Survey and Proposal of SOTA Algorithms for the Metric k-Center Problem

Zehua Wang
wangzehu24@mails.tsinghua.edu.cn
2024011332

January 7, 2025

Abstract

This survey provides a comprehensive examination of the metric k-center problem, focusing on both approximation and heuristic algorithms. We introduce three novel state-of-the-art heuristic algorithms: LS Scr, SA Swp, and SA Swp Scr. Notably, the SA Swp Scr algorithm achieves an impressive empirical approximation ratio of 1.049 with an efficient $O(n^2 \log n + C_0 kn)$ running time. To assess their practical performance, we implemented all discussed algorithms within a unified framework and conducted extensive experiments. Our findings highlight the strengths and limitations of each algorithm, offering valuable insights for practitioners. The complete implementation and datasets are available at `https://github.com/patrickwzh/k-Center_Evaluation`.

1 Introduction

In the era of expanding artificial intelligence, the volume of generated data is unprecedented. Clustering, a fundamental technique in machine learning, groups data into clusters based on feature similarities [8]. It is crucial in managing and analyzing large datasets in applications such as customer segmentation [27], gene expression analysis [17], and image processing [33].

Among clustering problems, the metric k-center problem is significant. Given a set of points in a metric space, the objective is to select k centers to minimize the maximum distance from any point to its nearest center. This problem has applications in:

- **Network Design:** Optimizing server or router placement to minimize maximum communication delay [15].
- **Facility Location:** Determining optimal locations for facilities like warehouses or service centers [34].
- **Clustering in Machine Learning:** Grouping data points into clusters with bounded diameters for tasks like anomaly detection and pattern recognition [14].

Exact algorithms, such as branch and bound [30], are computationally infeasible for large datasets due to the problem's NP-hardness [18]. Thus, approximation algorithms and heuristic methods are essential for practical applications.

Our contributions in this survey include:

- A comprehensive examination of the metric k-center problem, including its history, formal definitions, and NP-hardness.
- Discussion of various approximation algorithms, highlighting their methodologies and theoretical guarantees.
- Exploration of heuristic algorithms, focusing on their practical performance and applicability. Unlike approximation algorithms, heuristic algorithms do not provide theoretical approximation ratio bounds but often achieve better empirical performance on real-world datasets.
- Introduction of three novel state-of-the-art heuristic algorithms: LS Scr, SA Swp, and SA Swp Scr. Notably, the SA Swp Scr algorithm achieves an impressive empirical approximation ratio of 1.049 with an efficient $O(n^2 \log n + C_0 kn)$ running time.
- Implementation of a codebase to conduct extensive experiments evaluating the empirical approximation ratios and running times of the discussed algorithms on benchmark datasets. The codebase is available at `https://github.com/patrickwzh/k-Center_Evaluation`.
- Insights into the strengths and limitations of each algorithm, guiding practitioners in selecting appropriate methods for real-world applications.

2 History

The metric k-center problem was first proposed by Hakimi in 1964 [13]. In 1979, Kariv and Hakimi [18] proved the problem is NP-hard by showing its relationship with the Dominating Set problem. Specifically, they demonstrated a reduction from Dominating Set to the vertex k-center problem, establishing its computational difficulty.

In the same year, Hsu and Nemhauser [16] proved that no polynomial-time algorithm can achieve an approximation factor better than 2 for the metric k-center problem unless $P = NP$.

In 1985, Gonzalez [12], Dyer and Frieze [5], and Hochbaum and Shmoys [15] independently developed 2-approximation algorithms for the metric k-center problem. These algorithms provide the best possible approximation factor achievable in polynomial time. Hochbaum and Shmoys also introduced an algorithm that uses a binary search technique to find the minimal covering radius, known as the HS Algorithm [15].

In 1995, David Shmoys proposed a simplified greedy algorithm, known as the Sh Algorithm [32], which also achieves a 2-approximation factor by selecting centers based on a proposed covering radius.

Despite achieving the optimal approximation ratio, these algorithms often underperform on practical datasets compared to heuristic methods. To address this, researchers have developed heuristic algorithms that aim to provide better empirical results.

The Gr Algorithm, a pure greedy approach, selects centers iteratively to minimize the current make-span but lacks theoretical guarantees [29].

In 2005, Mihelič and Robič introduced the Scr Algorithm [25], a heuristic based on the "lazy" principle for constructing dominating sets, which can be adapted to the metric k-center problem.

In 2017, García-Díaz et al. proposed the Critical Dominating Set (CDS) Algorithm [11], a 3-approximation algorithm that, despite its higher theoretical bound, outperforms previous approximation algorithms in practice. They also introduced heuristic versions, the CDSh and CDSh+

algorithms, which improve running time and practical performance by incorporating heuristics into the CDS framework.

Over the years, numerous extensions and special settings of the k-center problem have been investigated, including:

- **Capacitated k-Center:** Introduces capacity constraints to the centers, limiting the number of points each center can cover [20].
- **Fault-Tolerant k-Center:** Ensures the solution remains effective even in the presence of center failures [19].
- **Dynamic k-Center:** Addresses scenarios where data points or the number of centers change over time [2].
- **Asymmetric k-Center:** Considers cases where the distance from point A to point B differs from the distance from B to A [28].
- **k-Center with Outliers:** Incorporates outliers that do not need to be covered by the centers [3, 22].
- **k-Center in Euclidean Space:** Focuses on the Euclidean metric space, leveraging its geometric properties to achieve a better theoretical approximation ratio of $(\sqrt{7}+1)/2 \approx 1.82$ [9].

These developments highlight the ongoing efforts to improve both the theoretical and practical aspects of the metric k-center problem. In this survey, we focus on the generic metric k-center problem in a general metric space, discussing both approximation and heuristic algorithms, including the Gon, Sh, HS, CDS, Gr, CDSh, CDSh+, and Scr algorithms. In addition, we introduce three novel heuristic algorithms: LS Scr, SA Swp, and SA Swp Scr, which aim to achieve better empirical performance with efficient running times.

3 Preliminaries

3.1 Formal Definition

To formally define the metric k-center problem, we first introduce the concept of a metric space, following Sariel Har-Peled's notes [14].

Definition 3.1 (Metric Space)**.** A *metric space* is a pair $(\mathcal{X}, d)$ where $\mathcal{X}$ is a set and $d : \mathcal{X} \times \mathcal{X} \to [0, +\infty)$ is a metric, satisfying the following axioms:

1. $d(x, y) = 0$ if and only if $x = y$.
2. $d(x, y) = d(y, x)$ for all $x, y \in \mathcal{X}$ (symmetry).
3. $d(x, y) + d(y, z) \geq d(x, z)$ for all $x, y, z \in \mathcal{X}$ (triangle inequality).

Given a metric space $(\mathcal{X}, d)$, we define the metric k-center problem as follows.

Definition 3.2 (Metric k-Center Problem)**.** Given a metric space $(\mathcal{X}, d)$ and an integer k, the objective is to find a center set $C \subseteq \mathcal{X}$ with $|C| = k$ that minimizes the maximum distance from any point in $\mathcal{X}$ to its nearest center in C. This maximum distance is termed the *make-span* of the center set C.

Intuitively, the make-span is the minimum radius r such that placing $|C|$ balls of radius r centered at all points in C covers every point in $\mathcal{X}$.

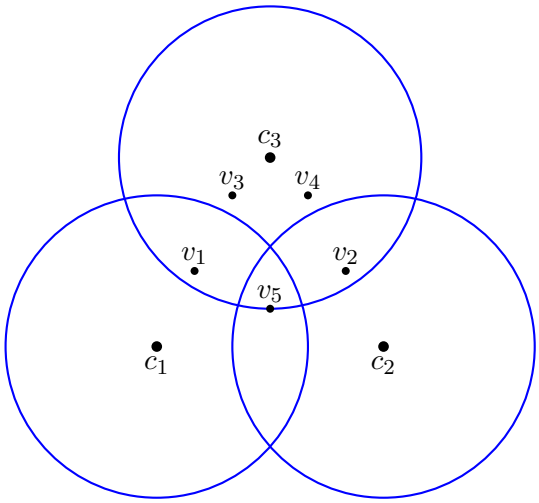


Figure 1: Illustration of the make-span in a 2D plane.

3.2 NP-Hardness of the Metric k-Center Problem

In 1979, Kariv and Hakimi [18] proved that the metric k-center problem is NP-hard via a reduction from the Dominating Set problem. This reduction shows that solving the k-center problem is at least as hard as finding the smallest dominating set in a graph, which is a known NP-hard problem.

