
| A3      | 17 ± 4           | 11.3 ± 2.7         | 14.2 ± 3.7       | 11.1 ± 2.6       |
| Birch1  | 26 ± 5           | 19 ± 4             | 22 ± 5           | 17.4 ± 3.7       |
| Birch2  | 18 ± 5           | 7.3 ± 1.8          | 11.6 ± 3.1       | 7.1 ± 1.6        |
| Unbalance | 11 ± 4       | 5.3 ± 2.1          | 6.6 ± 2.7        | 5.7 ± 2.2        |
| Dim1024 | 6.8 ± 0.8        | 4.001 ± 0.033      | 4.01 ± 0.17      | 4.001 ± 0.022    |

| Dataset | PNS\(^4\)(UNIF) | PNS\(^4\)(MAXMIN) | PNS\(^4\)(KM++) | PNS\(^4\)(GKM++) |
|---------|------------------|--------------------|------------------|------------------|
| A3      | 16 ± 4           | 11.6 ± 2.7         | 13.6 ± 3.6       | 11.6 ± 2.7       |
| Birch1  | 22 ± 4           | 17.6 ± 3.3         | 19.4 ± 3.9       | 17.1 ± 3.1       |
| Birch2  | 13.9 ± 3.2       | 6.9 ± 1.6          | 9.8 ± 2.2        | 7.2 ± 1.6        |
| Unbalance | 11 ± 4         | 6.7 ± 2.4          | 7.1 ± 2.8        | 6.8 ± 2.6        |
| Dim1024 | 7.7 ± 1.4        | 5.001 ± 0.018      | 5.02 ± 0.21      | 5.001 ± 0.019    |

| Dataset | PNSR |
|---------|------|
| A3      | 10.6 ± 2.2 |
| Birch1  | 16.7 ± 2.8 |
| Birch2  | 7.4 ± 1.2 |
| Unbalance | 7.6 ± 2.3 |
| Dim1024 | 5 ± 0  |
2. Real-world datasets (non-parallel)

a. Convergence time

| dataset      | UNIF   | MAXMIN | KM++ | GKM++ | PNN  |
|--------------|--------|--------|------|-------|------|
| Bridge       | 0.0329 | 0.0367 | 0.0036 | 0.00024 | 0.00587 | 0.00019 |
| House        | 0.99 | 0.94 | 0.71 | 0.78 | 14.01 |
| Miss America | 0.0797 | 0.139 | 0.082 | 0.123 | 0.946 | 0.008 |
| UrbanGB      | 9.1 | 6.6 | 7.1 | 10.2 | 1389 | 31 |
| Olivetti     | 0.063 | 0.078 | 0.073 | 0.139 | 0.48 | 0.05 |
| Isolet       | 0.49 | 0.41 | 0.48 | 0.66 | 46.42 | 0.19 |

| PNN (UNIF)  | PNN (MAXMIN) | PNN (KM++) | PNN (GKM++) |
|-------------|--------------|-------------|--------------|
| Bridge      | 0.162 | 0.162 | 0.166 | 0.191 |
| House       | 0.93 | 1.02 | 0.81 | 0.94 |
| Miss America| 0.226 | 0.244 | 0.241 | 0.301 |
| UrbanGB     | 9.8 | 7.6 | 7.8 | 9.7 |
| Olivetti    | 0.171 | 0.162 | 0.171 | 0.209 |
| Isolet      | 0.52 | 0.43 | 0.51 | 0.52 |

| PNN^2 (UNIF) | PNN^2 (MAXMIN) | PNN^2 (KM++) | PNN^2 (GKM++) |
|--------------|----------------|-------------|--------------|
| Bridge       | 0.0486 | 0.051 | 0.0507 | 0.0731 |
| House        | 0.76 | 0.79 | 0.76 | 0.83 |
| Miss America | 0.111 | 0.126 | 0.116 | 0.151 |
| UrbanGB      | 8.6 | 8.6 | 7.6 | 9.8 |
| Olivetti     | 0.104 | 0.098 | 0.101 | 0.146 |
| Isolet       | 0.46 | 0.47 | 0.44 | 0.47 |

