Iterative Graph-Based Radius-Constrained Clustering

Quentin Haenn[0009-0009-1663-0107], Brice Chardin[0000-0002-9298-9447], Mickael Baron[0000-0002-3356-0835], and Allel Hadjali[0000-0002-4452-1647]

ISAE-ENSMA, LIAS, France
{quentin.haenn, brice.chardin, mickael.baron, allel.hadjali}@ensma.fr

Abstract. Clustering aims at grouping data into homogeneous clusters. However, setting parameters such as a cluster count or a dissimilarity bound can be challenging. This paper introduces Curgraph, a novel iterative approach for radius-constrained clustering. Curgraph identifies optimal partitions with respect to maximum cluster radius by computing minimum dominating sets across partial graphs. Experimental results demonstrate Curgraph's effectiveness compared with state-of-the-art algorithms.

Keywords: Constrained Clustering · Radius Based Clustering · Minimum Dominating Set

1 Introduction

Cluster analysis involves several sub-problems, including: (1) partitioning a set of points into homogeneous and well-separated subsets, called clusters, (2) factoring in domain-specific knowledge, and (3) identifying representative points, or *prototypes*. To solve the second issue, user supervision may be required in the form of various constraints, such as a target number of cluster, an expected density, or examples of points that should (or should not) be grouped together [8]. The third issue is usually solved as a post-processing step. Unfortunately, once clusters are defined, it might be impossible to identify a prototype that is similar to every other point of the cluster.

This work builds upon radius-constrained clustering algorithms, for their ability to provide strong similarity bounds on resulting prototypes [1, 2, 4, 10]. However, these algorithms rely on a user-provided radius threshold, which can be challenging to set. To alleviate this requirement, iterative (both hierarchical and non-hierarchical) algorithms can compute multiple candidate partitions for various thresholds. In this paper, we introduce a novel non-hierarchical iterative clustering algorithm, called Curgraph, to compute the sequence of partitions with minimal radius.

Paper organization Section 2 provides formal definitions of the minimum radius clustering problem. Section 3 gives an overview of related work. Section 4 introduces the Curgraph algorithm. Section 5 presents experimental evaluations. A discussion of the results concludes the paper.

2 Problem Statement

In hard partitional clustering [12], a valid result is a partition of the input set of points, or population.

Definition 1 (Partitional clustering). *Let $\mathcal{X}$ be a population. A clustering is a partition $\mathcal{P} = \{C_1, C_2, \ldots, C_k\}$ of $\mathcal{X}$:*

- $\forall C_i \in \mathcal{P}, C_i \neq \emptyset$ (non-emptiness)
- $\forall C_i, C_j \in \mathcal{P}, i \neq j \Rightarrow C_i \cap C_j = \emptyset$ (disjointness)
- $\bigcup_{C_i \in \mathcal{P}} C_i = \mathcal{X}$ (coverage)

As a measure of homogeneity, the radius of a cluster C is the minimum eccentricity within C. By extension, the radius of a partition is the maximum radius of its clusters. Clustering algorithms usually seek to minimize this radius.

Definition 2 (Radius). *Given a dissimilarity function d, the radius of a cluster C and the radius of a partition $\mathcal{P}$ are:*

$$R(C) = \min_{a \in C} \max_{b \in C} d(a, b)$$
$$R(\mathcal{P}) = \max_{C \in \mathcal{P}} R(C)$$

With radius-based clustering, the natural definition of a prototype—the representative element of a cluster—is the point that offers the best worst-case dissimilarity to any other point in the cluster, that is, the point a of C such that $a = \arg\min_{a \in C} \max_{b \in C} d(a, b)$. This definition of a prototype guarantees that, for every point of C, the representation error is bounded by $R(C)$.

In this paper, the problem we aim to solve is to find the partitioning of a population into k clusters, such that the radius of the partition is minimal.