Lemma 3.1 (Minimum Dominating Set)**.** *Let $G = (V, E)$ be a graph and k an integer. A subset $S \subseteq V$ is a dominating set of G if for every vertex $v \in V \setminus S$, there exists a vertex $u \in S$ which is a neighbor of v. The decision version of the Dominating Set problem asks whether there exists a dominating set of size at most k. This problem is* NP*-complete.*

Proof. First, Dominating Set is in NP since, given a subset S of size at most k, we can verify in polynomial time whether S is a dominating set by checking that every vertex in $V \setminus S$ has a neighbor in S.

Second, we prove that Vertex Cover $\leq_p$ Minimum Dominating Set. For any Vertex Cover (VC) instance, such as the left graph shown in Figure 2, we transform the instance into a Minimum Dominating Set (MDS) instance by converting each edge into a triangle. The desired size k remains unchanged.

1. If the VC instance is a "yes" instance, then the same set of vertices also forms an MDS:
    - Vertices in the original graph are already connected to a vertex in the set.

- Newly added vertices are connected to both ends of an original edge, where at least one end is in the set.

2. Conversely, if the MDS instance is a "yes" instance for size k, we can assume all vertices in the chosen set are from the original graph since choosing an original vertex is not worse than choosing a newly added vertex in a triangle. Thus, the chosen set also applies to VC.

Overall, there exists a VC of size $\leq k$ if and only if there exists an MDS of size $\leq k$ in the transformed graph. □

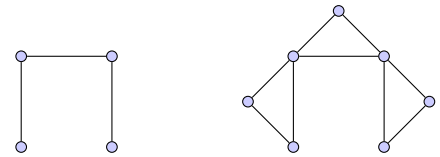


Figure 2: Reduction from Vertex Cover to Minimum Dominating Set

Theorem 3.2. *The metric k-center problem is* NP*-hard.*

Proof. Consider the decision version of the metric k-center problem: given a metric space $(\mathcal{X}, d)$, an integer k, and a distance r, does there exist a center set $C \subseteq \mathcal{X}$ with $|C| = k$ such that $\max_{v \in \mathcal{X}} d(v, C) \leq r$?

We show that this problem is NP-complete by reducing from the MDS problem. For any graph $G = (V, E)$ and integer k, a dominating set of size at most k in G corresponds to a center set of size k in the metric space (V, d), where $d(u, v) = 1$ if $(u, v) \in E$ and $d(u, v) = 2$ otherwise. The distance r is set to 1. Hence MDS $\leq_p$ Metric k-Center (Decision), and the metric k-center problem is NP-hard. □

3.3 Approximation Algorithms and Hardness of Approximation

Wen-Lian Hsu and George L. Nemhauser [16] proved that unless $P = NP$, no algorithm can achieve a better approximation factor than 2 for the metric k-center problem. This result establishes the theoretical limit for approximation algorithms in polynomial time.

Lemma 3.3 (Promised k-Center)**.** *The promised k-center problem is defined as follows. Given a metric space $(\mathcal{X}, d)$ and an integer r, it is promised that the optimal value of the given instance is at most r or at least $1.9999r$. The goal is to decide which case holds. This problem is* NP*-complete.*

Proof. First, the promised k-center problem is in NP since, given a center set C of size k, we can verify in polynomial time whether $\max_{v \in \mathcal{X}} d(v, C) \leq r$.

Second, we show that Minimum Dominating Set $\leq_p$ Promised k-Center. For any graph $G = (V, E)$ and integer k, we construct a metric space $(\mathcal{X}, d)$ as follows:

$$d(u, v) = \begin{cases} r & \text{if } (u, v) \in E, \\ \alpha r & \text{otherwise,} \end{cases}$$

where $1.9999 < \alpha < 2$. The promised k-center instance is a "Yes" instance if and only if there exists a dominating set of size at most k in G. □

Theorem 3.4. *Approximating the metric k-center problem within a factor better than 2 is* NP*-hard.*

Proof. Suppose we have an approximation algorithm with an approximation factor $\alpha < 2$. Consequently, we can use this algorithm to solve the promised k-center problem with α as the promised ratio. □

4 Approximation Algorithms

Several approximation algorithms have been proposed for the metric k-center problem, aiming to find efficient solutions within a guaranteed factor of the optimal solution. In this section, we discuss three prominent algorithms: the Gon Algorithm, the Sh Algorithm, and the HS Algorithm.

4.1 The Gon Algorithm

In 1985, Gonzalez [12] introduced a greedy algorithm that selects the next center as the point farthest from the current set of centers. This algorithm, also discovered independently by Dyer and Frieze [5], is known as the Gon Algorithm.

Algorithm 1 The Gon Algorithm [12]

Input: Set of points $\mathcal{X}$, integer k
Output: Set of k centers C
Randomly select the first center $c_1 \in \mathcal{X}$
$C \leftarrow \{c_1\}$
for $i = 2$ to k **do**
    Select $c_i \in \mathcal{X} \setminus C$ that maximizes $\min_{c \in C} d(c_i, c)$
    $C \leftarrow C \cup \{c_i\}$
end for
return C

This algorithm provides a 2-approximation guarantee.

Definition 4.1 (Cluster)**.** Given a make-span r and a center c, the *cluster* of c is defined as $\{v \mid d(v, c) \leq r\}$.

Theorem 4.1. *The Gon Algorithm is a 2-approximation algorithm for the metric k-center problem [12].*

Proof. Let C denote the set of centers obtained by The Gon Algorithm, and C^* denote the optimal set of centers with a make-span of r.

1. If each cluster of C^* contains only one center from C, then for any vertex v in G, let c^* be the center of the corresponding cluster and c be the corresponding center in C. We have

$$d(v, C) \leq d(v, c) \leq d(v, c^*) + d(c^*, c) \leq r + r = 2r.$$

2. If the condition in (1) is not satisfied, there must be at least one cluster of C^* (with center c^*) containing at least two centers (denoted c_1 and c_2) from C. Without loss of generality, assume c_2 was added after c_1 in C, and let C' be the set of centers before c_2 was added. For any vertex v in G, we have

$$d(v, C) \leq d(v, C') \leq d(c_2, C') \leq d(c_2, c_1) \leq d(c_2, c^*) + d(c^*, c_1) \leq 2r.$$

In both cases, the make-span of C is at most twice the optimal make-span, proving the 2-approximation guarantee. □

Theorem 4.2. *The running time of the Gon Algorithm is $O(kn)$, where n is the number of points in the metric space [12].*

Proof. The algorithm iteratively selects k centers, each requiring $O(n)$ time to find the farthest point. Thus, the total running time is $O(kn)$. □

If we run the Gon Algorithm until it exhausts all vertices, it will generate an ordering of the vertices known as the *greedy permutation*, as discussed in Har-Peled's notes [14].

Definition 4.2 (r-Net)**.** A set of vertices $S \subseteq V$ is an *r-net* of V if the following conditions hold:

1. The make-span of S is at most r, i.e., $\max_{v \in V} d(v, S) \leq r$. (Covering property)
2. No two vertices in S are within distance r of each other, i.e., $\forall u, v \in S, u \neq v, d(u, v) > r$. (Separation property)

Intuitively, an r-net of a set of vertices V is a compact representation of V in the resolution of r. Surprisingly, the greedy permutation of V provides us with such a representation for all resolutions [14].

Theorem 4.3. *Let V be a set of vertices in a finite metric space, and let $P = \langle c_1, c_2, \ldots, c_n \rangle$ be its greedy permutation with the associated sequence of make-spans $\langle r_1, r_2, \ldots, r_n \rangle$. For any i, we have that $C_i = \langle c_1, c_2, \ldots, c_i \rangle$ is an r_i-net of V.*

Proof. Notice that $r_i = d(c_i, C_{i-1})$. Subsequently, for any $j < i$, we have $d(c_j, c_i) \geq r_i$, which implies that C_i satisfies the separation property. The covering property follows from the definition of the make-span. □

Greedy permutations provide a natural ordering for streaming large datasets due to their informative prefixes [7]. These properties enable various applications, including color quantization [35], progressive image sampling [6], selecting landmarks for probabilistic roadmaps in motion planning [23], point cloud simplification [26], halftone mask generation [31], and hierarchical clustering [4].

4.2 The Sh Algorithm

Formally characterized in 1995 by David Shmoys [32], the Sh Algorithm is another greedy approach to the metric k-center problem. This algorithm exploits the properties of the metric space to achieve a 2-approximation guarantee. It takes in a proposed r on what the make-span should be and checks whether the proposition holds. Shmoys [32] designed the following algorithm.

