LSDS++: Dual Sampling for Accelerated k-means++

Chenglin Fan
CSE Department, Pennsylvania State University
201 Old Main, University Park, PA 16802, USA
fanchenglin@gmail.com

Ping Li, Xiaoyun Li
LinkedIn Ads
700 Bellevue Way NE, Bellevue, WA 98004, USA
{pinli, xiaoyli}@linkedin.com

Abstract

k-means clustering is an important problem in machine learning and statistics. The k-means++ initialization algorithm has driven new acceleration strategies and theoretical analysis for solving the k-means clustering problem. The state-of-the-art variant, called LocalSearch++, adds extra local search steps upon k-means++ to achieve constant approximation error in expectation. In this paper, we propose a new variant named LSDS++, which improves the sampling efficiency of LocalSearch++ via a strategy called *dual sampling*. By defining a new capture graph based on the concept of coreset, we show that the proposed LSDS++ is able to achieve the same expected constant error with reduced complexity. Experiments are conducted to justify the benefit of LSDS++ in practice.

1. Introduction

The k-means clustering is an important problem in statistics and machine learning, with numerous applications in finance, web, networks analysis, etc. (Abbasi and Younis, 2007). Given a set P of data points in the Euclidean space $\mathbb{R}^d$ such as user profiles and gene sequences, the goal of the k-means algorithm is to group the n data points into k disjoint clusters, such that points near each other should belong to the same cluster. Typically, given a set of centers, every data point p is assigned to the cluster whose center is the closest among all centers to p. Denote the center set as C (with $|C| = k$). Formally, the objective of k-means clustering is to find C that minimizes the following cost:

$$cost(P, C) = \sum_{p \in P} \min_{c \in C} d(p, c), \quad p, c \in \mathbb{R}^d, \qquad (1)$$

where $d(p, c) = \|p - c\|^2$ is the squared l_2 distance. The number of centers, k, could be tens, hundreds or even millions depending on the industrial application.

1.1. Algorithms for k-means

In the past 50 years, continuous progress has been made to solve the clustering problem in (1). One popular and widely used approach is the Lloyd's algorithm (Lloyd, 1982). With a set of randomly initialized centers, the Lloyd's algorithm repeatedly assigns each data point to its nearest center and updates the centers by taking the average of all points in the cluster back and forth until convergence. From an information theory viewpoint, this method implements the distortion optimal quantization for multi-dimensional data. However, one drawback of Lloyd's algorithm is that the theoretical guarantee on the final solution is hard to be established. Moreover, the number of iterations required to converge can be superpolynomial (Arthur and Vassilvitskii, 2006). As an alternative approach, Kanungo et al. (2004) proposed a local search algorithm. As presented in Algorithm 1, the algorithm iteratively finds the swap between a center point in the current center set and a point outside the center set that minimizes the clustering cost. It is shown that local search achieves an approximation factor of $(9 + \epsilon)$ in polynomial time (i.e., output C such that $cost(P, C) \leq (9 + \epsilon)opt_k$, where opt_k is the optimal k-means cost). However, the strategy of finding the optimal swap between one center and another data point in every iteration usually leads to prohibitively high running time empirically. Other approximation algorithms that use local search include (Bandyapadhyay and Varadarajan, 2016; Cohen-Addad, 2018; Friggstad et al., 2019). In general, these algorithms also have running time that depends doubly exponentially on the dimension.

1.2. Center initialization for k-means

It is known that the Lloyd's algorithm can be very sensitive to the choice of the initial centers (Milligan, 1980). Therefore, there have been many works on determining good initial centers for k-means, among which the k-means++ seeding algorithm (Arthur and Vassilvitskii, 2007) (also

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

Algorithm 1 One step of local search (Arya et al., 2004)

Input: dataset P, center set C

1: **if** $\exists q \in C, \exists p \in P \setminus C$ s.t. $cost(P, C \setminus \{q\} \cup \{p\}) < cost(P, C)$ **then**
2: $\quad p', q' = \arg\min_{p \in P, q \in C} cost(P, C \setminus \{q\} \cup \{p\})$
3: $\quad C \leftarrow C \setminus \{q'\} \cup \{p'\}$
4: **end if**
5: **return** C

Algorithm 2 k−means++ (Arthur and Vassilvitskii, 2007)

Input: dataset P, number of centers k

1: Uniformly sample $p \in P$ and set $C = \{p\}$
2: **for** $i = 2$ **to** k **do**
3: $\quad$ Sample $p \in P$ with probability $\frac{cost(\{p\}, C)}{\sum_{q \in P} cost(\{q\}, C)}$
4: $\quad C \leftarrow C \cup \{p\}$
5: **end for**
6: **return** C

known as D^2-sampling) is one simple but effective initialization approach. As presented in Algorithm 2, it samples k centers one by one in k iterations. The algorithm first picks a center randomly from the dataset P. After having picked the first $(i-1)$ centers, denoted by C_{i-1}, it picks a point $p \in P$ as the i-th center with probability proportional to the smallest distance, $\min_{c \in C_{i-1}} d(p, c)$, which equals to $cost(\{p\}, C)$. The final output centers are then used as the initial center set for k-means. The running time of k−means++ is $O(nkd)$, and it has been shown that this simple sampling procedure gives an $O(\log k)$-approximation in expectation for any dataset (Arthur and Vassilvitskii, 2007). (Makarychev et al., 2020) improved the constant in $O(\log k)$ of k−means++. Bahmani et al. (2012) considered parallelizing k−means++ to reduce the number of passes required to obtain a good initialization center set. Based on k−means++, Ackermann et al. (2012) proposed StreamKM++ for data streams. Bachem et al. (2016) applied the Metropolis-Hastings algorithm to approximate the k−means++ sampling.

