FLS: A New Local Search Algorithm for K-means with Smaller Search Space

Junyu Huang[1], **Qilong Feng**[1*], **Ziyun Huang**[2], **Jinhui Xu**[3] and **Jianxin Wang**[1,4]
[1]School of Computer Science and Engineering, Central South University, Changsha 410083, China
[2]Department of Computer Science and Software Engineering, Penn State Erie, the Behrend College
[3]Department of Computer Science and Engineering, State University of New York at Buffalo, NY, USA
[4]The Hunan Provincial Key Lab of Bioinformatics, Central South University, Changsha 410083, China
junyuhuangcsu@foxmail.com, csufeng@mail.csu.edu.cn, zxh201@psu.edu, jinhui@cse.buffalo.edu, jxwang@mail.csu.edu.cn

Abstract

The k-means problem is an extensively studied unsupervised learning problem with various applications in decision making and data mining. In this paper, we propose a fast and practical local search algorithm for the k-means problem. Our method reduces the search space of swap pairs from $O(nk)$ to $O(k^2)$, and applies random mutations to find potentially better solutions when local search falls into poor local optimum. With the assumption of data distribution that each optimal cluster has "average" size of $\Omega(\frac{n}{k})$, which is common in many datasets and k-means benchmarks, we prove that our proposed algorithm gives a $(100 + \epsilon)$-approximate solution in expectation. Empirical experiments show that our algorithm achieves better performance compared to existing state-of-the-art local search methods on k-means benchmarks and large datasets.

1 Introduction

Clustering problems have received much attention over the past decades. The goal of clustering is to partition the given dataset into clusters such that the points within the same cluster are more similar to each other, while the points in different clusters have less similarity. Among different objective functions, k-means is one of the most widely used, which aims to minimize the sum of the squared distances between points and their closest clustering centers.

Since the k-means problem is NP-hard, heuristic and approximation algorithms are two of the most widely used methods. The most popular heuristic algorithm solving this problem is the well-known Lloyd's algorithm [Lloyd, 1982]. Though depending on the initialization of center set selection and lacking theoretical guarantee, Lloyd's algorithm performs well in practice due to its fast speed and simplicity. By using k-means++ seeding technique [Arthur and Vassilvitskii, 2007] as the initial center set, empirical performance of Lloyd's algorithm becomes more stable, since k-means++ provides $O(\log k)$-approximate solution in expectation.

For approximation algorithms, local search is one of the most popular methods. The current best approximation ratio for the k-means problem based on local search is $9 + \epsilon$ given in [Kanungo *et al.*, 2004]. However, the algorithm in [Kanungo *et al.*, 2004] needs to enumerate $O(nk)$ swap pairs in each round. Even with single swap strategy, it is difficult to handle large datasets. Lattanzi and Sohler [Lattanzi and Sohler, 2019] proposed LS++ algorithm that combines k-means++ with local search to further improve the performance of local search method. The basic idea of LS++ is to iteratively sample centers for swap using D^2-sampling method. They proved that LS++ can return a constant factor approximate solution in expectation after $O(k \log \log k)$ rounds of sampling and swap. The total running time of LS++ is $O(ndk^2 \log \log k)$. Choo et al. [Choo *et al.*, 2020] proved that $O(k)$ rounds of sampling and swap are enough for getting a constant approximate solution. Although LS++ provides a way to reduce the search space of local search, it is easy for LS++ to fall into local optimum while there is still a considerable gap between the local optimum and the minimum clustering cost. For LS++, $100000k \log \log k$ rounds of sampling and swap return a 509-approximate solution [Lattanzi and Sohler, 2019].

In this paper, we propose a new local search algorithm for the k-means problem, which aims to achieve good solutions with smaller search space. Meryerson et al. [Meyerson *et al.*, 2004] proposed β-average data distribution assumption, in which each optimal cluster has size at least $\frac{\beta n}{k}$ for some constant $\beta \in (0, 1]$. We observe that k-means benchmarks [Fränti and Sieranoja, 2018] and many real-world datasets[1] follow the β-average data distribution. Based on β-average data distribution, we develop new searching strategies to reduce the search space. Firstly, we analyze the local structure of the current solution, and use nearest neighbor search for local adjustment. By single swap strategy, $O(k)$ swap pairs are constructed for local optimization. Secondly, random sampling is used to jump out of the local optimum of nearest neighbor search. With β-average assumption, clustering cost can be improved with high probability using uniform and random sampling. Thirdly, once the solution is trapped when nearest neighbor search and random sampling both fail, random mutation is performed to restart the searching process

*Corresponding author.

[1]https://archive.ics.uci.edu/ml/index.php

to find better solutions until convergence. The main contributions of this paper are summarized as follows.