Algorithm 2 The Sh Algorithm [32]

$S' \leftarrow S$ ▷ S' represents the set of points/vertices that still need to be covered
$C \leftarrow \emptyset$
while $S' \neq \emptyset$ **do**
    Select any point $s \in S'$
    $C \leftarrow C \cup \{s\}$
    Remove all points from S' that are at a distance at most $2r$ from s
end while
if $|C| \leq k$ **then**
    return C as the selected set of centers
else
    Declare that there is no set of k centers with covering radius at most r
end if

Theorem 4.4. *If the Sh Algorithm successfully terminates, it provides a 2-approximation solution for the metric k-center problem [32]. Specifically, it yields a solution with a make-span of at most $2r$. If the algorithm declares that no set of k centers with a make-span of at most r exists, then no such set exists.*

Proof. The first part of the theorem follows directly from the algorithm's design. For the second part, assume there exists an optimal set of centers C^* with a make-span of r. For any center $v \in C$, there exists an optimal center $c^* \in C^*$ such that $d(v, c^*) \leq r$ since C^* covers all points. For all vertices u_{c^*} in the cluster of c^*, we have

$$d(v, C) \leq d(v, c) \leq d(v, c^*) + d(c^*, c) \leq r + r = 2r.$$

This implies that the cluster of c^* is entirely covered by the $2r$-ball centered at v. However, there are at most k centers in C^* and more than k centers in C, indicating that some $v \in C$ lacks a corresponding c^* in C^*, contradicting the assumption that C^* covers all points. □

Theorem 4.5. *The running time of the Sh Algorithm is $O(kn)$, where n is the number of points in the metric space [32].*

Proof. The algorithm iteratively selects k centers, each requiring $O(n)$ time to find a point that is at least $2r$ away from the current set of centers. Thus, the total running time is $O(kn)$. □

4.3 The HS Algorithm

In 1985, Hochbaum and Shmoys [15] proposed an algorithm that takes Sh Algorithm as basis.

In this section, we keep the original paper's notation for clarity, i.e., we use an adjacency list to record the distance between points. In addition, assume that $E = \{e_1, e_2, \ldots, e_m\}$ is the set of edges in non-decreasing order of distance, and for each vertex, we have a non-decreasing list (in terms of distance) of adjacent vertices. Let $G_i = (V, E_i)$ where $E_i = \{e_1, e_2, \ldots, e_i\}$.

The HS Algorithm performs a binary search on the make-span r of the optimal solution, using the fact that the make-span corresponds to the distance of an edge.

For each candidate r, the algorithm maintains a set of centers S and an uncovered set T. It iteratively adds points to S and removes their neighbors within a distance of $2r$ from T. The algorithm concludes when the smallest r is found such that the size of S does not exceed k.

Hochbaum and Shmoys [15] proposed the following algorithm.

Algorithm 3 The HS Algorithm [15]

Input: A set V with $|V| = n$, and a set of edges E in non-decreasing weight order.
Output: A set S with $|S| \leq k$.
if $k = |V|$ **then**
    Output V and halt.
end if
$low \leftarrow 1$ ▷ S can be all V
$high \leftarrow m$ ▷ S can be any single $v \in V$
while $low < high$ **do** ▷ Binary search
    $mid \leftarrow \lfloor (high + low)/2 \rfloor$
    Let ADJ_{mid} denote the adjacency lists for G_{mid}. ▷ These need not be constructed because, given $w_{e_{mid}}$ and the sorted adjacency lists, we can simulate having them.
    $S \leftarrow \emptyset$
    $T \leftarrow V$
    while $\exists x \in T$ **do**
        $S \leftarrow S \cup \{x\}$
        for all $v \in ADJ_{mid}(x)$ **do**
            $T \leftarrow T - ADJ_{mid}(v) - \{v\}$
        end for
    end while
    if $|S| \leq k$ **then**
        $high \leftarrow mid$
        $S' \leftarrow S$
    else
        $low \leftarrow mid + 1$
    end if
end while
Output S'

Thus, the HS Algorithm can be viewed as the Sh Algorithm with a binary search on the make-span, ensuring a 2-approximation guarantee.

Theorem 4.6. *The HS Algorithm is a 2-approximation algorithm for the metric k-center problem [15].*

Proof. The proof follows from the binary search on the make-span r and the proof of the approximation ratio of the Sh Algorithm. □

Theorem 4.7. *The HS Algorithm's running time is $O(n^2 \log n)$, where n is the number of points in the metric space [15].*

Proof. In this case, $k = n$ is the worst-case scenario, where the algorithm outputs all points. The binary search requires $O(\log n)$ iterations, and each iteration involves $O(n^2)$ operations to update the sets S and T. Thus, the total running time is $O(n^2 \log n)$. □

4.4 The CDS Algorithm

Despite achieving a 2-approximation bound, the previous approximation algorithms often underperform on benchmarks compared to heuristic methods. This is due to their conservative decision-making to maintain the approximation guarantee [11]. The CDS (Critical Dominating Set) algorithm [11], a 3-approximation algorithm introduced in 2017, achieves better performance than all existing approximation algorithms despite its higher theoretical bound [11].

Definition 4.3. (Pruned Input Graph [15]) Given an input graph $G = (V, E)$, the *pruned input graph* G_r is a copy of G with all edges of weight greater than r removed. Notice that the minimum make-span of the k-center problem is the minimum r such that the size of the minimum dominating set in G_r is at most k. When this state is reached, we call G_r the *bottleneck graph*.

Definition 4.4. (Critical Vertex) Given a center set C_{i-1}, the *critical vertex* f_i is the vertex that maximizes the distance to C_{i-1} in the bottleneck graph G_r.

Definition 4.5. (Critical Neighbor) A *critical neighbor* is a vertex located within the one-hop neighborhood of the critical vertex in G_r.

Definition 4.6. (Score) The *Score* of a vertex is the number of vertices, non-dominated by the current partial solution set C_{i-1}, that would be dominated by adding the vertex to C_{i-1}.

The aim of the CDS procedure is to generate a dominating set of size at most k in the bottleneck graph G_r, given any make-span r.

For each iteration of the algorithm, it

1. finds the critical vertex f_i,
2. identifies the critical neighbors of f_i,
3. and selects the center with the maximum *Score* among the critical neighbors.

Garcia-Diaz et al. [11] designed the following algorithm.

Algorithm 4 The CDS Algorithm [11]

Input: An undirected graph $G = (V, E)$, an integer k, a covering radius r
Output: A set of vertices $C \subseteq V$, $|C| = k$
// By $N(v)$, we mean the neighbors of v in G_r
$C \leftarrow \emptyset$
$G_r \leftarrow \text{PrunedInputGraph}(G, r)$
$D \leftarrow \emptyset$ ▷ Set of dominated vertices
for each $v \in V$ **do**
    $\text{Score}[v] \leftarrow \text{GetDegree}(v, G_r)$
end for
for $i = 1$ to k **do**
    $f_i \leftarrow$ A vertex $v \in V$ that maximizes $\text{distance}(v, C)$
    $v_{f_i} \leftarrow$ A vertex $v \in N(f_i) \cup f_i$ of maximum Score
    $S \leftarrow (N(v_{f_i}) \cup v_{f_i}) \setminus D$
    for each $v \in S$ **do**

```
        for each u ∈ N(v) do
            Score[u] ← Score[u] − 1
        end for
    end for
    D ← D ∪ S
    C ← C ∪ {v_{f_i}}
end for
return C
```

To solve the k-center problem, we identify the r corresponding to the bottleneck graph G_r. Given that the make-span must be one of the edge weights, we search through the edge weights to find the smallest r such that the minimum dominating set in the bottleneck graph has a size of at most k.

Theorem 4.8. *The Critical Dominating Set (CDS) Algorithm is a 3-approximation algorithm for the metric k-center problem. Specifically, if the proposed make-span r is smaller than the optimal make-span r^*, the algorithm will output a set of centers C with a make-span of at most $3r^*$ [11].*

Proof. First, if during any of the k iterations of the Critical Dominating Set procedure, the farthest vertex is at a distance less than or equal to $3r^*$, then all vertices are within a distance of $3r^*$ from the current solution. Consequently, the algorithm returns a solution with a covering radius $r(C) \leq 3r^*$.

For the remainder of the proof, assume that during all iterations, the farthest vertex is at a distance greater than $3r^*$.