Recently, Lattanzi and Sohler (2019) proposed a method called LocalSearch++ which combines the local search strategy with k−means++ to further improve the quality of the initial centers found by k−means++. The algorithm is summarized in Algorithm 3. After we get the centers output by k−means++, LocalSearch++ runs some additional local search steps. In each step, LocalSearch++ first samples a new point $p \in P$ with probability proportional to $cost(\{p\}, C)$ similar to k−means++. Then, a center in C is replaced by the sampled p if and only if it minimizes the cost after being swapped with p (among all centers), and the cost after swap improves the current cost. Note that this operation (line 2 of Algorithm 3) requires searching over all points in the current center set C to find the minima. Lattanzi and Sohler (2019) showed that LocalSearch++ achieves a constant approximation in expectation with $O(k \log \log k)$ iterations. Choo et al. (2020) showed that with additional ϵk steps, the algorithm achieves a constant approximation guarantee of $O(1/\epsilon^3)$ with probability $1 - \exp(-\Omega(k^{0.1}))$. In another direction, Fan et al. (2022) applied metric embedding trees to the construction of initial clustering centers which was also combined with differential privacy (DP) (Dwork et al., 2006).

Algorithm 3 One step of LocalSearch++ (Lattanzi and Sohler, 2019)

Input: dataset P, center set C

1: Sample $p \in P$ with probability $\frac{cost(\{p\}, C)}{\sum_{q \in P} cost(\{q\}, C)}$
2: $q' = \arg\min_{q \in C} cost(P, C \setminus \{q\} \cup \{p\})$
3: **if** $cost(P, C \setminus \{p'\} \cup \{p\}) < cost(P, C)$ **then**
4: $\quad C \leftarrow C \setminus \{q'\} \cup \{p\}$
5: **end if**
6: **return** C

1.3. Our algorithm, results, and techniques

If we compare the heuristic of local search (Algorithm 1) with LocalSearch++ (Algorithm 3), we see that local search explores over all pairs of $p \in P$ and $q \in C$ to find the optimal swap, while LocalSearch++ only searches over $q \in C$, and samples the new candidate center p in the same manner as in k−means++. As such, in each LocalSearch++ step, we need to compute the distance for every point of each cluster to the other $k-1$ centers, which requires $O(ndk)$ time. Therefore, running LocalSearch++ may still be expensive in time cost if the number of centers k is relatively large, which is the case for many practical problems. In this work, we develop an algorithm which further reduces the time complexity per iteration to the time to compute the distance for every point of two clusters to other $k-1$ centers, and also attains constant approximation error in expectation after the same $O(k \log \log k)$ steps as required by LocalSearch++ to achieve a constant error level. The proposed method is called LSDS++ ("Local Search++ with Dual Sampling"). Before we introduce LSDS++ step by step in Algorithm 4 and Algorithm 5, we first (informally) summarize the main theoretical result of this paper as follows.

Theorem 1.1. *Run LSDS++ for $O(k \log \log k)$ steps to derive the output centers C. It holds that $E[cost(P, C)] = O(opt_k)$ where opt_k is the optimal k-means cost. The expected running time of each iteration is $O(dn)$.*

In Theorem 1.1, compared with LocalSearch++, our proposed LSDS++ achieves the same constant expected er-

ror with considerably less computational cost per iteration (reduced by a factor of k). This is the key advantage of LSDS++. We provide numerical results to validate the efficiency gain of LSDS++ in Section 4.

Dual sampling. The proposed LSDS++ algorithm is designed based on the technique of *dual sampling*. The idea is simple. Given that LocalSearch++ in each step samples the new candidate center $p \in P$ in a probabilistic way instead of by exhaustive search, one natural question is: can we also sample the center $q \in C$, rather than looping over all the possible centers to be swapped? It turns out that this could be feasible, with careful design. Specifically, naive uniform sampling of the center would not work, since it is very likely that a swap will remove a "good" center. In our proposed LSDS++, after we sample $p \in P$ following the distance-based rule of k−means++ and uniformly sample a center $q \in C$, we compare the cost after swapping q and p with the cost if we swap q' and p, where q' is the nearest center to p in C (i.e., $q' = \arg\min_{q \in C} d(p, q)$). We then simply take the swap with the lower cost. In other words, we tend to swap p with its nearest center in C, if this is better than swapping a random center point. This way, when picking the center to be swapped, we only need to evaluate no more than two centers instead of all k centers, which improves the efficiency per iteration by a factor of $O(k)$. As shown by Theorem 1.1, this dual sampling technique would not affect the approximation error nor the required number of iterations to achieve the constant error. Therefore, the total complexity can be significantly improved.

Analysis roadmap. We will use the superscript "*" to denote the optimal/oracle solution. Our analysis leverages an important concept called *coreset* (see Definition 2.4 for a formal definition). Basically, a coreset R of an oracle cluster c^* is the set of points within a short distance to c^*. The two key ideas in our analysis that lead to the same approximation guarantee as LocalSearch++ are: (i) replacing any center point with a point in its coreset may only increase the clustering cost by a constant factor, and (ii) the probability that a sampled point $p \in P$ belongs to some coreset (i.e., $p \in R$) is constant. Our analysis is then facilitated with the concept of "capture graph". In Lattanzi and Sohler (2019), an oracle center c^* is said to be captured by a current center c if $c = \arg\min_{c' \in C} d(c', c^*)$. In our analysis, we define a new capture graph where c^* is captured by c if its coreset R is a subset of cluster P_c. Now, suppose p is in the coreset of c^* (with constant probability). We show that with another constant probability (depending on k), if c^* is captured by a center $c \in C$, and c captures no other oracle centers, then swapping p with its nearest center would reduce the cost by a factor of $O(1 - \frac{1}{k})$; otherwise, swapping p with a randomly chosen center would also reduce the cost by $O(1 - \frac{1}{k})$. Therefore, in each step, after $p \in P$ is sampled, by comparing the swap with the nearest center q' and with a random center q and taking the better one, with a good probability we can still achieve the same rate of error reduction as in LocalSearch++.

2. Preliminaries

We recap the notations and introduce more definitions. Let P be the input point set, and C be a center set with k centers. For each center $c_i \in C$, we use $P(c_i)$ to denote the points assigned to c_i, and the cost of the center c_i is defined as $cost(c_i) = \sum_{p \in P(c_i)} d(c_i, p)$, where $d(c_i, p)$ is the distance measure between c_i and p. Let $cost(P, C)$ be the total cost of P with respect to the center set C. Hence, we have $opt_k = cost(P, C^*)$ where C^* is the optimal clustering center set.

We now formally define the D^2-sampling procedure which was first proposed in k−means++ and also adopted in LocalSearch++ and our proposed LSDS++.