- We introduce a fast and practical local search algorithm, named FLS, which reduces the search space of swap pairs from $O(nk)$ to $O(k^2)$ while maintaining the clustering quality using modified neighbor search techniques and random mutation. The total running time of our algorithm is $O(ndk^3 \log \frac{1}{\epsilon})$, where ϵ is a parameter to control approximation ratio. Smaller search space provides great potential for local search acceleration in practice.
- Under "average" cluster size assumption [Meyerson *et al.*, 2004; Ding and Huang, 2021] that each optimal cluster has size at least $\frac{\beta n}{k}$ for some constant $\beta \in (0, 1]$, we prove that our algorithm has approximation ratio $(100 + \epsilon)$ in expectation.
- FLS scales well on large datasets and makes improvements on time and clustering cost compared with state-of-art local search methods, especially on those "hard instances" defined in the k-means benchmarks [Fränti and Sieranoja, 2018].

1.1 Other Related Works

Arya et al. [Arya *et al.*, 2004] proposed the first local search heuristic for the k-median problem and the facility location problem. They proved that local search method can achieve $3 + \epsilon$ approximation in polynomial time for the k-median problem. Gupta and Tangwongsan [Gupta and Tangwongsan, 2008] gave a simpler analysis for the k-median problem. As for the k-means problem, Kanungo et al. [Kanungo *et al.*, 2004] proposed the first local search algorithm with ratio $(9 + \epsilon)$. Friggstad et al. [Friggstad *et al.*, 2019] proved that local search yields a PTAS for the k-means problem in doubling metrics. Local search strategy can be extended to different variants of clustering problems, including clustering with outliers, clustering with penalties, individual fair k-clustering, capacitated clustering, etc. [Angel *et al.*, 2015; Zhang *et al.*, 2019; Mahabadi and Vakilian, 2020; Gupta *et al.*, 2017].

2 Preliminaries

Throughout the paper, we use $P = \{p_1, p_2, ..., p_n\}$ and k to denote the given data set and the number of clusters, respectively. We use C^* to denote an optimal clustering center set for a given k-means instance (P, k). For two points p, q, let $d(p, q)$ be their squared distance. For a point p and a center set C, we also use $d(p, C) = \min_{c \in C} d(p, c)$ to denote the squared distance from p to the closest center of p in C. Given a center set C, we use $\Delta(P, C) = \sum_{p \in P} d(p, C)$ to denote the clustering cost induced by C. The goal of clustering is to find a set C of k centers such that the following objective function is minimized: $\Delta(P, C) = \sum_{p \in P} d(p, C)$.

For each point $c_h \in C$, we use P_h to denote the cluster constructed by c_h. For each point $c_h^* \in C^*$, we use P_h^* to denote the optimal cluster constructed by c_h^*. For an optimal center $c_h^* \in C^*$, we denote $s_{c_h^*} = \arg\min_{c_h \in C} d(c_h^*, c_h)$ as its closest point in C. For a center $c_h \in C$, we denote $o_{c_h} = \arg\min_{c_h^* \in C} d(c_h, c_h^*)$ as its closest point in C^*.

The theoretical guarantee of our algorithm is based on **Assumption 1**. We note that the proposed data distribution assumption is very reasonable in practice and has been used for clustering analysis in [Ding and Huang, 2021; Meyerson *et al.*, 2004]. Given an instance (P, k) of the k-means problem, we assume that each optimal cluster should be large enough, i.e. the minimum optimal cluster size is larger than $\frac{\beta n}{k}$ for a given constant β between 0 and 1. Otherwise, for any optimal cluster with cluster size smaller than $\frac{\beta n}{k}$, as pointed out in [Meyerson *et al.*, 2004], the points in this cluster can be regarded as outliers.

Assumption 1. Given an instance (P, k) of the k-means problem, and a constant $\beta \in (0, 1]$, we say (P, k) is β-average if the smallest optimal cluster has size at least $\frac{\beta n}{k}$.

3 The Proposed FLS Algorithm

The general idea of our algorithm is as follows. For an instance (P, k) of the k-means problem, initial center set C can be constructed using LS++ seeding [Lattanzi and Sohler, 2019] with $O(k)$ rounds of swaps. Then, we apply nearest neighbor search to find improvements. In each round of nearest neighbor search, we only use the nearest 10 points of each center in C to construct candidate swap pairs and there are only $O(k)$ swap pairs. If one of the swap pairs induces an $O(\frac{1}{k})$ reduction on current clustering cost, such swap is conducted. If nearest neighbor search fails to make improvements on clustering cost, it indicates that the solution reaches a local optimum around the center set. In this case, random sampling is used to find a set Q of $O(k \log \frac{1}{\eta})$ points in P for a given parameter η. Swap pairs are constructed between Q and C. If one of the swap pairs induces an $O(\frac{1}{k})$ cost reduction, it means that by applying sampling, we can jump out of the local optimum. Under **Assumption 1**, it can be proved that sampling $O(k \log \frac{1}{\eta})$ points for swap pairs construction makes a swap success with probability at least $1 - \eta$. Once nearest neighbor search and sampling methods both fail, we conduct a random mutation, where each center point is swapped with an arbitrary point in $P \backslash C$ with probability $\frac{1}{k}$. Thus, one center is changed in expectation in a single mutation round. Parameter R is used to control the total rounds of swaps, and parameter T is the given execution time (measured by seconds) such that FLS can execute at most T seconds. The specific FLS algorithm is given in **Algorithm 4**.