In the i-th iteration, let f_i be the critical vertex and c_i be the selected center. Since c_i is a critical neighbor, $d(c_i, f_i) \leq r \leq r^*$. Let $c^*_{f_i}$ be the nearest center to f_i in the optimal solution. Thus, $d(c^*_{f_i}, f_i) \leq r^*$. For any vertex v in the r^* cluster of $c^*_{f_i}$, we have:

$$d(c_i, v) \leq d(c_i, f_i) + d(f_i, c^*_{f_i}) + d(c^*_{f_i}, v) \leq r^* + r^* + r^* = 3r^*.$$

Finally, we prove by contradiction that all centers $c^*_{f_i}$ covered during the k iterations are distinct. Suppose there exist two centers $c^*_{f_i}$ and $c^*_{f_j}$ with $j < i$ that are the same. Then,

$$d(c_j, f_i) \leq d(c_j, f_j) + d(f_j, c^*_{f_j}) + d(c^*_{f_j}, f_i) \leq r^* + r^* + r^* = 3r^*.$$

However, $d(c_j, f_i) \geq d(C_{i-1}, f_i) > 3r^*$ by the initial assumption, leading to a contradiction. Thus, all centers $c^*_{f_i}$ are distinct, corresponding to all centers in the optimal solution. □

Theorem 4.9. *Similar to the HS Algorithm, we get an algorithm with $O(n^4)$ running time (CDS) if we try all edges as the make-span.*

Proof. The proof is straightforward. □

4.5 Performance in Practice

Despite their theoretical guarantees, these 2-approximation algorithms may not always provide close-to-optimal solutions in practice [11]. Empirical studies suggest that heuristic algorithms can outperform approximation algorithms on real-world datasets.

5 Heuristic Algorithms

While exact algorithms for the metric k-center problem exist, they are often computationally infeasible. Heuristic algorithms, though lacking strict theoretical guarantees, provide practical solutions with reasonable computational effort.

In this section, we review four existing heuristic algorithms: the Gr Algorithm, the CDSh Algorithm, the CDSh+ Algorithm, and the Scr Algorithm. Additionally, we introduce three novel heuristic algorithms: the LS Scr Algorithm, the SA Swp Algorithm, and the SA Swp Scr Algorithm. Our contributions, particularly the LS Scr and SA Swp Scr algorithms, demonstrate superior empirical performance, with the SA Swp Scr Algorithm achieving state-of-the-art results and the most efficient running time among all surveyed algorithms. The experimental results are detailed in the following section.

5.1 The Gr Algorithm

The Gr (Greedy Pure) algorithm adheres to the fundamental principle of greedy algorithms: making locally optimal choices at each step [29].

Algorithm 5 The Gr Algorithm

1: **Input:** Set of points $\mathcal{X}$, integer k
2: **Output:** Set of k centers C
3: $C \leftarrow \emptyset$
4: **for** $i = 1$ to k **do**
5: Select $c_i \in \mathcal{X}$ that minimizes the current make-span
6: $C \leftarrow C \cup \{c_i\}$
7: **end for**
8: **return** C

Theorem 5.1. *The running time of the Gr Algorithm is $O(kn^2)$, where n is the number of points in the metric space.*

Proof. The algorithm iteratively selects k centers, each requiring $O(n^2)$ time to search through all vertices ($O(n)$) and calculate the make-span ($O(n)$ for each vertex). Thus, the total running time is $O(kn^2)$. □

Due to its simplistic approach, the empirical performance of the Gr Algorithm is often suboptimal on most benchmark datasets [29].

5.2 The CDSh Algorithm

The $O(n^4)$ running time of the CDS Algorithm is impractical for large datasets. To address this limitation, the CDSh heuristic was proposed to compress the running time of the CDS Algorithm [11]. We use binary search to restrict the make-span to a smaller range.

Theorem 5.2. *We use the similar binary search technique as the HS Algorithm to obtain a heuristic with running time to $O(n^2 \log n)$ (CDSh). Note that this heuristic does not have any approximation guarantee because the CDS algorithm does not support failure detection.*

Proof. The proof follows the same logic as the HS Algorithm, where we perform a binary search on the make-span r to find the bottleneck graph G_r. □

5.3 The CDSh+ Algorithm

A variant heuristic of CDS named CDSh+ is created by running the CDSh algorithm n times with different random initial centers and selecting the best solution. This operation helps restrict the upper bound of make-span. This heuristic demonstrates strong empirical performance [11].

Theorem 5.3. *The running time of the CDSh+ Algorithm is* $O(n^3 \log n)$.

Proof. This is straightforward from the CDSh Algorithm. □

5.4 The Scr Algorithm

In 2005, Jurij Mihelič and Borut Robič proposed the Scr heuristic algorithm for the Minimum Dominating Set problem, which can be adapted to the metric k-center problem [25]. This heuristic constructs a dominating set according to the "lazy" principle, i.e., the dominating set is enlarged as late as possible.

Initially, the dominating set is empty. For each vertex $v \in V$, we keep the count $CovCnt[v]$, which is the number of times the vertex is covered by the remaining vertices (including itself). The $Score$ ranks the vertices as potential centers.

Mihelič and Robič [25] proposed the following efficient heuristic algorithm for the Minimum Dominating Set problem.

Algorithm 6 The dominating set heuristic algorithm. [25]

Input: Graph $G = (V, E)$

Output: Dominating set D
for all $v \in V$ **do**
    $CovCnt[v] \leftarrow \deg(v) + 1$
end for
$Score \leftarrow CovCnt$
$D \leftarrow \emptyset$
repeat
    $x \leftarrow$ select a node with minimal $Score$
    if $\exists y \in V : (x, y) \in E \wedge CovCnt[y] = 1$ **then**
        $D \leftarrow D \cup \{x\}$
        for all $y : (x, y) \in E$ **do**
            $CovCnt[y] \leftarrow 0$
        end for
    else
        for all $y : (x, y) \in E$ **do**
            if $CovCnt[y] > 0$ **then**
                $CovCnt[y] \leftarrow CovCnt[y] - 1$
                $Score[y] \leftarrow Score[y] + 1$
            end if
        end for
    end if
    $Score[x] \leftarrow \infty$

25: **until** $|V|$ times
26: **return** D

Theorem 5.4. *(Correctness of the dominating set heuristic [25]) The dominating set heuristic algorithm outputs a dominating set of the graph.*

Proof. Consider separately the isolated vertices and the vertices that are not isolated. For the isolated vertices, their *Score* is 1, hence they are selected into the dominating set at the beginning. Now consider a non-isolated vertex v. If at least one of v's neighbors is in the dominating set, then v is covered. If none of v's neighbors are in the dominating set after they are all checked, then $Score[v] = \deg(v) + 1 - \deg(v) = 1$, and hence v is selected into the dominating set. The algorithm terminates after $|V|$ iterations, therefore all vertices are checked. □

We can apply this to the metric k-center problem similar to the CDS algorithm by binary search on the pruned input graph. This is called the Scr Algorithm [25].

Theorem 5.5. *The running time of the Scr Algorithm is $O(n^2 \log n)$. [25]*

Proof. The dominating set heuristic algorithm has a running time of $O(n^2)$, and we perform binary search on the make-span r to find the bottleneck graph G_r. Thus, the total running time is $O(n^2 \log n)$. □

5.5 The LS Algorithm and the LS Scr Algorithm

We adopt a local search algorithm from k-means clustering [21] to solve the k-center problem, and improve it to achieve SOTA empirical performance.

The algorithm starts with an initial set of centers. For each iteration, we first split the points into clusters based on the nearest center. Then for each cluster, we update the center to the point that minimizes the make-span of this specific cluster. We repeat this process until the make-span converges. We run the whole algorithms multiple times with different initial centers and select the best solution to make it more consistent.

Algorithm 7 The LS Algorithm [21]

1: **Input:** Set of points $\mathcal{X}$, integer k.
2: **Output:** Set of k centers C.
3: Randomly select k points as the initial centers.
4: **while** not converged **do**
5: Assign each point to the nearest center.
6: **for** each cluster **do**
7: Update the center to the point that minimizes the make-span.
8: **end for**
9: **end while**

Theorem 5.6. *The LS algorithm terminates in a finite number of steps, but does not guarantee to terminate in polynomial time.*

Proof. The make-span is non-negative and decreases in each iteration. The total number of possible make-spans is finite, namely all pairwise distances between points and the sums of these distances.

Since the make-span is bounded below by the optimal solution, the algorithm must terminate in a finite number of steps.

Since there are exponentially many possible make-spans, the algorithm does not guarantee to terminate in polynomial time. However, in practice, the algorithm often converges fairly quickly. (See the experimental evaluation section for more details.) □

Because the initial centers of the LS algorithm can be chosen arbitrarily, we may use the Scr algorithm's output as the initial centers to improve the performance. We refer to this algorithm as the LS Scr algorithm. This algorithm slightly improves the empirical performance of the Scr algorithm, which is the best-performing heuristic algorithm in practice, hence achieving the state-of-the-art heuristic empirical performance.

Algorithm 8 The LS Scr Algorithm

```
Input: Set of points 𝒳, integer k.
Output: Set of k centers C.
Run the Scr algorithm to obtain the initial centers.
while not converged do
    Assign each point to the nearest center.
    for each cluster do
        Update the center to the point that minimizes the make-span.
    end for
end while
```

5.6 The SA Swp Algorithm and the SA Swp Scr Algorithm

We propose a heuristic algorithm based on simulated annealing (SA) and enhance it to achieve a significant improvement over the state-of-the-art Scr algorithm.