Definition 2.1 (D^2-sampling)**.** Given a set $C \subseteq P$ of candidate centers, a point $p \in P$ is sampled with probability $\mathbf{Pr}[p] = cost(p, C) / \sum_{q \in P} cost(q, C)$.

Definition 2.2 (Centroid)**.** For an arbitrary set of points Q, we denote their centroid by $\mu_Q = \frac{1}{|Q|} \sum_{q \in Q} q$. Note that μ_Q may not be a point from Q.

The following folklore lemma describes an important property of the cost function and the centroid. This is analogous to the bias-variance decomposition in statistical learning and to the parallel axis theorem in physics.

Lemma 2.3. *Let $Q \subseteq P$ be a set of points. For any point $c \in P$ (possibly not in Q),*

$$cost(Q, c) = |Q| \cdot d(\mu_Q, c) + cost(Q, \mu_Q).$$

2.1. Coresets and capture graph

Assume $C^* = \{c_1^*, ..., c_k^*\}$ is the unique center set of the optimal k-means solution (tie broken). Let $P_1^*, ..., P_k^*$ be the corresponding optimal clusters (partitions) of the data points. We use $C = \{c_1, ..., c_k\}$ to denote the current center set (in each iteration) with corresponding clusters $P_1, ..., P_k$. When the indices are not relevant, for simplicity we will drop the index and write, for example, $c \in C$.

Our analysis will make use of the following definition.

Definition 2.4 (Coreset)**.** For a cluster P_j^* with center c_j^*, the point set $R_j = \{p \in P_j^* : d(p, c_j^*) \leq cost(P_j^*, c_j^*) \cdot \frac{2}{|P_j^*|}\}$ is called the coreset of P_j^*.

The following lemma states a useful property that the cost of any point in the coreset can be bounded by the optimal cost within a constant factor.

Lemma 2.5. *For any $j = 1, ..., k$ and $c \in R_j$, it holds that $cost(P_j^*, c) \leq 3cost(P_j^*, c_j^*)$.*

Proof. Based on Lemma 2.3, we have that $cost(P_j^*, c) = |P_j^*| \cdot d(c, c_j^*) + cost(P_j^*, c_j^*) \leq 3cost(P_j^*, c_j^*)$, where we bound $d(c, c_j^*)$ by Definition 2.4. □

In the line of research on local search for k-means (e.g., Kanungo et al. (2004); Lattanzi and Sohler (2019)), the capture graph is a standard tool for the error analysis. We recap the definition below.

Definition 2.6 (Capture graph (Kanungo et al., 2004))**.** A capture graph is a bipartite graph between C and C^*. We say that $c_j^* \in C^*$ is captured by a center $c_i \in C$, if c_i is the nearest center to c_j^* among all centers in C. If c_j^* is captured by c_i, an edge exists between (c_j^*, c_i) in the capture graph.

In the above definition, we see that a center $c \in C$ may capture more than one optimal center in C^*, and any optimal center $c^* \in C^*$ must be captured by one point in C. In this paper, our analysis is based on a new capture graph leveraging the concept of coresets.

Definition 2.7 (Coreset capture graph)**.** We say that c_j^* is captured by a center $c_i \in C$, if the coreset R_j of P_j^* is such that $R_j \subseteq P_i$, where P_i is the cluster of c_i. An edge is added between (c_j^*, c_i) if c_j^* is captured by c_i.

Here, c_j^* is captured by c_j only when the coreset of c_j^* is a subset of P_j. Notably, in Definition 2.7, there may exist some oracle centers c_j^* that are not captured by any center in C. We define subsets of matching and isolate candidate centers based on the new capture graph defined above.

Definition 2.8 (H_1 and H_0 centers in new capture graph)**.** In the capture graph as in Definition 2.7, we divide the vertices in C into three types such that $C = H_1 \cup H_0 \cup H_{>1}$. Each vertex in H_1 has degree 1; each vertex in H_0 has degree 0, and each vertex in $H_{>1}$ has degree larger than one.

2.2. Reassignment cost

In the analysis of local search, it is often useful to identify the clusters whose removal will not significantly increase the cost, which makes them good candidates to be swapped with a newly sampled candidate center (Kanungo et al., 2004). To this regard, the "reassignment cost" is a useful quantity measuring the change in cost when removing a center point. Without loss of generality, for indexing, we assume that for $h \in H_1$ we have that $c_h \in C$ captures c_h^*, namely the indices of the clusters with a one-to-one correspondence in the capture graph are identical.

Definition 2.9 (Reassignment cost (Lattanzi and Sohler, 2019))**.** Consider the subset of centers H_1 and H_0 defined in Definition 2.8. For each center c in H_1, the reassignment cost of c is defined as

$$reassign(P, C, c) = cost(P \setminus P_h^*, C \setminus \{c\}) - cost(P \setminus P_h^*, C).$$

For each center c in H_0, the reassignment cost is defined as

$$reassign(P, C, c) = cost(P, C \setminus \{c\}) - cost(P, C).$$

We derive the following result.

Lemma 2.10. *For $r \in H_0 \cup H_1$, it holds that,*

$$reassign(P, C, c_r) \leq \frac{21cost(P_r, C)}{100} + 72cost(P_r, C^*).$$

Proof. Let $C' = \{c_1', c_2', ..., c_k'\}$ be a set of k centers with each $c_i' \in (R_i \setminus P_r)$ if $i \neq r$ and $c_r' \in R_r$. This is always possible when $r \in H_1 \cup H_0$ based on Definition 2.8. We only consider the case $r \in H_1$ here, because the case for $r \in H_0$ is follows similarly.