3.1 Data Structure

In order to conduct nearest neighbor search of the FLS algorithm quickly, we use the selection algorithm [Blum *et al.*, 1973] such that the 10-nearest neighbors of each center in the current center set C of size k can be found in time $O(nkd)$. During the swap of local search, one crucial step is to recalculate the cost after swap when replacing an original center with a new one. For each point $p \in P$, the new distance from p to the modified center set C should be calculated. Thus, during the local search process, we always maintain the closest and the second closest centers to each data point $p \in P$. In this way, the cost change can be calculated more efficiently by checking whether the closest center

Algorithm 1 LS++-Seeding

Input: An instance (P, k) of the k-means problem
Output: A center set C of size k

Initialize center set C using k-means++.
for $i = 1, 2, ..., O(k)$ **do**
  Sample a point $p \in P$ according to probability $\frac{d(p,C)}{\Delta(P,C)}$.
  if $\exists\ q \in C$ such that $\Delta(P, C\setminus\{q\}\cup\{p\}) < (1 - \frac{1}{100k})\Delta(P, C)$ **then**
    Let $q \in C$ be the point such that $\Delta(P, C\setminus\{q\}\cup\{p\})$ is minimized.
    $C = C\setminus\{q\}\cup\{p\}$.
  end if
end for
return C.

Algorithm 2 Search-Swap

Input: An instance (P, k) of the k-means problem, a center set C of size k
Output: A center set C_1 of size k

Set $C_1 = C$.
Search for the 10-nearest neighbors for each $c \in C$. For each center $c \in C$, let N_c denote the neighborhood of c.
Construct swap pairs set $G = \{(c, j) : c \in C, j \in N_c\}$.
if $\exists\ (c, j) \in G$ such that $\Delta(P, C\setminus\{c\}\cup\{j\}) < (1 - \frac{1}{100k})\Delta(P, C)$ **then**
  Let (c, j) be the swap pair such that $\Delta(P\setminus\{c\}\cup\{j\})$ is minimized.
  $C_1 = C\setminus\{c\}\cup\{j\}$.
end if
return C_1.

is swapped out. In each iteration, it takes time $O(ndk)$ to find the closest and the second closest centers of each data point, since there may exist some heavy clusters with size $\Omega(n)$. Thus, LS++-Seeding and Search-Swap algorithms take time $O(ndk^2)$ and $O(ndk)$, respectively. Sampling-Swap takes time $O(ndk^3 \log \frac{1}{\epsilon})$, and random mutation takes time $O(k)$. Since λ, η, R are all constants, the total running time of our algorithm is $O(ndk^3 \log \frac{1}{\epsilon})$.

3.2 Analysis

Now we will give the analysis of Algorithm 4. Let $\{P_1^*, ..., P_k^*\}$ and $\{c_1^*, ..., c_k^*\}$ be the sets of optimal clusters and the corresponding clustering centers, respectively. We define $R(j) = \left\{p \in P_j^* : d(p, c_j^*) \le 2\frac{\Delta(P_j^*, \{c_j^*\})}{|P_j^*|}\right\}$ for each optimal cluster P_j^* as the set of points close to its clustering center c_j^*. We show that $R(j)$ takes a large fraction of P_j^*.

Lemma 1. *Let P_j^* be an arbitrary optimal cluster and $T(j) = \left\{p \in P_j^* : d(p, c_j^*) \le \alpha\frac{\Delta(P_j^*, \{c_j^*\})}{|P_j^*|}\right\}$ be the set of points within P_j^* that are close to the optimal center c_j^*, then it holds that $|T(j)| \ge (1 - \frac{1}{\alpha})|P_j^*|$*

Algorithm 3 Sampling-Swap

Input: An instance (P, k) of the k-means problem, a center set C of size k, parameters ϵ, λ and η
Output: A center set C_1 of size k

Set $C_1 = C$.
for $i = 1, 2, ..., O(\frac{k}{1-\eta}\log\frac{1}{\epsilon})$ **do**
  Randomly and uniformly sample a set $S \subseteq P$ with size $\frac{k}{\lambda}\log\frac{1}{\eta}$.
  Construct candidate swap pairs set $G = \{(c, j) : c \in C, j \in S\}$.
  if $\exists\ (c, j) \in G$ such that $\Delta(P, C\setminus\{c\}\cup\{j\}) < (1 - \frac{1}{100k})\Delta(P, C)$ **then**
    Let (c, j) be the swap pair such that $\Delta(P\setminus\{c\}\cup\{j\})$ is minimized.
    $C_1 = C\setminus\{c\}\cup\{j\}$.
    return C_1.
  end if
end for
return C_1.