| PNN^3 (UNIF) | PNN^3 (MAXMIN) | PNN^3 (KM++) | PNN^3 (GKM++) |
|--------------|----------------|-------------|--------------|
| Bridge       | 0.0652 | 0.0666 | 0.0681 | 0.0936 |
| House        | 0.81 | 0.82 | 0.87 | 1.06 |
| Miss America | 0.136 | 0.141 | 0.142 | 0.183 |
| UrbanGB      | 8.37 | 8.7 | 8.81 | 12.1 |
| Olivetti     | 0.144 | 0.153 | 0.143 | 0.183 |
| Isolet       | 0.47 | 0.46 | 0.46 | 0.51 |

| PNN^4 (UNIF) | PNN^4 (MAXMIN) | PNN^4 (KM++) | PNN^4 (GKM++) |
|--------------|----------------|-------------|--------------|
| Bridge       | 0.0953 | 0.0996 | 0.101 | 0.1281 |
| House        | 0.92 | 0.89 | 1.03 | 1.23 |
| Miss America | 0.178 | 0.182 | 0.187 | 0.231 |
| UrbanGB      | 9.11 | 9.27 | 10.2 | 13.7 |
| Olivetti     | 0.161 | 0.178 | 0.161 | 0.202 |
| Isolet       | 0.56 | 0.51 | 0.53 | 0.61 |

| PNN^5 (UNIF) | PNN^5 (MAXMIN) | PNN^5 (KM++) | PNN^5 (GKM++) |
|--------------|----------------|-------------|--------------|
| Bridge       | 0.113 | 0.113 | 0.101 | 0.1281 |
| House        | 1.42 | 1.42 | 1.42 | 1.42 |
| Miss America | 0.216 | 0.216 | 0.216 | 0.216 |
| UrbanGB      | 16.2 | 16.2 | 16.2 | 16.2 |
| Olivetti     | 0.193 | 0.193 | 0.193 | 0.193 |
| Isolet       | 0.66 | 0.66 | 0.66 | 0.66 |
b. SSE cost (average)

| dataset   | UNIF  | MAXMIN | KM++  | GKM++ | PNN  |
|-----------|-------|--------|-------|-------|------|
| Bridge    | 11793 ± 9 | 11383 ± 4 | 11547 ± 7 | 11236 ± 34 | 10827.9 |
| House     | 1009 ± 13  | 1019 ± 6  | 959.8 ± 3.7 | 953.8 ± 2.9 | 949.857 |
| Miss America | 607 ± 6   | 579.7 ± 2.7 | 569.3 ± 3.9 | 550.8 ± 1.9 | 531.611 |
| UrbanGB   | 693 ± 134  | 298 ± 4   | 273 ± 5   | 243.1 ± 1.9 | 231.562 |
| Olivetti  | 12981 ± 261 | 12543 ± 143 | 12816 ± 254 | 12233 ± 136 | 11621.8 |
| Isolet    | 119669 ± 8 | 123447 ± 1506 | 119440 ± 8 | 119081 ± 6 | 117693 |

| ref(unif) | ref(maxmin) | ref(km++) | ref(gkm++) |
|-----------|-------------|------------|------------|
| Bridge    | 11622 ± 5  | 11596 ± 4  | 11567 ± 4  | 11589 ± 4  |
| House     | 996 ± 11   | 996 ± 8    | 958.7 ± 3.7 | 956.4 ± 3.1 |
| Miss America | 584 ± 4   | 584.6 ± 3.3 | 579.3 ± 3.2 | 583.1 ± 3.1 |
| UrbanGB   | 533 ± 4    | 293 ± 5    | 263 ± 4    | 242.3 ± 1.9 |
| Olivetti  | 12963 ± 203 | 12581 ± 186 | 12797 ± 226 | 12593 ± 196 |
| Isolet    | 119048 ± 7 | 123299 ± 1207 | 119093 ± 7 | 118358 ± 381 |