The vertices in clusters other than P_r will still be assigned to their current center. Every point $p \in P_r \cap P_i^*$ $(i \neq r)$ will be assigned to the center in C that captures the point c_i'. The distance between a point $p \in P_r \cap P_i$ and the center it is assigned to can be bounded by two parts: 1) the distance between p and q_p, and the distance between $q_p = c_i'$ and p; 2) the distance between q_p and the center (just rename it c_i) who captures c_i' in the capture graph. Then by triangle inequality, we have $cost(p, c_i) \leq cost(p, q_p) + cost(q_p, c_i)$. Moreover, based on Lemma 2.3, we have that $|cost(p, C) - cost(q_p, C)| \leq \frac{cost(p,C)}{10} + 11cost(p, C')$ and $|cost(q_p, C) - cost(c_i, C)| \leq \frac{cost(q_p,C)}{10} + 11cost(q_p, C')$. Summing these up with all $p \in P_r \setminus P_r'$, we obtain $reassign(P, C, c_r) \leq \frac{21cost(P_r,C)}{100} + 24cost(P_r, C') \leq \frac{21cost(P_r,C)}{100} + 24 \times 3cost(P_r, C^*)$ based on Lemma 2.5. □

3. LSDS++ with Dual Sampling

Algorithm 4 Local Search for k-means++ via Dual Sampling

Input: dataset P, number of iterations Z

1: $C = k$-means++ seeding(P, k)
2: **for** $i = 1$ **to** Z **do**
3: $\quad C =$ LSDS++(P, C)
4: **end for**
5: **return** C

In this section, we introduce and analyze the proposed LSDS++ method. As presented in Algorithm 4, our algorithm has similar general structure as LocalSearch++—we first deploy k-means++ to get a initial center set, and

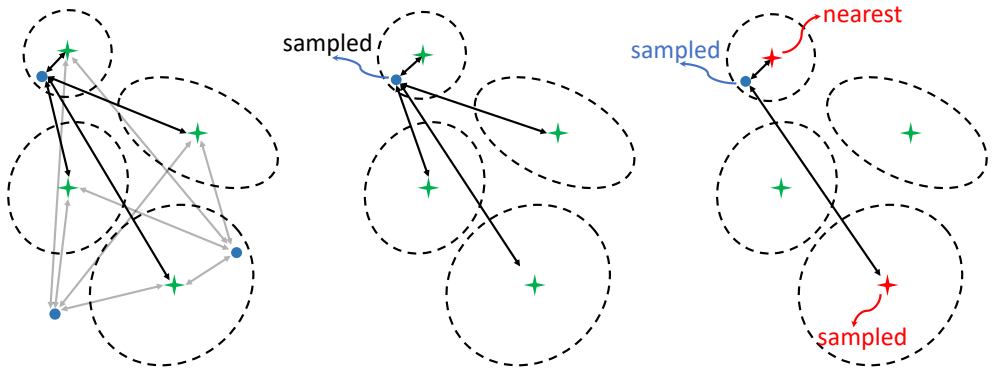

Figure 1. Illustration of local search (left), LocalSearch++ (middle) and the proposed LSDS++ (right), correspond to Algorithm 1, 3 and 5, respectively. Stars represent cluster centers in C, and circles are data points in P. In local search, we try the swaps between every pair of $p \in P$ and $q \in C$ to find the optimal swap. In LocalSearch++, p is sampled by D^2-sampling, and every swap with $q \in C$ is evaluated. In LSDS++, with a randomly sampled $p \in P$, we only evaluate two swaps of p: (i) with a randomly sampled $q \in C$ and (ii) with the nearest center in C.

Algorithm 5 One step of LSDS++

Input: dataset P, centers C

1: Sample $p \in P$ with probability $\frac{cost(\{p\},C)}{\sum_{q\in P} cost(\{q\},C)}$
2: $q' := \arg\min_{x\in C} d(x,p)$
3: Uniformly sample $q \in C$
4: **if** $cost(P, C \setminus \{q\} \cup \{p\}) > cost(P, C \setminus \{q'\} \cup \{p\})$ **then**
5: $\quad q = q'$
6: **end if**
7: **if** $cost(P, C \setminus \{q\} \cup \{p\}) < cost(P, C)$ **then**
8: $\quad C \leftarrow C \setminus \{q\} \cup \{p\}$
9: **end if**
10: **return** C

then run Z additional steps of local search with *dual sampling*. The one-step operations are summarized in Algorithm 5. In each iteration, we first apply D^2-sampling (Definition 2.1) to sample a candidate point $p \in P$. Then, we find the $q' \in C$ that is closest to p in C, and uniformly randomly sample $q \in C$. We then evaluate the swaps between (p, q') and (p, q), and keep the one with the smaller cost. Finally, as long as the swap improves the current cost, we trigger the swap operation (line 7 to line 9). LSDS++ avoids searching over all points in C when determining q, which leads to improved computational cost. See Figure 1 for an intuitive illustration of the operations of local search, LocalSearch++ and LSDS++.

The rest of this section will be devoted to the proof of Theorem 1.1. The main component is to analyze the improvement in the cost in every iteration, after which we can aggregate the cost of multiple iterations to get the overall error guarantee. Our general proof structure follows that of Lattanzi and Sohler (2019), but we need new definitions and bounds to handle the new capture graph associated with the coresets. Note that the constants in our analysis are not optimal and are possible to be further reduced. Our analysis considers two cases in each LSDS++ iteration. In our analysis, we assume $cost(P, C) \geq 2000 opt_k$; otherwise the current solution C already achieves constant error as desired.

3.1. Case 1: $\sum_{h\in H_1} cost(P_h^*, C) \geq \frac{1}{3} cost(P, C)$

In this case, we will show that with a constant probability $\frac{3}{125}$, we will sample $p \in R_h$ for some $h \in H_1$. In this case, in each iteration of LSDS++, swapping p with its nearest center $q' = \arg\min_{q\in C} d(p, q)$ would reduce the cost by at least $(1 - \frac{1}{100k})$.

We will need the following definition based on Lemma 2.5.

Definition 3.1 (Lattanzi and Sohler (2019))**.** A cluster index $h \in H_1$ (with center c_h) is called good if it holds that

$$cost(P_h^*, C) - 9cost(P_h^*, c_h^*) - reassign(P, C, c_h) \geq \frac{cost(P, C)}{100k}.$$

The left-hand side of Definition 3.1 represents the gain in the cost of replacing c_h by a point $p \in R_h$ for $h \in H_1$.

Next, we show that the cost of good clusters takes a large portion in the total cost.