Definition 3 (Minimum radius clustering). *Let $\mathcal{X}$ be a population, k a desired number of clusters. We denote $\mathcal{P}_k$ the set of all partitions $\mathcal{P}$ of $\mathcal{X}$ such that $|\mathcal{P}| = k$. The minimum radius clustering problem consists in finding one minimum radius partitioning $\mathcal{P}_k^*$ of $\mathcal{X}$ into k clusters. Note that $\mathcal{P}_k^*$ is not necessarily unique.*

$$\mathcal{P}_k^* = \arg\min_{\mathcal{P} \in \mathcal{P}_k} R(\mathcal{P})$$

The minimum radius clustering problem may be associated with the following decision problem: given a dataset $\mathcal{X}$, a dissimilarity d, a dissimilarity threshold t and a number of clusters k, is there a partition of $\mathcal{X}$ into k clusters such that $R(P_k^*) \leq t$?

Theorem 1. *The minimum radius clustering decision problem is NP-complete.*

To establish a theoretical framework, we introduce the concept of graph-based clustering. A simple graph $G_t = (\mathcal{X}, E_t)$ is a couple where $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ is a set of vertices and E_t is a set of undirected edges associated with a threshold t: $E_t = \{\{x_i, x_j\} \mid d(x_i, x_j) \leq t\}$. Namely, x_i and x_j are adjacent in G_t if their pairwise dissimilarity is at most t.

A set $D_t \subseteq \mathcal{X}$ is a dominating set if and only if, for each vertex $x_i \in \mathcal{X}$, either $x_i \in D_t$ or there exists an adjacent vertex $x_j \in D_t : \{x_i, x_j\} \in E_t$. A minimum dominating set is a dominating set with the smallest possible cardinality. This minimum cardinality is called the domination number of the graph, $\gamma(G_t)$.

Proof of Theorem 1. We establish an equivalence with the minimum dominating set (MDS) problem, which is NP-complete. The MDS (or domination number) decision problem is formulated as: given a graph G and a number k, is $\gamma(G) \leq k$?
($\Rightarrow$) Consider the graph $G_t = (\mathcal{X}, E_t)$ and compute a minimum dominating set. If $\gamma(G_t) \leq k$ then $R(P_k^*) \leq t$, else $R(P_k^*) > t$.
($\Leftarrow$) Consider a graph $G = (\mathcal{X}, E)$ and associate a dissimilarity d as: $d(x, y) = 1$ if $\{x, y\} \in E$, else $d(x, y) = 2$. Compute a minimum radius clustering with k clusters. If $R(P_k^*) \leq 1$, then $\gamma(G) \leq k$, else $\gamma(G) > k$.
Since the MDS problem, which is NP-complete, is equivalent to the minimum radius clustering problem, the latter is also NP-complete.

As a consequence of Theorem 1, finding the minimum radius partition for k clusters is computationally intensive regardless of the algorithm used. However, computing a graph's minimum dominating set may be more practical than spending hours tuning parameters for traditional algorithms that yield suboptimal results. This is the primary motivation behind the proposed algorithm. Nevertheless, to be able to scale on larger datasets, approximate solutions will also be explored.

3 Related Work

In constraint-based clustering, various wideness measures like diameter, radius, and average distance have been introduced as cluster-level dissimilarity constraints [1, 2, 4, 6, 8, 9]. Past research has mostly focused on diameter constraints [1, 6, 11], and radius constraints have remained underexplored [1, 2, 4, 13]. A key difference between diameter and radius constraints is that identifying a prototype with a diameter-based algorithm has to be performed as a post-processing step. Additionally, diameter constraints can be translated into cannot-link instance-level constraints [6, 7], while radius constraints cannot.

Clustering under radius constraints (CRC) consists in finding a partition whose radius is at most equal to a user-provided threshold [4]. Since a trivial solution would be to assign each point to its own cluster, a secondary goal is usually to minimize the number of clusters. Like minimum radius clustering,

CRC has been linked to the minimum dominating set problem in graphs [1], where a dominating set is a candidate set of prototypes for the resulting partition.