| PNN(unif) | PNN(maxmin) | PNN(km++) | PNN(gkm++) |
|-----------|-------------|------------|------------|
| Bridge    | 11231 ± 4  | 10944 ± 24 | 11076 ± 36 | 10951 ± 24  |
| House     | 956.2 ± 3.3 | 951.3 ± 2.7 | 948.7 ± 2.1 | 947.4 ± 1.9 |
| Miss America | 561 ± 4   | 533.9 ± 0.8 | 540.6 ± 1.4 | 534.2 ± 0.9 |
| UrbanGB   | 269 ± 4    | 238.7 ± 1.1 | 232.2 ± 0.8 | 229.9 ± 0.4 |
| Olivetti  | 12159 ± 137 | 11909 ± 8   | 12072 ± 126 | 11891 ± 7   |
| Isolet    | 117966 ± 193 | 118496 ± 291 | 117982 ± 201 | 117893 ± 168 |

| PNN^2(unif) | PNN^2(maxmin) | PNN^2(km++) | PNN^2(gkm++) |
|-------------|---------------|-------------|---------------|
| Bridge      | 11082 ± 36    | 10923 ± 27  | 10963 ± 28    | 10921 ± 19   |
| House       | 949.4 ± 2.7   | 947.1 ± 1.9 | 947.8 ± 1.8   | 946.6 ± 2.2  |
| Miss America | 548.6 ± 3.1   | 533.4 ± 0.9 | 535.2 ± 0.9   | 533.6 ± 0.8  |
| UrbanGB     | 238.9 ± 1.1   | 230.3 ± 0.37 | 229.9 ± 0.4   | 229.44 ± 0.31 |
| Olivetti    | 11926 ± 102   | 11796 ± 7   | 11897 ± 8     | 11816 ± 6    |
| Isolet      | 117886 ± 171  | 118194 ± 186 | 117896 ± 168  | 117853 ± 166 |

| PNN^3(unif) | PNN^3(maxmin) | PNN^3(km++) | PNN^3(gkm++) |
|-------------|---------------|-------------|---------------|
| Bridge      | 10952 ± 24    | 10913 ± 21  | 10913 ± 26    | 10908 ± 27   |
| House       | 948.1 ± 2.2   | 946.7 ± 2.3 | 946.6 ± 1.8   | 946.1 ± 2.2  |
| Miss America | 539.2 ± 1.6   | 533.6 ± 0.9 | 533.8 ± 0.9   | 533.6 ± 0.8  |
| UrbanGB     | 233.9 ± 0.8   | 229.46 ± 0.32 | 229.62 ± 0.26 | 229.67 ± 0.27 |
| Olivetti    | 11940 ± 8     | 11793 ± 6   | 11887 ± 8     | 11807 ± 7    |
| Isolet      | 117905 ± 177  | 118066 ± 147 | 117934 ± 164  | 117907 ± 186 |

| PSNR       |
|------------|
| Bridge     | 10911 ± 22  |
| House      | 946.6 ± 1.9 |
| Miss America | 533.6 ± 0.8 |
| UrbanGB    | 229.59 ± 0.29 |
| Olivetti   | 11809 ± 6   |
| Isolet     | 117936 ± 188 |
c. **SSE cost (minimum)**

| dataset  | UNIF  | MAXMIN | KM++  | GKM++ | PNN   |
|----------|-------|--------|-------|-------|-------|
| Bridge   | 11575.5 | 11286.9 | 11402.1 | 11129.6 | 10827.9 |
| House    | 980.161 | 1002.81 | 951.335 | 947.956 | 949.857 |
| Miss America | 590.809 | 573.246 | 560.353 | 546.44 | 531.611 |
| UrbanGB  | 501.889 | 285.497 | 259.426 | 238.387 | 231.562 |
| Olivetti | 12286.4 | 12167 | 12313.8 | 11941.9 | 11623.8 |
| Isolet   | 118018 | 120656 | 117929 | 117859 | 117693 |

| ref(unif) | ref(maxmin) | ref(km++) | ref(gkm++) |
|-----------|--------------|-----------|-------------|
| Bridge    | 11455.9 | 11479.7 | 11442.4 | 11421.2 |
| House     | 975.093 | 977.691 | 949.049 | 950.761 |
| Miss America | 574.243 | 574.396 | 571.617 | 575.463 |
| UrbanGB   | 442.297 | 281.425 | 251.641 | 237.643 |
| Olivetti  | 12415.2 | 12205 | 12288.4 | 12221.8 |
| Isolet    | 117698 | 120137 | 117823 | 117856 |