Lemma 3.2. *If* $\sum_{h\in H_1} cost(P_h^*, C) \geq \frac{1}{3} cost(P, C)$*, we have that*

$$\sum_{h\in H_1,\, h \text{ is good}} cost(P_h^*, C) > \frac{87}{400} cost(P, C).$$

Proof. Based on Definition 3.1, we have

$$\begin{aligned}
&\sum_{h\in H_1,\text{ h is not good}} cost(P_h^*, C)\\
&\le \sum_{h\in H_1} reassign(P, C, c_h)\\
&\qquad + 3\sum_{h\in H_1} cost(P_h^*, c_h^*) + \frac{cost(P,C)}{100}\\
&\le \sum_{h\in H_1} 72cost(P_h, C^*) + 9opt_k + 22cost(P,C)/100\\
&\le 72opt_k + 9opt_k + \frac{22cost(P,C)}{100},
\end{aligned}$$

where we apply Lemma 2.10. Therefore, we know that $\sum_{h\in H_1,\text{ h is not good}} cost(P_h^*, C) \le \frac{22cost(P,C)}{100} + 81opt_k$. Since by assumption $cost(P,C) \ge 2000opt_k$, we have that $\sum_{h\in H_1,\text{ h is good}} cost(P_h^*, C) \ge (\frac{1}{3} - \frac{22}{100} - \frac{81}{2000})cost(P,C) > \frac{87cost(P,C)}{400}$. □

Then we have the following lemma.

Lemma 3.3. *If $\sum_{h\in H_1} cost(P_h^*, C) \ge \frac{1}{3}cost(P,C) \ge \frac{2000}{3}opt_k$, with probability at least $\frac{24}{1000}$ the new clustering after swapping has cost at most $(1 - \frac{1}{100k})cost(P,C)$.*

Proof. By Lemma 6 in (Lattanzi and Sohler, 2019), it holds that $cost(R_j, C) \ge \frac{1}{9}cost(P_j^*, C)$ when $cost(P_j^*, C) \ge 9cost(P_j^*, c_j^*)$. Hence,

$$\begin{aligned}
\sum_{h\in H_1, h\text{ is good}} cost(R_h, C) &> \frac{87}{400}\cdot\frac{1}{9}cost(P,C)\\
&\ge \frac{24}{1000}cost(P,C).
\end{aligned}$$

By Definition 2.1, we have at least $\frac{24}{1000}$ probability to sample $p \in R_h$ for some good cluster h. In this event, by Definition 3.1, the swap reduces the cost by $(1 - \frac{1}{100k})$, which proves the claim. □

3.2. Case 2: $\sum_{h\in H_1} cost(P_h^*, C) < \frac{1}{3}cost(P,C)$

Denote $X = \{1, ..., k\} \setminus H_1 = H_0 \cup H_{>1}$. We only need to consider $|H_0| \ge 1$ in this case; otherwise $|H_0| = 0$ and $|X| = 0$ since $|X| \le 2|H_0|$. Then $\sum_{h\in H_1} cost(P_h^*, C) = cost(P,C)$ which renders Case 2 invalid.

When $|H_0| \ge 1$, by the condition of Case 2, we know that $\sum_{r\in X} cost(P_r^*, C) \ge \frac{2}{3}cost(P,C)$. We will show that with probability at least $\frac{|H_0|}{90k}$, each iteration of LSDS++ reduces the clustering cost by at least $(1 - \frac{1}{200|H_0|})$ by uniform random sampling.

Next, we define the good clusters in X as follows.

Definition 3.4. A cluster index $i \in X$ is called good if $|H_{0,i}| \ge \frac{|H_0|}{10}$ where

$$\begin{aligned}
H_{0,i} = \{h \in H_0 : &cost(P_i^*, C) - reassign(P, C, c_h)\\
&-9cost(P_i^*, c_i^*) \ge \frac{cost(P,C)}{200|H_0|}\}.
\end{aligned}$$

Analogue to Definition 3.1, the above definition estimates the cost of removing c_h and inserting a new cluster center in R_i. Note that, Definition 3.4 is different from previous work (Lattanzi and Sohler, 2019) in that here a good cluster requires $|H_{0,i}|$ to be large enough. This is because when picking $q \in C$, LSDS++ uses random sampling instead of finding the exact minima. Therefore, we will need the good clusters to contain a sufficient number of points so that they can be sampled with good probability.

In the following, we argue that the sum of cost of good clusters is large. This will be useful to show that the probability of sampling such a cluster is high enough.

Lemma 3.5. *If $\sum_{i\in X} cost(P_i^*, C) \ge \frac{2}{3}cost(P,C)$, we have that*

$$\sum_{i\in X,\ i\text{ is good}} cost(P_i^*, C) > \frac{1}{10}cost(P,C).$$

Proof. By Definition 3.4 and Lemma 2.10, since $|X| \le 2|H_0|$, we have

$$\begin{aligned}
&\sum_{i\in X,\text{ i is not good}} cost(P_i^*, C)\\
&\le \frac{2}{9/10}\sum_{i\in H_0} reassign(P, C, c_i) + 9\sum_{i=1}^{k} cost(P_i^*, c_i^*)\\
&\qquad + \frac{2|H_0|cost(P,C)}{200|H_0|}\\
&= \frac{20}{9}\left(21\frac{cost(P,C)}{100} + 72opt_k)\right) + 9opt_k + \frac{cost(P,C)}{100}\\
&< \frac{48cost(P,C)}{100} + 169opt_k.
\end{aligned}$$

Then we obtain

$$\sum_{i\in X, i\text{ is not good}} cost(P_i^*, C) \le 48cost(P,C)/100 + 169opt_k.$$

As $cost(P,C) \ge 2000opt_k$, we obtain

$$\begin{aligned}
\sum_{i\in X, i\text{ is good}} cost(P_i^*, C) &\ge (\frac{2}{3} - \frac{48}{100} - \frac{169}{2000})cost(P_i^*, C)\\
&> \frac{1}{10}cost(P,C),
\end{aligned}$$

which finishes the proof. □

Lemma 3.6. *If* $\sum_{i\in X} cost(P_i^*, C) \geq \frac{2}{3}cost(P, C) \geq \frac{4000}{3}opt_k$, *with probability at least* $\frac{|H_0|}{90k}$, *the new clustering has cost at most* $cost(P, C) - (cost(P_i^*, C) - reassign(P, C, c_l) - 9cost(P_i^*, c_i^*)) \leq (1 - \frac{1}{200|H_0|})cost(P, C)$.