Algorithm 4 FLS

Input: An instance (P, k) of k-means, parameters ϵ, λ, η, R, T.
Output: A center set C_1 of size k

Initialize C by calling algorithm LS++-Seeding and set $C_1 = C$.
Set $r = 0$.
while $r < R$ or execution time does not exceed T **do**
  Set $r = r + 1$.
  if Search-Swap induces clustering cost reduction **then**
    $C =$ Search-Swap(P, k, C).
  else if Sampling-Swap induces clustering cost reduction **then**
    $C =$ Sampling-Swap$(P, k, C, \epsilon, \lambda, \eta)$
  else
    Let C_1 be the center set with minimum clustering cost between C and C_1.
    for each $c \in C$ **do**
      For each $c \in C$, with probability $\frac{1}{k}$, replace c with an arbitrary point in $P\setminus C$.
    end for
  end if
end while
return C_1.

Proof.

$$\Delta(P_j^*, \{c_j^*\}) \ge \Delta(P_j^*\setminus T(j), \{c_j^*\}) = \sum_{p\in P_j^*\setminus T(j)} d(p, c_j^*)$$
$$\ge \left(1 - \frac{|T(j)|}{|P_j^*|}\right)|P_j^*|\frac{\alpha\Delta(P_j^*, \{c_j^*\})}{|P_j^*|},$$

which implies that $T(j) \ge (1 - \frac{1}{\alpha})|P_j^*|$. □

For nearest neighbor search process in Algorithm 2, it seeks to locally improve the solution round by round. Such heuristic is fast and efficient in practice since the searching

process only guarantees local optimum. In this searching step, the algorithm focuses more on finding stable local solutions instead of picking points that make clustering cost reduced sharply. Once nearest neighbor search gets stuck, clustering structure needs to be adjusted greatly, where we move to the random sampling part.

Assume that (P, k) is an instance of the k-means problem. Let C denote the current center set and C^* denote a set of optimal centers. Follow the work of Lanttanzi and Sohler [Lattanzi and Sohler, 2019], we give the definition of good cluster with respect to C^* as follows.

Definition 1. *A cluster P_h^* is called good, if there exists a pair of points (c_j, c_h^*) such that $c_h^* \in C^*$ and $c_j \in C$ and $\Delta(P_h^*, C) - \Gamma(P, C, c_j, c_h^*) - 9\Delta(P_h^*, \{c_h^*\}) > \frac{1}{100k}\Delta(P, C)$, where $\Gamma(P, C, c_j, c_h^*) = \Delta(P \backslash P_h^*, C \backslash \{c_j\}) - \Delta(P \backslash P_h^*, C)$ represents the reassignment cost after swapping c_j out.*

The above definition estimates the cost change of replacing c_j with a point close to c_h^*. Then, we want to show that with good probability, we can sample a point q close to c_h^* for swap to reduce the current cost by a factor of $O(\frac{1}{k})$. This prevents the algorithm from getting stuck too early.

For each center $c_h^* \in C^*$, let $s_{c_h^*}$ denote its closest center in C. For simplicity, we say that center c_h^* is captured by $s_{c_h^*}$. For a center $c_j \in C$, let $Z(c_j)$ be the set of centers in C^* captured by c_j. If $|Z(c_j)| = 0$, we say that c_j is a lonely center. Let c_h^* be an optimal center such that $s_{c_h^*} = c_j$. If $|Z(c_j)| = 1$, then (c_j, c_h^*) forms a matched swap pair. If $|Z(c_j)| > 1$, for each $c_h^* \in Z(c_j)$, we find a lonely center $c_q \in C$ to make (c_q, c_h^*) a matched swap pair. During the swap pair construction, each lonely center must be used at most twice. Hence, each optimal center $c_h^* \in C^*$ is matched with a center $c_j \in C$ such that each c_j is used at most twice. For a point $p \in P$, let o_p and s_p denote its closest center in C^* and C, respectively. The following lemma gives an upper bound of reassignment cost induced by a matched swap pair.