| pnns(unif) | pnns(maxmin) | pnns(km++) | pnns(gkm++) |
|------------|--------------|-----------|-------------|
| Bridge     | 11127.2 | 10884.7 | 10978.3 | 10891 |
| House      | 947.169 | 945.352 | 942.428 | 942.332 |
| Miss America | 552.914 | 531.419 | 537.25 | 531.905 |
| UrbanGB    | 257.816 | 236.246 | 230.192 | 228.972 |
| Olivetti   | 11906.7 | 11696.7 | 11830.7 | 11695.2 |
| Isolet     | 117690 | 118074 | 117689 | 117687 |

| pnns²(unif) | pnns²(maxmin) | pnns²(km++) | pnns²(gkm++) |
|-------------|---------------|-------------|-------------|
| Bridge      | 10996.4 | 10876.8 | 10875.5 | 10875.4 |
| House       | 944.24  | 942.37  | 943.523 | 941.603 |
| Miss America | 542.016 | 530.86  | 532.809 | 531.783 |
| UrbanGB     | 236.764 | 229.509 | 229.114 | 228.723 |
| Olivetti    | 11724  | 11590  | 11756.7 | 11605.3 |
| Isolet      | 117688 | 117710 | 117688 | 117693 |

| pnns³(unif) | pnns³(maxmin) | pnns³(km++) | pnns³(gkm++) |
|-------------|---------------|-------------|-------------|
| Bridge      | 10896.4 | 10846.3 | 10850.6 | 10839.3 |
| House       | 942.839 | 941.284 | 941.561 | 941.258 |
| Miss America | 535.863 | 531.543 | 531.309 | 531.513 |
| UrbanGB     | 232.218 | 228.719 | 228.963 | 228.87 |
| Olivetti    | 11735.1 | 11633.9 | 11690.1 | 11653.9 |
| Isolet      | 117692 | 117691 | 117706 | 117690 |

| PNNR |
|------|
| Bridge | 10864.5 |
| House  | 942.079 |
| Miss America | 531.185 |
| UrbanGB | 228.838 |
| Olivetti | 11657.2 |
| Isolet  | 117692 |
d. Lloyd’s iterations

| Dataset     | UNIF         | MAXMIN       | KM++        | GKM++        | PNN         |
|-------------|--------------|--------------|-------------|--------------|-------------|
| Bridge      | 22 ± 4       | 24 ± 5       | 21 ± 4      | 19 ± 4       | 14          |
| House       | 99 ± 18      | 99 ± 21      | 71 ± 16     | 63 ± 14      | 49          |
| Miss America| 29 ± 4       | 48 ± 11      | 33 ± 6      | 33 ± 6       | 22          |
| UrbanGB     | 61 ± 12      | 62 ± 21      | 56 ± 13     | 46 ± 11      | 26          |
| Olivetti    | 9.2 ± 2.2    | 9.2 ± 2.7    | 9.1 ± 2.3   | 7.1 ± 2.1    | 6           |
| Isolet      | 29 ± 9       | 24 ± 7       | 27 ± 7      | 28 ± 11      | 8           |

| Dataset     | REF(UNIF)    | REF(MAXMIN)  | REF(KM++)   | REF(GKM++)   |
|-------------|--------------|--------------|-------------|--------------|
| Bridge      | 26 ± 8       | 26 ± 9       | 26 ± 9      | 26 ± 8       |
| House       | 94 ± 22      | 102 ± 22     | 76 ± 18     | 73 ± 16      |
| Miss America| 37 ± 11      | 46 ± 12      | 39 ± 11     | 44 ± 13      |
| UrbanGB     | 62 ± 21      | 58 ± 31      | 56 ± 23     | 46 ± 18      |
| Olivetti    | 11 ± 4       | 11 ± 4       | 11 ± 4      | 11 ± 4       |
| Isolet      | 31 ± 16      | 26 ± 12      | 29 ± 12     | 27 ± 11      |