Proof. Similar to the reasoning for proving Lemma 3.3, with probability at least $\frac{1}{10}\frac{1}{9}\frac{|H_0|}{k} = \frac{|H_0|}{90k}$, we can sample $p \in R_h$ for some good h in H_0. Then the result follows from Definition 3.4. □

3.3. Putting parts together

Finally, we combine pieces together and obtain the main Theorem 1.1 which is restated in detail below.

Theorem 3.7. *Let* $P \subseteq \mathbb{R}^d$ *be a set of points and* C *be the output of Algorithm 4 with* $Z \geq 20000k \log\log k$ *steps. Then we have* $E[cost(P, C)] = O(opt_k)$. *Consider a variant method of Algorithm 4 where only the randomly selected center is considered at each iteration, then the expected running time of that is* $O(ndk \log\log k)$.

Proof. We first consider case two, under the condition that $\sum_{h\in H_1} cost(P_h^*, C) < \frac{1}{3}cost(P, C)$. Let $cost_0$ denote the cost of the solution after k−means++ in Algorithm 4.

First, we consider running $Y = 10000k \log\log k$ iterations that all satisfy Case 2. By Lemma 3.6, LSDS++ reduces the cost by multiplicative $(1 - \frac{1}{200|H_0|})$ with probability $\frac{|H_0|}{90k}$ in each iteration. Analytically, we assume that we additionally increase the cost additively by $2000opt_k$ at the end of each iteration. Let Y_i $(i = 1, 2, ..., k)$ be the number of iterations that $|H_0| = i$ in Y steps. So $\sum_{i=1}^k Y_i = Y$ holds. Denote the output after Y steps as C_Y. Then we have

$$
\begin{aligned}
&E[cost(P, C_Y)] \\
=&2000opt_k + \prod_{i=1}^{k}\left(\sum_{i=0}^{Y_i}\binom{Y_i}{i}(\frac{i}{90k})^i(1-\frac{i}{90k})^{Y_i-i}\right. \\
&\qquad\qquad\left.(1-\frac{1}{200i})^i\right) \\
=&2000opt_k + cost_0\prod_{i=1}^{k}(1-\frac{1}{18000k})^{Y_i} \\
=&2000opt_k + cost_0(1-\frac{1}{18000k})^{18000k\log\log k} \\
\leq&2000opt_k + cost_0/\log k \\
=&O(opt_k),
\end{aligned}
$$

where we use the fact that $E[cost_0] = O(\log k \cdot opt_k)$ for k−means++ by Arthur and Vassilvitskii (2007).

Now we consider Case 1. We assume independently running $Y = 100000k \log\log k$ steps that satisfy Case 1. With similar analysis, we can show that after $Y \geq 100000k \log\log k$ steps, $E[cost(P, C_Y)] = O(opt_k)$ also holds.

To combine the two cases together, notice that in $Z = 20000k \log\log k$ iterations, one of Case 1 and Case 2 must hold in more than $10000k \log\log k$ iterations, which reduces the cost to the constant error as required. Iterations in the other case will not increase the cost, which proves the overall constant approximation error.

Regarding the time complexity, we execute $O(k \log\log k)$ rounds of local search, and each step involves two stages (sample $p \in P$ and compute k-means costs). First, to sample $p \in P$, each step only takes $O(dn)$ time by maintaining the distance between every data point to the current center set. In the local search step, we compute the cost of swapping the new sample point with an old center q. For each point in the cluster with center q, we need to compute its distance to all other centers. Since we uniformly sample a center $q \in C$, on average the size of every cluster is $O(\frac{n}{k})$ each step, hence the total expected time is $O(ndk \log\log k)$. □

The proposed LSDS++ achieves constant approximation error in expectation, in $O(k \log\log k)$ steps. As a comparison, LocalSearch++ in Lattanzi and Sohler (2019) requires $O(k \log\log k)$ steps to reach a constant error. However, our algorithm is more efficient, especially when the number of clusters k is large (which is common in industry).

4. Numerical Experiments

We provide clustering experiments on various public datasets to demonstrate the practical benefits of the proposed LSDS++ method. The main objective is to validate that LSDS++ is able to attain the same error level of LocalSearch++, with substantially improved efficiency.

4.1. Datasets

We conduct our experiments on five benchmark datasets:

- DNA (Hsu and Lin, 2002): a dataset with 2000 instances of DNA sequences and 180 features. The dataset is available at LIBSVM repository[1].
- RNA (Uzilov et al., 2006): this dataset contains 59535 samples with 8 features. The dataset was used to study the detection of non-coding RNAs (Uzilov et al., 2006).
- PHY: 50000 data points from particle physics with 78 features. The data were obtained from high-energy collision experiments and used in KDD Cup 2004[2].

[1] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets

[2] http://osmot.cs.cornell.edu/kddcup/datasets.html

- BIO: 145751 data samples with 74 features that measure the match between a protein and a native sequence. This dataset was also used in KDD Cup 2004.
- MNIST (LeCun et al., 1998): a hand-written digit dataset with 60000 samples and 784 features.

In data pre-processing, on all datasets, each sample point is normalized to have unit l_2 norm.

4.2. Algorithms

The following center initialization algorithms are compared:

- k−means++ (Arthur and Vassilvitskii, 2007) : the classic random seeding approach (Algorithm 2).
- LocalSearch++ (Lattanzi and Sohler, 2019): we first run k−means++, and then apply extra local search steps with random sampling (Algorithm 3).
- LSDS++: Our proposed algorithm using the trick of dual sampling (Algorithm 5).

In our experiments, following the setting in Lattanzi and Sohler (2019), we first run k−means++ to obtain an initial center set C_0. Then, starting from C_0, we run 500 steps of LocalSearch++ and LSDS++, respectively. Moreover, we also test the performance of running 20 standard local search steps (Arya et al., 2004) starting from the initial centers found by the above algorithms, respectively, to justify whether the improved initialization could indeed lead to better clustering eventually. We test the number of clusters $k \in \{3, 5, 10, 20, 30, 50\}$. All the presented results are averaged over 10 independent runs.

4.3. Results

In Figure 2, we report the cost of the initial centers against the number of iterations. Denote the "current" center set in each iteration of the algorithm as C_f. We report the relative cost ratio $cost(P, C_f)/cost(P, C_0)$, which represents the improvement of the two refined algorithms upon the vanilla k−means++. We present the results for $k = 10$ and $k = 30$, and the conclusions for other k values are similar. As we can see, the proposed LSDS++ achieves roughly the same cost as that of LocalSearch++ with the same number of steps, as stated by our theory.