The SA Swp Algorithm begins with an initial set of centers, which can be chosen randomly or derived from another algorithm. During each update step, a center is randomly selected and replaced with a random point, referred to as a swap operation. If the new set of centers results in a smaller make-span, the change is accepted. Otherwise, the change is accepted with a probability of $e^{-\delta/T}$, where δ is the change in make-span, and T is the temperature. The initial temperature is set to the make-span of the initial centers, and the terminating temperature is a constant multiple of the initial temperature. The temperature is gradually decreased to control the probability of accepting worse solutions, and the algorithm terminates when the temperature reaches a threshold. To ensure consistency, the algorithm is run multiple times, and the best solution is selected.

Another potential update operation involves replacing a center c with the farthest point p from the current center set excluding c. This variant, inspired by the Gon Algorithm, is referred to as the SA Swp Gon Algorithm. However, due to its poor performance, it is not elaborated upon here. The source code is available in the repository mentioned in the abstract and the next section.

Algorithm 9 The SA Swp Algorithm

```
Input: Set of points 𝒳, integer k.
Output: Set of k centers C.
Randomly select k points as the initial centers.
T_0 ← make-span of the initial centers.
```

$T \leftarrow \beta T_0$ ($\beta = 1$)
while $T > \gamma T_0$ **do**
    Randomly select a center c.
    Randomly select a point p other than the current center set.
    Replace c with p.
    Calculate the new make-span.
    $\delta \leftarrow$ new make-span $-$ old make-span.
    if $\delta < 0$ or rand$(0, 1) < e^{-\delta/T}$ **then**
        Accept the change.
    end if
    $T \leftarrow T \times \alpha$
end while

Theorem 5.7 (Running Time of the SA Swp Algorithm)**.** *The running time of the SA Swp Algorithm is $O(C_0kn)$, where C_0 is the number of iterations, a large but constant number.*

Proof. The number of iterations is constant. Each iteration requires $O(n)$ time for the update process and $O(kn)$ time for calculating the make-span. Thus, the total running time is $O(C_0kn)$. Although the number of iterations is large, it remains constant. □

Similar to the LS Scr algorithm, the Scr algorithm's output can be used as the initial centers to enhance performance. However, careful consideration of both the initial temperature and the decay rate is necessary. Since the Scr algorithm already provides a good heuristic, starting with a lower temperature and decaying it more slowly when the initial centers are close to the optimal solution is advisable to avoid ending up with a worse solution.

In pursuit of this goal, we introduce a parameter to approximate the scale of the optimal make-span. Empirically, we observe that the SA Swp algorithm can refine the Scr algorithm's solutions when the minimal make-span is relatively large. Intuitively, larger make-spans amplify the absolute error under the same approximation ratio, thereby increasing the likelihood that the SA Swp algorithm explores and identifies a better solution.

Definition 5.1 (Sparseness)**.** Consider a graph $G(V, E)$ with $|V| = n$, $|E| = m$, and an integer k. We define its *sparseness* parameter as

$$\text{sparseness} \;=\; \frac{1}{k}\left(\frac{\log n}{\log \frac{2m}{n}}\right)^2.$$

Here, dividing by k acknowledges that more centers generally reduce the minimal make-span, and the factor $\frac{\log n}{\log \frac{2m}{n}}$ reflects the order of the expected edge distance between a given node and a random node. We square this term to capture its greater-than-linear influence on the minimal make-span.

Theorem 5.8. *Asymptotically, the expected edge distance from any fixed vertex to a random vertex is $\frac{\log n}{\log \frac{2m}{n}}$.*

Proof. Let d denote the expected degree of a vertex. Summing up the degrees of all vertices, we have $dn = 2m$, implying $d = 2m/n$. Generally speaking, d is larger than $O(1)$. The number of

vertices at a given edge distance l (i.e., the number of edges needed to go from a given point to another) is $O(d^l)$. Therefore, the total number of vertices is

$$\sum_{l=0}^{l_{\max}} O(d^l) = O\left(\frac{d^{l_{\max}} - 1}{d-1}\right) = O(d^{l_{\max}}) = n.$$

Subsequently, the expected number of edges needed to go from a given point to a random point is

$$\bar{l} = \frac{1}{n}\sum_{l=0}^{l_{\max}} l \cdot O(d^l) = \frac{1}{n} d \frac{\partial}{\partial d} \sum_{l=0}^{l_{\max}} O(d^l) = O(l_{\max}) = O\left(\frac{\log n}{\log d}\right) = O\left(\frac{\log n}{\log \frac{2m}{n}}\right).$$

□

The make-span, defined as the maximum distance between any point and its nearest center, is influenced by this expected edge distance. As the expected edge distance increases, the make-span also tends to increase. However, this influence is greater than linear due to the combinatorial nature of the problem and the distribution of points in the graph. Squaring the term $\frac{\log n}{\log \frac{2m}{n}}$ reflects this greater-than-linear relationship, amplifying its effect on the sparseness parameter.

We use the sparseness parameter to control the initial temperature and the decay rate of the SA Swp algorithm. According to the form of the sparseness parameter, it is easily shown that $\frac{1}{n} \leq \text{sparseness} \leq (\log n)^2$. Let

$$\alpha = \exp(-a \cdot \text{sparseness}),$$
$$\beta = \exp(-b/\text{sparseness}),$$

where a and b are constants to be determined by empirical studies. Under this setting, the number of iterations of the SA Swp Scr algorithm has a constant upper bound, which is crucial for practical applications.

Theorem 5.9 (Constant Upper Bound of the SA Swp Scr Algorithm)**.** *The number of iterations of the SA Swp algorithm after Scr initialization has a constant upper bound*

$$\#\textit{iterations} \leq \frac{(\log(1/\gamma))^2}{4ab}.$$

Proof. Let s denote the sparseness parameter. The initial temperature is set to βT_0, and the terminating temperature is set to γT_0. The temperature decreases by a factor of α in each iteration. The number of iterations is

$$\begin{aligned}
\#\text{iterations} &= \log_\alpha \frac{\gamma}{\beta} \\
&= \frac{\log\gamma - \log\beta}{\log\alpha} \\
&= \frac{\log(1/\gamma) - b/s}{a \cdot s} \\
&= \frac{s\log(1/\gamma) - b}{a \cdot s^2}.
\end{aligned}$$

This function of s, sketched in figure 3, has an upper bound. We can find the maximum of this function by taking the derivative and setting it to zero.

$$\log(1/\gamma) \cdot as^2 - 2as(s\log(1/\gamma) - b) = 0 \implies s = \frac{2b}{\log(1/\gamma)}.$$

By checking the second derivative, we can confirm that this is a maximum. Substituting this value back into the function, we obtain the upper bound

$$\#\text{iterations} \leq \frac{(\log(1/\gamma))^2}{4ab}.$$

□

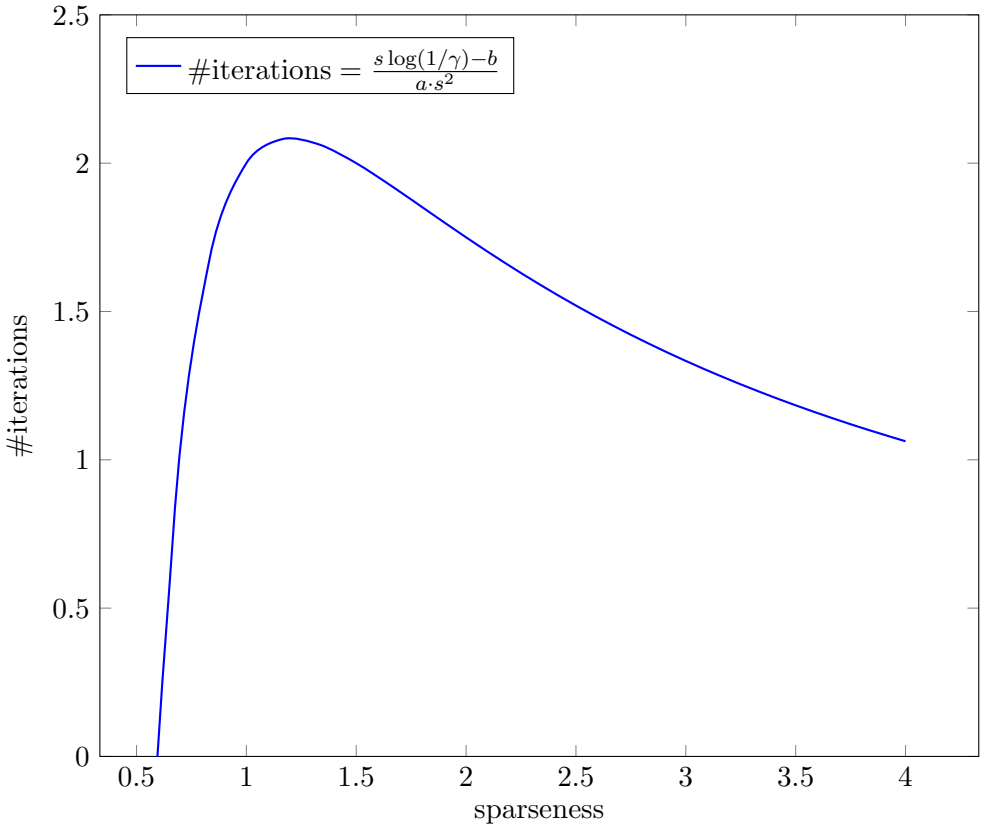


Figure 3: Sketch of #iterations as a function of sparseness.