| Dataset     | PNNs(UNIF)   | PNNs(MAXMIN) | PNNs(KM++)  | PNNs(GKM++)  |
|-------------|--------------|--------------|-------------|--------------|
| Bridge      | 24 ± 5       | 27 ± 5       | 24 ± 5      | 23 ± 4       |
| House       | 73 ± 16      | 81 ± 17      | 69 ± 14     | 67 ± 16      |
| Miss America| 36 ± 8       | 47 ± 11      | 38 ± 7      | 39 ± 9       |
| UrbanGB     | 59 ± 13      | 77 ± 21      | 49 ± 12     | 43 ± 11      |
| Olivetti    | 9.8 ± 2.7    | 9.4 ± 2.6    | 9.3 ± 2.6   | 7.6 ± 1.9    |
| Isolet      | 29 ± 9       | 31 ± 12      | 28 ± 11     | 24 ± 8       |

| Dataset     | PNNs²(UNIF)  | PNNs²(MAXMIN) | PNNs²(KM++) | PNNs²(GKM++) |
|-------------|--------------|--------------|-------------|--------------|
| Bridge      | 26 ± 5       | 27 ± 6       | 26 ± 6      | 26 ± 5       |
| House       | 72 ± 16      | 76 ± 18      | 71 ± 16     | 69 ± 16      |
| Miss America| 39 ± 8       | 46 ± 11      | 41 ± 11     | 41 ± 8       |
| UrbanGB     | 49 ± 11      | 61 ± 18      | 46 ± 9      | 41 ± 9       |
| Olivetti    | 9.6 ± 2.8    | 8.6 ± 2.4    | 9.1 ± 2.4   | 8.2 ± 1.9    |
| Isolet      | 26 ± 11      | 26 ± 11      | 26 ± 9      | 23 ± 9       |

| Dataset     | PNNs³(UNIF)  | PNNs³(MAXMIN) | PNNs³(KM++) | PNNs³(GKM++) |
|-------------|--------------|--------------|-------------|--------------|
| Bridge      | 26 ± 6       | 24 ± 5       | 26 ± 5      | 24 ± 4       |
| House       | 71 ± 16      | 72 ± 17      | 71 ± 16     | 68 ± 16      |
| Miss America| 41 ± 9       | 41 ± 9       | 41 ± 8      | 41 ± 11      |
| UrbanGB     | 49 ± 11      | 51 ± 13      | 46 ± 11     | 41 ± 8       |
| Olivetti    | 10.6 ± 2.3   | 9.6 ± 2.1    | 10.2 ± 2.3  | 9.3 ± 1.7    |
| Isolet      | 26 ± 11      | 26 ± 9       | 26 ± 9      | 22 ± 8       |

| Dataset     | PNNs⁴       |
|-------------|--------------|
| Bridge      | 21 ± 4       |
| House       | 68 ± 16      |
| Miss America| 37 ± 8       |
| UrbanGB     | 39 ± 8       |
| Olivetti    | 7.6 ± 1.6    |
| Isolet      | 23 ± 7       |
### 3. Real-world datasets (parallel)

#### a. Convergence time

| Dataset | UNIF | MAXMIN | KM++ | GKM++ | PNN |
|---------|------|--------|------|-------|-----|
| UrbanGB | 2.16 ± 0.24 | 1.96 ± 0.28 | 1.74 ± 0.26 | 4.83 ± 0.19 | 756 ± 16 |
| US Census | 21 ± 5 | 29 ± 5 | 26 ± 6 | 48 ± 4 | - |

#### b. SSE cost (average)

| Dataset | UNIF | MAXMIN | KM++ | GKM++ | PNN |
|---------|------|--------|------|-------|-----|
| UrbanGB | 679 ± 118 | 297 ± 5 | 272 ± 5 | 242.8 ± 1.7 | 231.562 |
| US Census $(\times 10^7)$ | 1.204 ± 0.011 | 1.32 ± 0.04 | 1.195 ± 0.010 | 1.171 ± 0.006 | - |

#### c. SSE cost (minimum)

| Dataset | UNIF | MAXMIN | KM++ | GKM++ | PNN |
|---------|------|--------|------|-------|-----|
| UrbanGB | 492.318 | 286.887 | 260.773 | 238.612 | 231.562 |
| US Census $(\times 10^7)$ | 1.18023 | 1.24505 | 1.1833 | 1.15998 | - |