Figure 3 plots the cost ratio versus wall-clock time. Recall that the complexity of LocalSearch++ per iteration is $O(dnk^2)$, while the complexity of the proposed LSDS++ per iteration is $O(dnk)$ if the randomly picked center is preferred. From the plots, we verify that LSDS++ is able to achieve essentially the same cost as LocalSearch++, but with significantly improved efficiency. We also observe

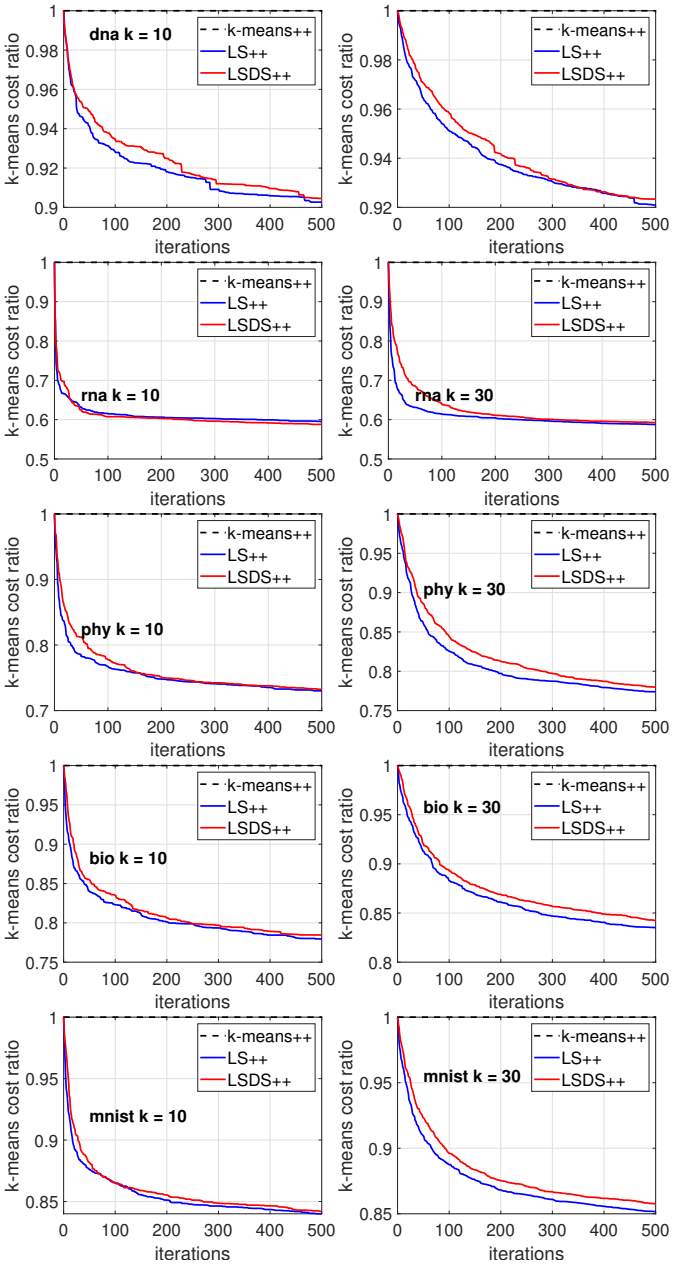

Figure 2. The k-means cost ratio against the number of iterations. We first run k-means++ to obtain centers C_0. Then we run LS++ and LSDS++ starting from C_0 for 500 iterations, respectively. The curves are the corresponding $cost(P, C_f)/cost(P, C_0)$ where C_f is the center set after each iteration.

that the gain in running time is more substantial when k increases. For example, when $k = 10$, LSDS++ runs $\approx 5\times$ times faster than LocalSearch++, and with $k = 30$ the improvement is more than $\approx 10\times$. This is consistent with our result that LSDS++ saves a factor of k in complexity. Our experiments demonstrate that LSDS++ can serve as an accurate and highly efficient seeding (initialization) algorithm for the k-means clustering in practice.