Besides graph-related solutions, the equiwide clustering algorithm [1] is a notable approach to the CRC problem, using an exact linear programming implementation that efficiently handles small datasets. Alternatively, hierarchical agglomerative clustering (HAC) algorithms with minimax linkage [2, 4] comply with radius constraints but are suboptimal for both minimizing the number of clusters and the partition radius. The most efficient minimax HAC variant identified is Protoclust [4, 16]. As with other hierarchical algorithms, getting a single partition requires a cut of the resulting hierarchy, using either a radius threshold or a number of clusters.

The last related problem is the k-center optimization problem. It involves selecting k points to minimize the maximum distance from any point to its nearest center. Having centers drawn from within the input set of points is similar to radius constraints, while being able to place centers anywhere on a vector space is related with diameter constraints. Efficient approximate algorithms exist, even for large datasets [15].

However, these solutions assume a metric space that satisfies the triangle inequality, which makes them unusable with other dissimilarities, e.g., dynamic time warping (DTW) used to classify time series [3].

Recent advancements in exact and approximate algorithms for the MDS problem [5, 14] allow for some efficient solutions in graph related work for the CRC problem [10]. Like other CRC solutions, MDS requires a predefined dissimilarity threshold, relying on data-specific user knowledge – a challenge this paper aims to address. Without a clear dissimilarity threshold, analysts may have to examine clustering results across varying thresholds or cluster counts.

For diameter constraints, the Clustergraph algorithm [11] has been designed to efficiently compute a sequence of optimal partitions, each with the minimum diameter for every possible number of cluster. In this paper, we drew inspiration from Clustergraph, adapting it to handle radius constraints. The main differences are the underlying graph algorithms and how the dataset is modeled as a graph.

4 Curgraph

Curgraph (**C**lustering **u**nder **r**adius constraints using **graph**s) considers a sequence of partial graphs G_t, where the dissimilarity threshold t takes its values in the pairwise dissimilarities between points of $\mathcal{X}$, ranked in decreasing order. A minimum dominating set D_t is computed for each partial graph. Since a dominating set does not properly define a partition, we need to assign each vertex x_i to a cluster based on the nearest dominating vertex that covers it, resulting in $|D_t|$ clusters. As stated in theorem 2, the set of all partitions $\mathcal{P}$ of $\mathcal{X}$ from 1 to $|\mathcal{X}|$ clusters obtained in this way contains all minimum radius partitions of $\mathcal{X}$.

Theorem 2. *The partition derived from a minimum dominating set of the last partial graph $G_t = (\mathcal{X}, E_t)$ dominable by k vertices is a minimum radius partition of $\mathcal{X}$ into k clusters.*

Proof. Let t_i represent the i-th value of t in the sequence of decreasing distinct values of t: $t_i > t_{i+1}$. The associated partial graph is $G_{t_i} = (X, E_{t_i})$, and assume it is the last k-dominable partial graph. The assignment of vertices to the nearest dominating vertex defines a partition, denoted $\mathcal{P}_k^i$. For any x_i and x_j in $\mathcal{X}$ such that $d(x_i, x_j) > t_i$, x_i and x_j are not adjacent in G_{t_i} and neither can dominate the other. Therefore, $R(P_k^i) \leq t_i$. As the next partial graph in sequence, $G_{t_{i+1}}$, is not k-dominable, any partition of $\mathcal{X}$ into k clusters with D its set of prototypes must contain at least one vertex x such that $\forall c \in D$, $d(x, c) > t_{i+1}$. Hence, no partition $\mathcal{P}_k'$ of $\mathcal{X}$ with $R(\mathcal{P}_k') \leq t_{i+1} < t_i$ exists and $\mathcal{P}_k^i$ has minimum radius among $\mathcal{P}_k$ with $R(\mathcal{P}_k^i) = R(\mathcal{P}_k^*) = t_i$.