Lemma 2. *Let (c_j, c_h^*) be a matched pair. Then $\Gamma(P, C, c_j, c_h^*) \le (4 + \frac{2}{\lambda})\Delta(P_j, C^*) + 2\lambda\Delta(P_j, C)$.*

Proof.

$$\begin{aligned}
\Gamma(P, C, c_j, c_h^*) &\le \sum_{p \in P_j \backslash P_h^*} d(p, s_{o_p}) - d(p, s_p) \\
&\le \sum_{p \in P_j \backslash P_h^*} \left(\sqrt{d(p, o_p)} + \sqrt{d(o_p, s_{o_p})}\right)^2 - d(p, s_p) \\
&\le \sum_{p \in P_j \backslash P_h^*} \left(\sqrt{d(p, o_p)} + \sqrt{d(o_p, s_p)}\right)^2 - d(p, s_p) \\
&\le \sum_{p \in P_j \backslash P_h^*} 4d(p, o_p) + 2\sqrt{\frac{2}{\lambda}}\sqrt{2\lambda d(p, o_p) d(p, s_p)} \\
&\le \sum_{p \in P_j \backslash P_h^*} (4 + \frac{2}{\lambda}) d(p, o_p) + 2\lambda d(p, s_p) \\
&\le (4 + \frac{2}{\lambda})\Delta(P_j, C^*) + 2\lambda\Delta(P_j, C).
\end{aligned}$$

Note that the second and fourth steps follow from the triangle inequality, the third step follows from the fact that s_{o_p} is the nearest point to o_p in C, and the fifth step follows from Cauchy Inequality. □

In the following, we analyze that if current clustering cost $\Delta(P, C)$ is large enough, there must exist at least one good cluster.

Lemma 3. *If $\Delta(P, C) > 100OPT$, then there must exist at least one good cluster induced by some matched pair (c_j, c_h^*)*

Proof. Assume that there is no good cluster. For each $c_h^* \in C^*$, denote its matched swap pair as (c_{m_h}, c_h^*). By the definition of good cluster and Lemma 2, it holds that

$$\begin{aligned}
\Delta(P_h^*, C) &\le \Gamma(P, C, c_{m_h}, c_h^*) + 9\Delta(P_h^*, \{c_h^*\}) \\
&\quad + \frac{1}{100k}\Delta(P, C) \\
&\le (4 + \frac{2}{\lambda})\Delta(P_{m_h}, C^*) + 2\lambda\Delta(P_{m_h}, C) \\
&\quad + 9\Delta(P_h^*, \{c_h^*\}) + \frac{1}{100k}\Delta(P, C).
\end{aligned}$$

Thus, by summing up all matched pairs, we have

$$\begin{aligned}
&\sum_{c_h^* \in C^*} \Delta(P_h^*, C) = \Delta(P, C) \\
&\le \sum_{c_h^* \in C^*} (4 + \frac{2}{\lambda})\Delta(P_{m_h}, C^*) + 2\lambda\Delta(P_{m_h}, C) \\
&+ 9\Delta(P_h^*, \{c_h^*\}) + \frac{1}{100k}\Delta(P, C) \\
&\le (2 \times (4 + \frac{2}{\lambda}) + 9)OPT + (4\lambda + \frac{1}{100})\Delta(P, C).
\end{aligned}$$

Let $\lambda = 0.12375$. Then, we have $\Delta(P, C) \le 100OPT$. □

According to Lemma 3, if the local optimum after nearest neighbor search has clustering cost larger than $100OPT$, there exists at least one good cluster P_h^* induced by a matched pair (c_j, c_h^*). In the following, we argue that by random sampling $O(\frac{k}{\beta}\log\frac{1}{\eta})$ points from P for swap pairs construction, with probability at least $1 - \eta$, the clustering cost can be reduced by a factor of $O(\frac{1}{k})$.

Lemma 4. *With probability at least $1 - \eta$, by sampling $O(\frac{k}{\beta}\log\frac{1}{\eta})$ points randomly from P, the sampled set contains at least one point from $R(h)$ for an optimal center c_h^*.*

Proof. For an optimal cluster P_h^*, since $|P_h^*| \ge \frac{\beta n}{k}$, by Lemma 1 and the definition of $R(h)$, we know that $|R(h)| \ge \frac{\beta n}{2k}$. Let $\zeta = \frac{|R(h)|}{|P|}$. If we randomly sample a set V of points from P, the probability that V contains at least one point from $R(h)$ is at least $1 - (1 - \zeta)^{|R(h)|}$. If the probability that at least one point from $R(h)$ is sampled is at least $1 - \eta$, then $1 - (1 - \zeta)^{|V|} \ge 1 - \eta$. Thus, V has size at least $\frac{\log\frac{1}{\eta}}{\log\frac{1}{1-\zeta}} \le \frac{1}{\zeta}\log\frac{1}{\eta}$. Since $\zeta = \frac{|R(h)|}{|P|} \ge \frac{\beta}{2k}$, if $|V| \ge \frac{2k}{\beta}\log\frac{1}{\eta}$, V must contain at least one point from $R(h)$ with probability at least $1 - \eta$. □

By Lemma 4, we know that by sampling $O(\frac{k}{\beta}\log\frac{1}{\eta})$ points from P, there exists at least one point q close to the optimal center c_h^* in the sampled set, where (c_j, c_h^*) induces a good

Datasets	Number of clusters	Sample size	Dimension	β
A2	35	5250	2	0.9467
A3	50	7500	2	0.9256
Dim32	16	1024	32	1
Birch2	100	100000	2	0.997
Unbalance	8	6500	2	0.1231
Yeast	10	1048	8	0.0337
Covertype	7	581012	54	-
SUSY	10	5000000	18	-