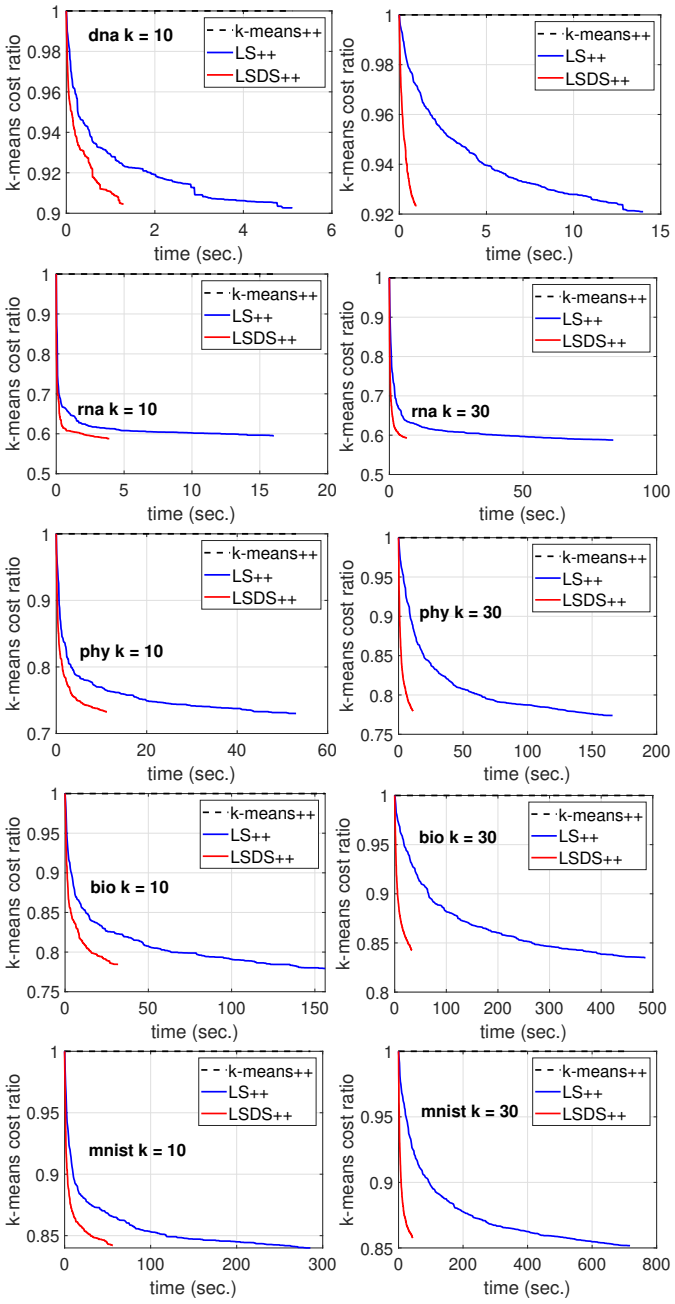

Figure 3. The cost ratio of LocalSearch++ and LSDS++ over the cost of k−means++ against the wall-clock running time.

In Table 1, we report the ratio of the cost after further running 20 local search steps upon initialization, over the cost at k−means++ initialization. That is, for example, we first run LSDS++ (same as in Figure 2), and then run 20 steps of local search, and divide the cost by the cost of C_0 where C_0 is the cost at k−means++ initialization. We observe that: (i) the proposed LSDS++ achieves a similar final cost as that of LocalSearch++; (ii) LSDS++ always attains smaller clustering cost than running local search starting from k−means++, and significantly so on RNA, PHY, and BIO. These results justify that the proposed LSDS++ is able to improve the final clustering quality when additional local search steps are applied from the initial centers of LSDS++.

Table 1. The relative cost after running extra 20 steps of local search starting from the initial clusters found by k−means++, and 500 steps of LocalSearch++ and LSDS++, respectively. For consistency, the cost is relative to the cost of the k−means++ initialization, i.e., C_0 in Figure 2 and Figure 3.

	k = 10			k = 30		
	LS++	**LSDS++**	**KM++**	**LS++**	**LSDS++**	**KM++**
dna	**0.5506**	0.5509	0.5519	**0.5706**	**0.5706**	0.5750
rna	**0.5633**	0.5541	0.6283	**0.5680**	0.5696	0.6620
phy	0.5954	**0.5950**	0.6674	0.5974	**0.5968**	0.6500
bio	**0.5976**	**0.5975**	0.6148	**0.6322**	0.6329	0.6434
mnist	0.5550	**0.5545**	0.5588	**0.5646**	0.5659	0.5703

5. Conclusion

We propose LSDS++, an accurate and highly efficient initialization algorithm for the k-means clustering problem. We propose a "dual sampling" strategy in the local search step which reduces the number of candidate swap centers each step of LocalSearch++ by a factor of $O(k)$, where k is the number of clusters in k-means. By introducing coresets and a new capture graph into our analysis, we prove that the proposed LSDS++ achieves the same constant expected approximation error as LocalSearch++. Numerical results are also provided, which verify that compared with LocalSearch++, the proposed LSDS++ can attain the same clustering cost with significantly reduced running time. We expect LSDS++ to be adopted in practice as a highly efficient large-scale clustering algorithm.

References

Ameer Ahmed Abbasi and Mohamed F. Younis. A survey on clustering algorithms for wireless sensor networks. *Comput. Commun.*, 30(14-15):2826–2841, 2007.

Marcel R Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammersen, and Christian Sohler. Streamkm++ a clustering algorithm for data streams. *Journal of Experimental Algorithmics (JEA)*, 17: 2–1, 2012.

David Arthur and Sergei Vassilvitskii. How slow is the k-means method? In Nina Amenta and Otfried Cheong, editors, *Proceedings of the 22nd ACM Symposium on Computational Geometry (SCG)*, pages 144–153, Sedona, AZ, 2006.

David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the*

Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 1027–1035, New Orleans, LA, 2007.

Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM J. Comput.*, 33(3):544–562, 2004.

Olivier Bachem, Mario Lucic, S. Hamed Hassani, and Andreas Krause. Approximate k-means++ in sublinear time. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI)*, pages 1459–1467, Phoenix, AZ, 2016.

Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable k-means++. *Proc. VLDB Endow.*, 5(7):622–633, 2012.

Sayan Bandyapadhyay and Kasturi R. Varadarajan. On variants of k-means clustering. In *Proceedings of the 32nd International Symposium on Computational Geometry (SoCG)*, Boston, MA, 2016.

Davin Choo, Christoph Grunau, Julian Portmann, and Václav Rozhon. k-means++: few more steps yield constant approximation. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pages 1909–1917, Virtual Event, 2020.

Vincent Cohen-Addad. A fast approximation scheme for low-dimensional k-means. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 430–440, New Orleans, LA, 2018.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Theory of Cryptography Conference (TCC)*, pages 265–284, New York, NY, 2006.

Chenglin Fan, Ping Li, and Xiaoyun Li. k-median clustering via metric embedding: Towards better initialization with differential privacy. *arXiv preprint arXiv:2206.12895*, 2022.

Zachary Friggstad, Mohsen Rezapour, and Mohammad R. Salavatipour. Local search yields a PTAS for k-means in doubling metrics. *SIAM J. Comput.*, 48(2):452–480, 2019.

Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE Trans. Neural Networks*, 13(2):415–425, 2002.

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. *Computational Geometry*, 28(2-3):89–112, 2004.

Silvio Lattanzi and Christian Sohler. A better k-means++ algorithm via local search. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 3662–3671, Long Beach, CA, 2019.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inf. Theory*, 28(2):129–136, 1982.

Konstantin Makarychev, Aravind Reddy, and Liren Shan. Improved guarantees for k-means++ and k-means++ parallel. In *Advances in Neural Information Processing Systems (NeurIPS)*, virtual, 2020.

Glenn W Milligan. An examination of the effect of six types of error perturbation on fifteen clustering algorithms. *Psychometrika*, 45(3):325–342, 1980.

Andrew V Uzilov, Joshua M Keegan, and David H Mathews. Detection of non-coding rnas on the basis of predicted secondary structure formation free energy change. *BMC Bioinformatics*, 7(1):1–30, 2006.