Following this sequential approach, we need to compute a minimum dominating set for each partial graph. As the underlying problem is NP-hard, this represents a major drawback of the approach, and it will be the bottleneck of the algorithm, especially on larger datasets. However, we can improve this behavior by considering a dominating set D_{t_i} as a potential solution for the next partial graph. Since the partial graphs are built sequentially by gradually removing edges in decreasing dissimilarity order, typically only one edge is removed at a time. If this removal does not affect a vertex's link to its dominant point, the existing dominating set remains valid for $G_{t_{i+1}}$. In practice, as shown in section 5, most iterations of the algorithm do not have to compute a new dominating set. Nevertheless, since even a single MDS call can be prohibitive, we also consider approximate MDS algorithms to identify suboptimal partitions.

4.1 General Algorithm

The algorithm takes a set of points $\mathcal{X}$ and an optional maximum cluster count, $k_{\max}$. It outputs a minimum radius partition for each number of clusters up to $k_{\max}$, or up to $|\mathcal{X}|$ if $k_{\max}$ is not set. The current cluster count is denoted k, and the current dominating set is D. The outline of the algorithm is shown in Algorithm 1.

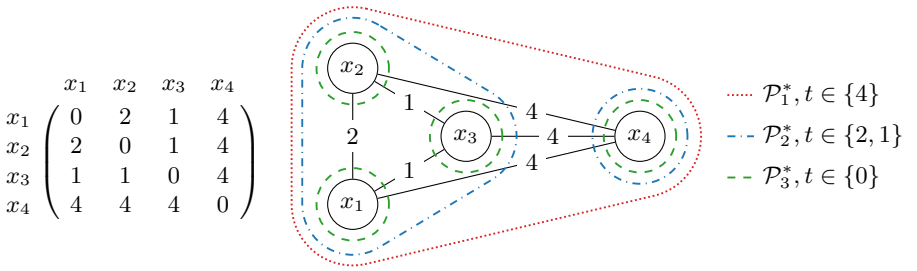
First, the INITIALIZE function is called. This function handles the special case for $k = 1$, and computes the ranked list, noted T, of pairwise dissimilarities between points in $\mathcal{X}$ lower than $R(\mathcal{P}_1^*)$, in decreasing order. The algorithm then iterates over values t in T. For each t, the partial graph G_t' is generated. If the previous graph's dominating set D no longer dominates G_t', a new dominating set is calculated. If the cardinality of this new dominating set exceeds the current cluster count, the PARTITION function computes the optimal partition associated with the previous iteration, and k is updated. In this PARTITION function, each point in $\mathcal{X}$ is assigned to the nearest dominating vertex. The algorithm stops once $k_{\max}$ clusters are reached and returns the sequence of optimal partitions $(\mathcal{P}_1^*, \mathcal{P}_2^*, \ldots, \mathcal{P}_{k_{\max}}^*)$ for $\mathcal{X}$.

Example 1. The operation of Curgraph is illustrated on a sample dataset with 4 points for which the dissimilarity matrix is given in figure 1a. This dataset is also represented as a graph, displayed in figure 1b. The first partition, $\mathcal{P}_1$, is

Algorithm 1 The Curgraph algorithm

Input: A dataset $\mathcal{X}$ and a maximum number of clusters $k_{\max}$ (defaults to $|\mathcal{X}|$).
Output: A Sequence of partitions of $\mathcal{X}$ $(\mathcal{P}_1^*, \mathcal{P}_2^*, \ldots, \mathcal{P}_{k_{\max}}^*)$, each having the minimum radius for the corresponding cardinality.