Table 1: Datasets.

cluster. By swapping q with c_j, we will show that the current clustering cost will be reduced by a factor of $O(\frac{1}{k})$. Observe that

$$\begin{aligned}
&\Delta(P, C \cup \{q\} \setminus \{c_j\}) = \Delta(P, C) - (\Delta(P, C) \\
&- \Delta(P, C \cup \{q\} \setminus \{c_j\})) \\
&\leq \Delta(P, C) - (\sum_{p \in P \setminus P_h^*} d(p, C) + \sum_{p \in P_h^*} d(p, C) \\
&- \sum_{p \in P \setminus P_h^*} d(p, C \setminus \{c_j\}) - \sum_{p \in P_h^*} d(p, q)) \\
&= \Delta(P, C) - (\Delta(P_h^*, C) - \Gamma(P, C, c_j, c_h^*) - \Delta(P_h^*, \{q\})) \\
&\leq \Delta(P, C) - (\Delta(P_h^*, C) - \Gamma(P, C, c_j, c_h^*) - 6\Delta(P_h^*, \{c_h^*\})) \\
&\leq (1 - \frac{1}{100k})\Delta(P, C),
\end{aligned}$$

where the second inequality follows from the fact that $q \in R(h)$ is within distance $\frac{2\Delta(P_h^*, \{c_h^*\})}{|P_h^*|}$ to c_h^* and triangle inequality, and the last step follows from the definition of good cluster. Hence, following the results in [Rozhoň, 2020], by repeating the sampling process $O(k \log \frac{1}{\epsilon})$ times, clustering cost can be reduced to $(100OPT + \epsilon)$ in expectation.

4 Experiments

In this section, we compare our algorithm with other local search based methods for the k-means problem.

Datasets. We use the k-means benchmarks [Fränti and Sieranoja, 2018] and other datasets from UCI Machine Learning Repository[2] to conduct our experiments. The datasets used in our experiments are summarized in Table 1. For any instance of the k-means problem, the β value in **Assumption 1** for an instance is defined as the ratio between the minimum optimal cluster size and the average cluster size $\frac{n}{k}$ using ground truth provided by benchmarks. It is easy to see that as long as the β value of given instance is between 0 and 1, then the instance satisfies **Assumption 1**. If there is no provided ground truth, we use linear programming solver Gurobi to calculate an optimal solution. Based on those "hard instances" defined in [Fränti and Sieranoja, 2018], we test whether different local search algorithms can find good solutions. Fränti and Sieranoja [Fränti and Sieranoja, 2018] tested different k-means datasets, and defined "hard instances" as those with high separation, large number of clusters or unbalanced distribution. "Hard instances" include the instances in datasets A2, A3, Dim32, Birch2 and Unbalance. We also compare the scalability of different local search algorithms on large datasets such as Covertype and SUSY, which contains 581012 and 5000000 points, respectively.

Algorithms and Parameters. In our experiment, we consider three algorithms: our algorithm described in Algorithm 4, LS++ proposed in [Lattanzi and Sohler, 2019] and LS proposed in [Kanungo *et al.*, 2004], which are summarized as follows.

- **LS.** This is the local search method proposed by Kanungo et al. [Kanungo *et al.*, 2004], which searches swap pairs on whole given dataset and performs swaps iteratively until convergence.
- **LS++.** This is the algorithm proposed by Lattanzi and Sohler [Lattanzi and Sohler, 2019], which uses D^2-sampling in each round to sample a point and construct a candidate swap pair set of size k.
- **FLS.** This is our algorithm described in Algorithm 4. Parameters λ, η are set to be 0.5 and 0.5, respectively. For the sampling process in Algorithm 3, we only repeat the sampling for 3 times.

Experimental Setup. We run all the above algorithms on each dataset 5 times and give the average results. All the algorithms use the same initial center set obtained by 25 rounds of LS++. During the local search process, we also study the parallelization of search and swap. In each round of search and swap, the constructed swap pairs can be parallelized for calculation using multiple cores and the one with the maximum cost reduction is taken as the final swap pair. LS requires high computing resources since it constructs $O(nk)$ swap pairs in each round while the upper bound of parallelization for LS++ is k cores. Our algorithm constructs $O(k^2)$ swap pairs which can make full use of computing resources in practice. In experiments, we only use k cores for parallelization for the sake of fairness since LS++ only contains k swap pairs in each round. For hardware, we use machines with 72 Intel Xeon Gold 6230 CPUs.