Exploiting the above fact, we derive a clean upper bound for the running time of the SA Swp Scr algorithm.

Theorem 5.10 (Running Time of the SA Swp Scr Algorithm)**.** *The running time of the SA Swp Scr Algorithm is* $O(n^2 \log n + C_0 kn)$*, where* C_0 *is a large constant.*

Proof. The Scr algorithm runs in $O(n^2 \log n)$ time. The number of iterations has a constant upper bound by Theorem 5.9. Each iteration requires $O(n)$ time for the update process and $O(kn)$ time

for calculating the make-span. Thus, the total running time is $O(n^2 \log n + C_0 kn)$. The constant C_0 is emphasized to indicate the large but constant number of iterations. □

While the asymptotic running time is neat, the constants a, b, and γ are crucial for the algorithm's performance. Below is the pseudocode for the SA Swp Scr algorithm.

Algorithm 10 The SA Swp Scr Algorithm

Input: Set of points $\mathcal{X}$, integer k.
Output: Set of k centers C.
Run the Scr algorithm to obtain the initial centers.
Calculate the sparseness parameter.
Set the initial temperature T_0 to the make-span of the initial centers.
sparseness $\leftarrow \frac{1}{k}\left(\frac{\log n}{\log \frac{2m}{n}}\right)^2$.
$\alpha \leftarrow \exp(-a \cdot \text{sparseness})$.
$\beta \leftarrow \exp(-b/\text{sparseness})$.
Set the terminating temperature T to βT_0.
while $T > \gamma T_0$ **do**
    Randomly select a center c.
    Randomly select a point p other than the current center set.
    Replace c with p.
    Calculate the new make-span.
    $\delta \leftarrow$ new make-span − old make-span.
    if $\delta < 0$ or $\text{rand}(0, 1) < e^{-\delta/T}$ **then**
        Accept the change.
    end if
    $T \leftarrow T \times \alpha$
end while

5.7 Other Heuristics

There are various complex metaheuristics for the k-center problem, including Tabu Search (TS) and Variable Neighborhood Search (VNS) [24]. TS uses memory structures to avoid revisiting previous solutions, while VNS systematically changes the neighborhood structure to escape local optima. Due to the structral and computational complexity of these algorithms, we do not discuss them in detail here.

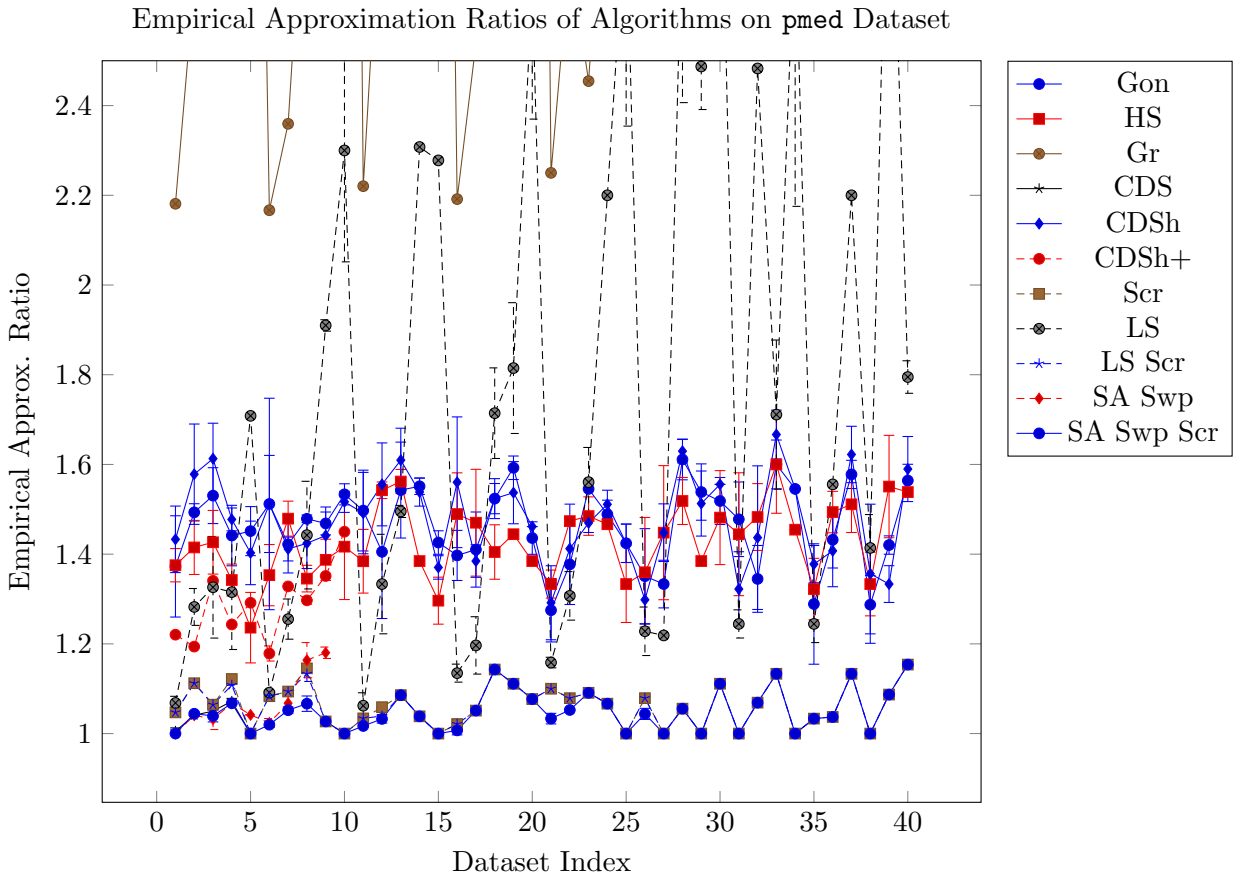
6 Experimental Evaluation

To evaluate the performance of the approximation and heuristic algorithms, we conduct experiments on a standard benchmark dataset, namely the `pmed` dataset from the OR-Library [1]. We compare the empirical approximation ratios, and running times of the algorithms, including the Gon, HS, Gr, CDS, CDSh, CDSh+, Scr, LS, LS Scr, SA Swp, and SA Swp Scr algorithm. The codes, datasets, and explicit results are available on `https://github.com/patrickwzh/k-Center_Evaluation`.

Since the `pmed` datasets are graphs, we preprocess all pairs shortest paths to convert them into metric spaces, and store it into `.preprocessed` files.

6.1 Empirical Approximation Ratios

We compared the empirical approximation ratios of the algorithms on the `pmed` dataset. Due to the running time constraints, larger datasets are not tested on slow algorithms like CDS. The optimal solutions are from Garcia-Diaz et al. [10].



The SA Swp Scr algorithm achieves the best empirical approximation ratios on most datasets, outperforming all other algorithms except for a marginal difference of only 0.007 on `pmed03`, where the SA Swp algorithm, also proposed by us, is slightly better.

We average the approximation ratios over all datasets. The results are shown in Table 1.