function CURGRAPH($\mathcal{X}$, $k_{\max}$)
  $T, \mathcal{P}_1^*, D_1 \leftarrow$ INITIALIZE($\mathcal{X}$)
  $G_t \leftarrow$ PartialGraph($\mathcal{X}, R(\mathcal{P}_1^*)$)
  $D \leftarrow D_1$; $k \leftarrow 2$
  while $k \leq k_{\max}$ **do**
    $t \leftarrow$ Next(T)
    $G_t' \leftarrow$ PARTIALGRAPH($\mathcal{X}, t$)
    if D is not a dominating set of G_t' **then**
      $D' \leftarrow$ COMPUTEMDS(G_t')
      if $|D'| > k$ **then**
        $\mathcal{P}_k^* \leftarrow$ PARTITION(G_t, D)
        $k \leftarrow |D'|$
      $D \leftarrow D'$; $G_t \leftarrow G_t'$
  return $(\mathcal{P}_1^*, \mathcal{P}_2^*, \ldots, \mathcal{P}_{k_{\max}}^*)$



(a) Pairwise dissimilarities (b) Curgraph clusters with all possible thresholds

Fig. 1: Example of Curgraph partitions

computed from its prototype $a = \arg\min_{a \in \mathcal{X}} \max_{b \in \mathcal{X}} d(a, b)$. In this case, any point can be elected as the prototype. We suppose $D_1 = \{x_1\}$ is chosen. The radius of this partition is $R(\mathcal{P}_1^*) = 4$. The ranked list of dissimilarities lower than 4 is $T = [2, 1, 0]$.

Then, $t = 2$, and $\{x_1\}$ no longer dominates the partial graph with edges greater than 2 removed. A call to the MDS solver is then performed to compute a new MDS. There are three solutions in this case: $\{x_1, x_4\}$, $\{x_2, x_4\}$ and $\{x_3, x_4\}$, one of which is returned by the solver. We suppose $\{x_3, x_4\}$ is chosen.

Then, $t = 1$, and $\{x_3, x_4\}$ still dominates the new partial graph. If any other dominating set had been chosen in the previous step, a new call to the MDS solver would have been required, leading in all cases to the only solution $D' = \{x_3, x_4\}$.

Finally, $t = 0$ and $\{x_3, x_4\}$ does not dominate the new partial graph. The MDS solver is called and provides $\{x_1, x_2, x_3, x_4\}$ as a new dominating set. Its cardinality, 4, is greater than $k = 2$. The previous result, $\{x_3, x_4\}$, is therefore the minimum radius partition $\mathcal{P}_2^*$ for $k = 2$.

4.2 Tackling the MDS bottleneck

As discussed in Section 5, computing an exact dominating set can be costly, even if this occurs only on few iterations. Casado et al. [5] proposed an efficient MDS heuristic that could replace the exact method for large datasets, reducing the execution time without significantly affecting the accuracy [10]. The Curgraph algorithm presented in Algorithm 1 remains mostly the same, but since it is based on the assumption that the cardinality of the dominating set is nondecreasing, an additional verification of this property is included, and the algorithm backtracks to update previous results if an inconsistency is detected. Specifically, if $|D'| < k$ after computing a new dominating set D', then partitions $\mathcal{P}^*_{|D'|}, \ldots, \mathcal{P}^*_{k-2}, \mathcal{P}^*_{k-1}$ are invalidated, and the current cluster count k is reset to $|D'|$. Furthermore, in Algorithm 1 the whole `while` loop can be parallelized to compute partitions concurrently, as each iteration only depends on the previous iteration. To do so, the list of thresholds T is split into n sublists, where n is the number of available threads. Each thread computes the partition for its sublist of thresholds. The results are then merged to obtain the final sequence of partitions. In case of overlapping partitions (e.g. multiple threads found a partition with 3 clusters), the partition with the smallest radius is kept. Parallelization induces the extra cost of having to call the MDS solver on the first iteration of each sublist.

5 Experiments & Results

We evaluate Curgraph in its exact and approximate versions, and compare it against Protoclust on seven OpenML datasets [17].