Methodology. In our experiments, we run each algorithm for 5 times with fixed execution time and give the gaps between costs of solutions and corresponding optimal clustering costs. Gap is defined as the clustering cost relative to the optimal clustering cost as $gap = \sum_{i=1}^{5} \frac{1}{5} \frac{S_i - OPT}{OPT}$, where S_i

[2]https://archive.ics.uci.edu/ml/index.php

Dataset	Method	Gap(%)	Time point	Gap(%)	Time point	Gap(%)	Time point	Dataset	Method	Gap(%)	Time point	Gap(%)	Time point	Gap(%)	Time point
A2	LS	89.39	60s	89.39	120s	89.39	180s	**Unbalance**	LS	27.84	60s	27.84	120s	27.84	180s
	LS++	6.35		4.87		4.03			LS++	5.92		2.86		2.68	
	FLS	**0.97**		**0.93**		**0.56**			FLS	**1.28**		**0.88**		**0.88**	
A3	LS	91.19	60s	91.19	120s	91.19	180s	**Yeast**	LS	13.03	60s	13.03	120s	13.03	180s
	LS++	**11.74**		7.56		6.64			LS++	1.59		1.59		0.86	
	FLS	15.99		**0.96**		**0.91**			FLS	**0**		**0**		**0**	
Dim32	LS	27.83	20s	27.09	40s	25.22	60s	**Covertype**	LS	20.81	2400s	20.81	3000s	20.81	3600s
	LS++	0.79		0.22		0.22			LS++	6.11		3.82		3.82	
	FLS	**0**		**0**		**0**			FLS	**4.59**		**1.75**		**0**	
Birch2	LS	274.97	3600s	274.12	4800s	274.12	7200s	**SUSY**	LS	30.79	12000s	30.79	18000s	30.79	24000s
	LS++	**13.99**		12.77		11.49			LS++	**6.23**		4.52		1.98	
	FLS	16.62		**10.74**		**10.46**			FLS	10.94		**2.76**		**0**	

Table 2: Comparisons of clustering costs with fixed execution time.

Method	A2		A3		Dim32		Birch2		Unbalance		Yeast		Covertype		SUSY	
	Gap(%)	Time(s)	Gap(%)	Time(s)	Gap(%)	Time(s)	Gap(%)	Time(s)	Gap(%)	Time(s)	Gap(%)	Time(s)	Gap(%)	Time(s)	Gap(%)	Time(s)
LS		NA		NA		NA		NA		NA		NA		NA		NA
LS++	0	NA	0	NA	0	NA	0	NA	0	NA	0	NA	0	NA	0	NA
FLS		NA		NA		**2**		NA		NA		**18**		**3517**		**20732**
LS		NA		NA		NA		NA		NA		NA		NA		NA
LS++	1	NA	1	NA	1	16	1	NA	1	NA	1	128	1	NA	1	NA
FLS		**54**		**119**		**2**		NA		**111**		**3**		**3374**		**18975**
LS		NA		NA		NA		NA		NA		NA		NA		NA
LS++	2	NA	2	NA	1	8	2	NA	2	NA	2	35	2	NA	2	23535
FLS		**53**		**117**		**1**		NA		**42**		**1**		**2933**		**18466**
LS		NA		NA		NA		NA		NA		NA		NA		NA
LS++	3	NA	3	NA	3	4	3	NA	3	116	3	32	3	NA	3	23174
FLS		**52**		**116**		**1**		NA		**13**		**1**		**2787**		**17636**
LS		NA		NA		NA		NA		NA		NA		NA		NA
LS++	4	NA	4	NA	4	4	4	NA	4	96	4	29	4	2598	4	22179
FLS		**52**		**115**		**1**		NA		**11**		**1**		**2519**		**17359**
LS		NA		NA		NA		NA		NA		NA		NA		NA
LS++	5	101	5	NA	5	3	5	NA	5	67	5	28	5	2498	5	16553
FLS		**51**		**114**		**1**		NA		**10**		**1**		**2396**		**16181**
LS		NA		NA		NA		NA		NA		NA		NA		NA
LS++	6	78	6	NA	6	3	6	NA	6	16	6	4	6	2465	6	**12526**
FLS		**51**		**114**		**1**		NA		**9**		**1**		**2200**		16097
LS		NA		NA		NA		NA		NA		NA		NA		NA
LS++	7	55	7	159	7	2	7	NA	7	14	7	4	7	**1154**	7	**10438**
FLS		**50**		**114**		**1**		NA		**8**		**1**		2054		15720

Table 3: Comparisons of running time to reach fixed costs.

is the clustering cost of solutions returned by different algorithms in the i-th experiment. For large datasets, since optimal solutions cannot be found by solver, we use the solution with the minimum cost among LS, LS++ and FLS as the ground truth solution, and compute the differences between the ground truth solution and other solutions. To compare the running time, we give the time needed to reach different gaps for each algorithm in Table 3.