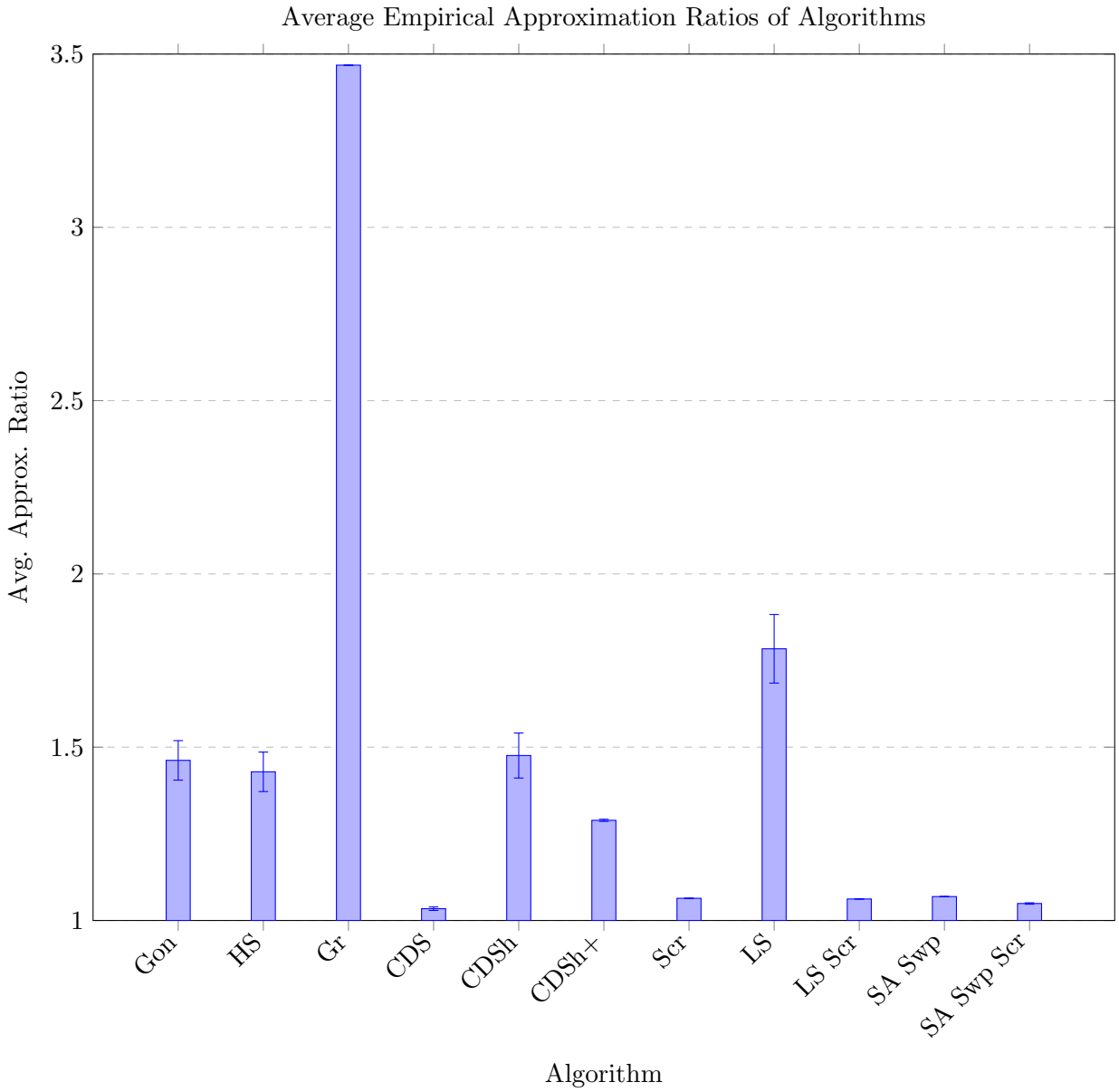
Algorithm	Avg. Approx. Ratio	StdDev.
Gon	1.462	0.057
HS	1.429	0.057
Gr*	3.468	0.000
CDS*	1.034	0.005
CDSh	1.476	0.065
CDSh+	1.289	0.003
Scr	1.064	0.000
LS	1.784	0.099
LS Scr	1.062	0.000
SA Swp*	1.069	0.000
SA Swp Scr	1.049	0.002

Table 1: Average Empirical Approximation Ratios and Standard Deviations of Algorithms

The entries marked with * indicate that the algorithm was only tested on part of the datasets due to its running time.

Notice that all three algorithms proposed in this survey, namely the LS Scr, SA Swp, and SA Swp Scr algorithms, achieve SOTA level empirical performance. The SA Swp Scr algorithm outperforms all algorithms. Note that although the CDS algorithm seems to have a lower average approximation ratio, it was only tested on the first five datasets. The SA Swp Scr algorithm outperforms the CDS algorithm on all the first five datasets, indicating its superior performance. This can be seen directly from figure 6.1.

For better visualization, we plot the average empirical approximation ratios of the algorithms.



The results indicate that the CDS, Scr, LS Scr, SA Swp, and SA Swp Scr algorithms achieve the best empirical approximation ratios, showcasing superior performance. However, the CDS algorithm's running time is prohibitively high, rendering it impractical for real-world applications. The CDSh+ algorithm also demonstrates commendable performance, while the remaining algorithms exhibit suboptimal performance with approximation ratios around 0.5. Overall, our newly proposed algorithms, particularly the SA Swp Scr algorithm, offer the best empirical performance.

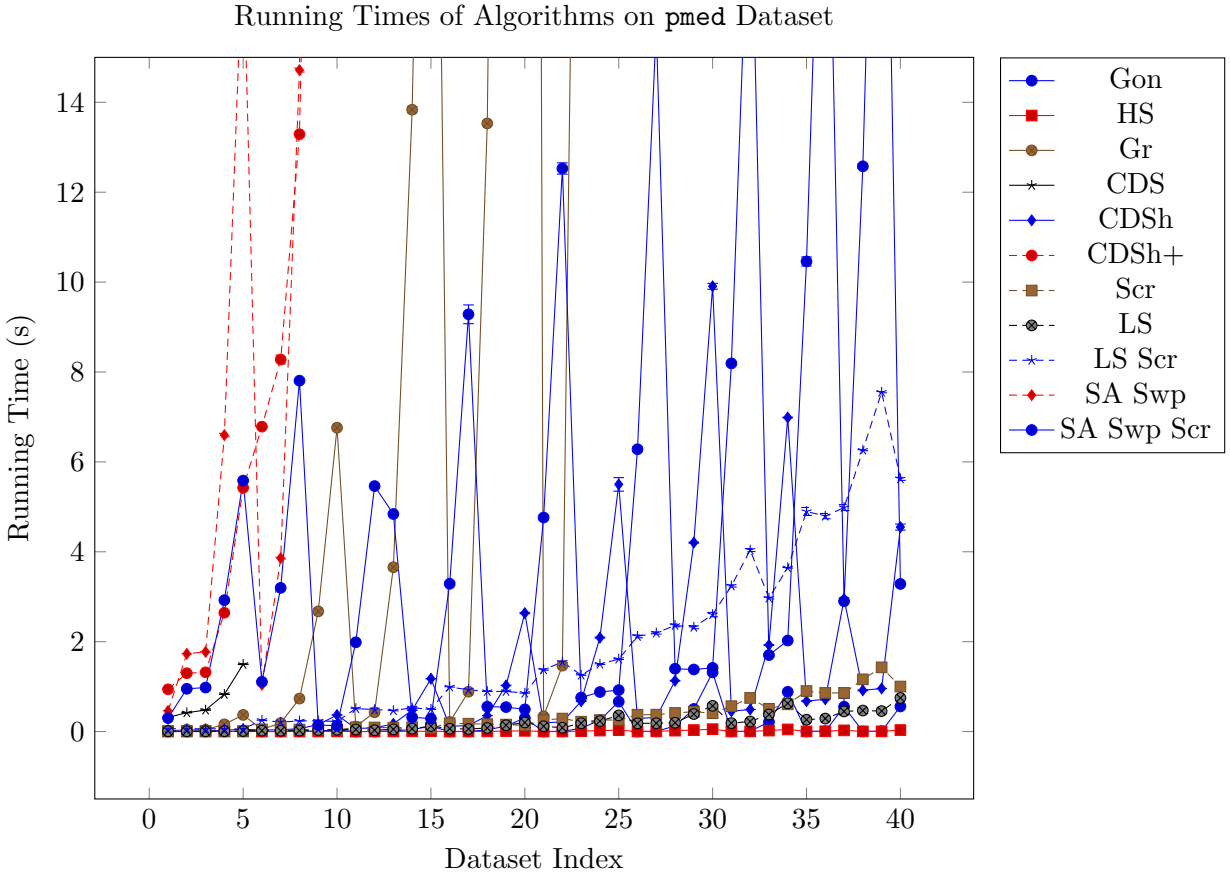
6.2 Running Times

The running times of the algorithms are summarized in Table 2.

Algorithm	Running Time
Gon	$O(kn)$
HS	$O(n^2 \log n)$
Gr	$O(kn^2)$
CDS	$O(n^4)$
CDSh	$O(n^2 \log n)$
CDSh+	$O(n^3 \log n)$
Scr	$O(n^2 \log n)$
LS	Exponential
LS Scr	Exponential
SA Swp	$O(C_0 kn)$
SA Swp Scr	$O(n^2 \log n + C_0 kn)$

Table 2: Theoretical Running Times of Algorithms

The running times of the algorithms were evaluated on a MacBook Pro equipped with an M4 Pro chip and 48GB of RAM. The algorithms were implemented in Python.