Exact optimal dominating sets are computed using the reference implementation of a branch a bound MDS algorithm proposed by Jiang et al. [14]. The approximate MDS solver used in this paper is a re-implementation in C++ of the algorithm proposed by Casado et al. [5]. Curgraph has no specific feature linked with either implementation, and other exact or approximate MDS solvers could have been considered. Curgraph is written in Python, version 3.11. Protoclust [4] is written in R, version 4.2. All experiments were conducted inside Docker containers on a machine with an Intel Xeon G5118 CPU with 12 cores at 2.30GHz and 64GB of RAM. Each algorithm is run three times to evaluate their stability, with a time limit of 10 minutes[1].

The experiments produce partitions for each dataset from one to a maximum of $k_{\max} = |\mathcal{X}|$ clusters. Computation times and partition radii are recorded. If

[1] The experiment code and data are available at https://forge.lias-lab.fr/curgraph-experiments

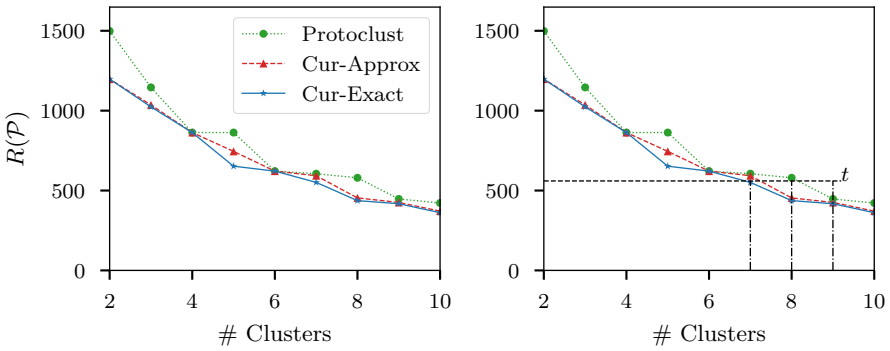


Fig. 2: Compactness of Curgraph and Protoclust partitions on WDBC

Table 1: Mean execution time (in seconds) for each algorithm

Dataset (size)	**Protoclust**	**Cur-Exact**	**Cur-Approx**	**Cur-Parallel**
Iris (150)	0.03±0.00	10.16±0.01	7.99±0.01	2.82±0.07
Wine (178)	0.04±0.00	14.61±0.01	17.37±0.02	8.28±0.20
Glass (214)	0.06±0.00	35.01±0.15	24.76±0.04	8.13±0.05
Ionosphere (351)	0.12±0.00	334.56±0.69	96.34±0.08	23.48±0.04
WDBC (569)	0.58±0.12	TL[1] (14)	379.25±0.48	121.88±0.52
Synt. Cont. (600)	0.49±0.00	TL[1] (10)	257.85±0.77	115.93±2.56
Vehicle (846)	0.99±0.08	TL[1] (3)	497.60±0.90	339.50±8.22

[1] When the time limit is reached, the size of the last optimal partition is given.

one algorithm hits the time limit, the experiment is stopped, but partitions found up to this point are retrieved.

Curgraph-Exact always finds optimal partitions. Curgraph-Approx results are not guaranteed to be optimal and, in practice, 37 % of resulting partitions are not. With Protoclust, 92 % of partitions are suboptimal. Curgraph-Approx partitions are almost always as compact (19.8 %) or more compact (79.7 %) than with Protoclust. Fig. 2 (left) illustrates this behavior by showing $R(\mathcal{P}_k)$ for Curgraph and Protoclust on the WDBC dataset. Curgraph-Exact produces a Pareto front of minimal radius and minimal size partitions, and both Curgraph-Appox and Protoclust generate partitions relatively close to this front. The difference can still have an impact on partitions resulting from cuts performed with a user-provided dissimilarity threshold or target number of clusters. On Fig. 2 (right), a cut at $t = 560$ results in a partition with cardinality 7 with Curgraph-Exact, but cardinality 8 with Curgraph-Approx and 9 with Protoclust.