4.1 Results

Table 2 compares the clustering costs of LS, LS++ and FLS at several different time points. On small datasets, our algorithm finds optimal solutions on two of the datasets, while LS++ fails to find any optimal solution. LS++ performs well at the beginning. However, for larger execution time, LS++ is easy to get stuck in local optimum. It is more and more difficult for LS++ to find improvements as the gap between clustering cost and optimal clustering cost narrows. As for LS, clustering costs hardly change since LS constructs $O(nk)$ swap pairs for enumeration in each round, and it is impossible for LS to find good solutions in fixed execution time. Solutions returned by our algorithm are very close to the optimal clustering costs on small datasets, and there is only a gap ranging from 0% to 0.88%. On "hard instances", our algorithm improves the performance of LS++ from 0.22% to 5.73% on clustering costs. On large datasets, our algorithm scales well and achieves the minimum final clustering costs, which improves the performance of LS++ from 1.03% to 3.82% on clustering costs.

Table 3 summarizes the time spent to reach a fixed clustering cost. FLS finds four datasets with gap 0% while LS++ fails to find any of them. When the quality of solution is high, for example the gap to optimal solutions ranges from 0% to 2%, FLS can find high-quality solutions quickly. On the contrary, LS++ fails to find high-quality solutions on most datasets. It takes LS++ nearly forty and eight times, compared to ours, to find solutions with gap 1% on datasets Yeast and Dim32, respectively. From Table 3, we can get that our algorithm FLS is faster to find near-optimal solutions (solutions with small gaps to optimal solutions) than LS++.

5 Conclusion

In this paper, we propose a fast local search algorithm for k-means. By developing new searching techniques and random mutation, the search space of local search in each round is reduced greatly while the quality of solution is maintained. Under "average" assumption of data distribution in practice, we prove that our algorithm returns a constant factor approximate solution in expectation. Finally, we show experimentally that our algorithm achieves better performance on both small and large datasets compared with other local search methods.

Acknowledgments

This work was supported by National Natural Science Foundation of China (Grant Nos. 62172446, 61872450), 111 Project (Grant No. B18059).

References

[Angel *et al.*, 2015] Eric Angel, Nguyen Kim Thang, and Damien Regnault. Improved local search for universal facility location. *Journal of Combinatorial Optimization*, 29:237–246, 2015.

[Arthur and Vassilvitskii, 2007] 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. SIAM, 2007.

[Arya *et al.*, 2004] Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33:544–562, 2004.

[Blum *et al.*, 1973] Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7:448–461, 1973.

[Choo *et al.*, 2020] Davin Choo, Christoph Grunau, Julian Portmann, and Václav Rozhon. k-means++: Few more steps yield constant approximation. In *International Conference on Machine Learning*, pages 1909–1917. PMLR, 2020.

[Ding and Huang, 2021] Hu Ding and Jiawei Huang. Is simple uniform sampling efficient for center-based clustering with outliers: when and why? *arXiv preprint arXiv:2103.00558*, 2021.

[Fränti and Sieranoja, 2018] Pasi Fränti and Sami Sieranoja. K-means properties on six clustering benchmark datasets. *Applied Intelligence*, 48:4743–4759, 2018.

[Friggstad *et al.*, 2019] Zachary Friggstad, Mohsen Rezapour, and Mohammad R Salavatipour. Local search yields PTAS for k-means in doubling metrics. *SIAM Journal on Computing*, 48:452–480, 2019.

[Gupta and Tangwongsan, 2008] Anupam Gupta and Kanat Tangwongsan. Simpler analyses of local search algorithms for facility location. *arXiv preprint arXiv:0809.2554*, 2008.

[Gupta *et al.*, 2017] Shalmoli Gupta, Ravi Kumar, Kefu Lu, Benjamin Moseley, and Sergei Vassilvitskii. Local search methods for k-means with outliers. *Proceedings of the VLDB Endowment*, 10:757–768, 2017.

[Kanungo *et al.*, 2004] 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*, 28:89–112, 2004.

[Lattanzi and Sohler, 2019] Silvio Lattanzi and Christian Sohler. A better k-means++ algorithm via local search. In *International Conference on Machine Learning*, pages 3662–3671. PMLR, 2019.

[Lloyd, 1982] Stuart Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28:129–137, 1982.

[Mahabadi and Vakilian, 2020] Sepideh Mahabadi and Ali Vakilian. Individual fairness for k-clustering. In *International Conference on Machine Learning*, pages 6586–6596. PMLR, 2020.

[Meyerson *et al.*, 2004] Adam Meyerson, Liadan O'callaghan, and Serge Plotkin. A k-median algorithm with running time independent of data size. *Machine Learning*, 56:61–87, 2004.

[Rozhoň, 2020] Václav Rozhoň. Simple and sharp analysis of k-means||. In *International Conference on Machine Learning*, pages 8266–8275. PMLR, 2020.

[Zhang *et al.*, 2019] Dongmei Zhang, Chunlin Hao, Chenchen Wu, Dachuan Xu, and Zhenning Zhang. Local search approximation algorithms for the k-means problem with penalties. *Journal of Combinatorial Optimization*, 37:439–453, 2019.