The results indicate that the HS, Gon, Scr, and LS algorithms exhibit the best performance in terms of running time. The CDSh algorithm demonstrates moderate performance, while the CDSh+, CDS and Gr algorithms have the longer running times. Notice that the running time of

the SA Swp and the SA Swp Scr algorithms fluctuates hugely with k. This is because the number of iterations has not yet reached the upper bound, which is a large number due to the specific choice of the constants. Therefore, the slow running time of the SA Swp and the SA Swp Scr algorithms is due to the large constant factor in the upper bound of the number of iterations, and would outperform the other algorithms if the order of input size is large enough.

6.3 Discussion

The experimental results indicate that the SA Swp Scr algorithm achieves the best empirical performance among all tested algorithms, with an average approximation ratio of 1.049 and an asymptotic running time of $O(n^2 \log n + C_0 kn)$. The LS Scr algorithm also shows strong performance, with an average approximation ratio of 1.062 and exponential running time. The SA Swp algorithm achieves an average approximation ratio of 1.069 and a running time of $O(C_0 kn)$, while the Scr algorithm achieves an average approximation ratio of 1.064 and a running time of $O(n^2 \log n)$.

The CDS algorithm achieves an average approximation ratio of 1.034, but its running time of $O(n^4)$ is impractical for large datasets. The CDSh+ algorithm strikes a good balance between approximation ratio and running time, with an average approximation ratio of 1.289 and a running time of $O(n^3 \log n)$.

7 Conclusion

This survey examined the metric k-center problem, its history, formal definitions, NP-hardness, computational complexity, and various algorithms designed to address it. We discussed approximation algorithms such as the Gon, Sh, HS, and CDS algorithms, highlighting their theoretical guarantees and practical limitations. Additionally, we explored heuristic algorithms including the Gr, CDSh, CDSh+, Scr, and LS algorithms, evaluating their empirical performance and applicability to real-world datasets.

We introduced three novel heuristic algorithms: LS Scr, SA Swp, and SA Swp Scr. Among these, the SA Swp Scr algorithm demonstrates superior empirical performance with an average approximation ratio of 1.049 and a running time of $O(n^2 \log n + C_0 kn)$. The LS Scr and SA Swp algorithms also show strong empirical performance, further validating the effectiveness of our proposed heuristics.

Our review underscores the trade-offs between the theoretical efficiency of approximation algorithms and the practical effectiveness of heuristics. While approximation algorithms provide provable bounds, heuristics often yield superior performance in practice, particularly on large and complex datasets.

In conclusion, heuristic algorithms, especially the Scr, LS Scr, SA Swp, and SA Swp Scr algorithms, outperform traditional approximation algorithms in practical scenarios. The SA Swp Scr algorithm, in particular, achieves an optimal balance between empirical approximation ratio and running time, making it a highly effective solution for the metric k-center problem. These findings highlight the importance of considering both theoretical and empirical performance when selecting algorithms for real-world applications.

References

[1] J. E. Beasley. Or-library: Distributing test problems by electronic mail. *The Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

[2] TH Hubert Chan, Arnaud Guerqin, and Mauro Sozio. Fully dynamic k-center clustering. In *Proceedings of the 2018 World Wide Web Conference*, pages 579–587, 2018.

[3] Moses Charikar, Samir Khuller, David M. Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, page 642–651, USA, 2001. Society for Industrial and Applied Mathematics.

[4] Sanjoy Dasgupta and Philip M. Long. Performance guarantees for hierarchical clustering. *Journal of Computer and System Sciences*, 70(4):555–569, 2005. Special Issue on COLT 2002.

[5] M.E Dyer and A.M Frieze. A simple heuristic for the p-centre problem. *Operations Research Letters*, 3(6):285–288, 1985.

[6] Y. Eldar, M. Lindenbaum, M. Porat, and Y.Y. Zeevi. The farthest point strategy for progressive image sampling. *IEEE Transactions on Image Processing*, 6(9):1305–1315, 1997.

[7] David Eppstein, Sariel Har-Peled, and Anastasios Sidiropoulos. Approximate greedy clustering and distance selection for graph metrics. *Journal of Computational Geometry*, page Vol. 11 No. 1 (2020), 2020.

[8] Absalom E Ezugwu, Abiodun M Ikotun, Olaide O Oyelade, Laith Abualigah, Jeffery O Agushaka, Christopher I Eke, and Andronicus A Akinyelu. A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects. *Engineering Applications of Artificial Intelligence*, 110:104743, 2022.

[9] Tomás Feder and Daniel Greene. Optimal algorithms for approximate clustering. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, page 434–444, New York, NY, USA, 1988. Association for Computing Machinery.

[10] Jesus Garcia-Diaz, Rolando Menchaca-Mendez, Ricardo Menchaca-Mendez, Saúl Pomares Hernández, Julio César Pérez-Sansalvador, and Noureddine Lakouari. Approximation algorithms for the vertex k-center problem: Survey and experimental evaluation. *IEEE Access*, 7:109228–109245, 2019.

[11] Jesus Garcia-Diaz, Jairo Sanchez-Hernandez, Ricardo Menchaca-Mendez, and Rolando Menchaca-Mendez. When a worse approximation factor gives better performance: a 3-approximation algorithm for the vertex k-center problem. *Journal of Heuristics*, 23:349–366, 2017.

[12] Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical computer science*, 38:293–306, 1985.

[13] S Louis Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations research*, 12(3):450–459, 1964.

[14] Sariel Har-Peled. *Geometric approximation algorithms*. Number 173. American Mathematical Soc., 2011.

[15] Dorit Hochbaum and David Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research - MOR*, 10:180–184, 05 1985.

[16] Wen-Lian Hsu and George L. Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1(3):209–215, 1979.

[17] Daxin Jiang, Chun Tang, and Aidong Zhang. Cluster analysis for gene expression data: a survey. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1370–1386, 2004.

[18] Oded Kariv and S Louis Hakimi. An algorithmic approach to network location problems. i: The p-centers. *SIAM journal on applied mathematics*, 37(3):513–538, 1979.

[19] Samir Khuller, Robert Pless, and Yoram J Sussmann. Fault tolerant k-center problems. *Theoretical Computer Science*, 242(1-2):237–245, 2000.

[20] Samir Khuller and Yoram J Sussmann. The capacitated k-center problem. *SIAM Journal on Discrete Mathematics*, 13(3):403–418, 2000.

[21] David J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, USA, 2002.

[22] Richard Matthew McCutchen and Samir Khuller. Streaming algorithms for k-center clustering with outliers and with anonymity. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques: 11th International Workshop, APPROX 2008, and 12th International Workshop, RANDOM 2008, Boston, MA, USA, August 25-27, 2008. Proceedings*, pages 165–178. Springer, 2008.

[23] E. Mazer, J. M. Ahuactzin, and P. Bessiere. The ariadne's clew algorithm. *Journal of Artificial Intelligence Research*, 9:295–316, November 1998.

[24] Jurij Mihelič and Borut Robič. Approximation algorithms for the k-center problem: an experimental evaluation. In *Operations Research Proceedings 2002: Selected Papers of the International Conference on Operations Research (SOR 2002), Klagenfurt, September 2–5, 2002*, pages 371–376. Springer, 2003.

[25] Jurij Mihelič and Borut Robic. Solving the k-center problem efficiently with a dominating set algorithm. *CIT*, 13:225–234, 09 2005.

[26] Carsten Moenning and Neil A. Dodgson. A new point cloud simplification algorithm. 2003.

[27] Anahita Namvar, Mehdi Ghazanfari, and Mohsen Naderpour. A customer segmentation framework for targeted marketing in telecommunication. In *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, pages 1–6. IEEE, 2017.

[28] Rina Panigrahy and Sundar Vishwanathan. Ano (log* n) approximation algorithm for the asymmetricp-center problem. *Journal of Algorithms*, 27(2):259–268, 1998.

[29] Rattan Rana and Deepak Garg. Heuristic approaches for k-center problem. In *2009 IEEE International Advance Computing Conference*, pages 332–335, 2009.

[30] Jiayang Ren, Ningning You, Kaixun Hua, Chaojie Ji, and Yankai Cao. A global optimization algorithm for k-center clustering of one billion samples, 2022.

[31] Reza Shahidi, Cecilia R. Moloney, and Giovanni Ramponi. Design of farthest-point masks for image halftoning. *EURASIP Journal on Advances in Signal Processing*, 2004:1–13, 2004.

[32] David Shmoys. Computing near-optimal solutions to combinatorial optimization problems. 20, 02 1996.

[33] George Stockman, Steven Kopstein, and Sanford Benett. Matching images to models for registration and object detection via clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-4(3):229–241, 1982.

[34] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, USA, 1st edition, 2011.

[35] Zhigang Xiang. Color image quantization by minimizing the maximum intercluster distance. *ACM Trans. Graph.*, 16(3):260–276, July 1997.