Table 1 presents total execution times on each dataset. The parallel version of Curgraph—with the approximate MDS solver—is denoted as Cur-Parallel. If an algorithm hits the time limit, the cadinality of the last optimal partition found is indicated in parentheses. Curgraph-Exact does not scale well with larger

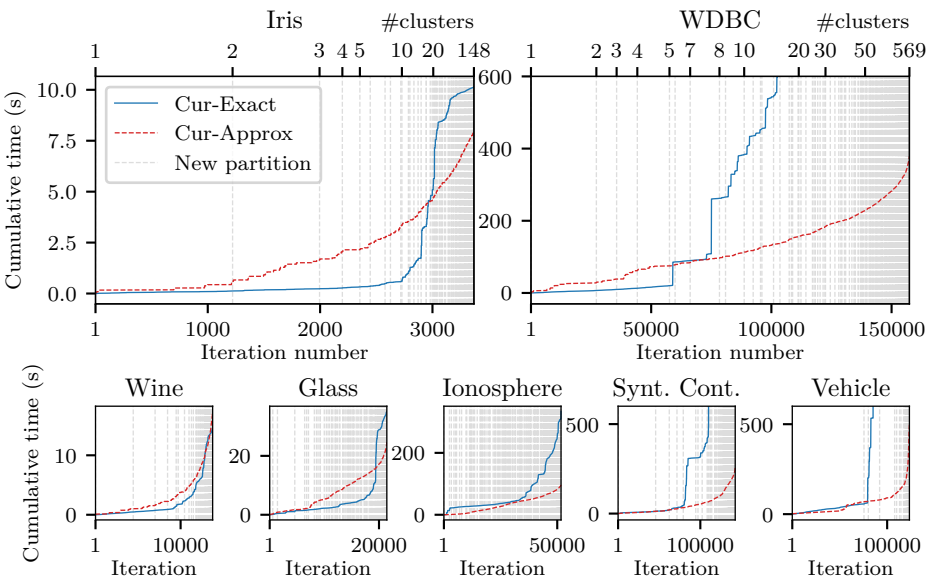


Fig. 3: Cumulative execution time for Curgraph

datasets, and hits the time limit on the three largest, even if those are relatively small. As previously noted, the difficulty of the underlying MDS problem and the frequency of algorithm invocations contribute to Curgraph's computational cost.

Figure 3 shows the cumulative execution time for each non-parallelized variant of Curgraph on each dataset. Surprisingly, on small datasets, Curgraph-Exact is faster than Curgraph-Approx for determining partitions containing small number of clusters. This is a direct consequence of the performance profile of the underlying MDS solver. However, as the threshold t gets smaller—in later iterations—, more calls to the MDS solver are required, which significantly increases the execution time. When $k_{\max} = |\mathcal{X}|$, Curgraph-Approx is more reliable than Curgraph-Exact, and generally faster—the only exception being the wine dataset. Nevertheless, an analyst might be interested only in partitions with a relatively small number of clusters. In this case, Curgraph can be stopped early, when the specified maximum number of clusters is reached, saving time.

Figure 4 illustrates the internal behavior of Curgraph, between the 2850th to 3050th iterations with the Iris dataset—where 68% of the total execution time is spent using the exact version. This figure highlights that most iterations do not require the computation of a minimum dominating set. Each of these *no MDS call* iterations is computed in less than 80 µs. By comparison, iterations that perform an MDS call last between 5 ms and 1.2 s. Table 2 shows that this behavior remains true for all datasets, as iterations relying on MDS occur at most 8% of the time, but account for at least 90% of the total execution time. Overall, results suggest that our assumption of an MDS remaining valid for

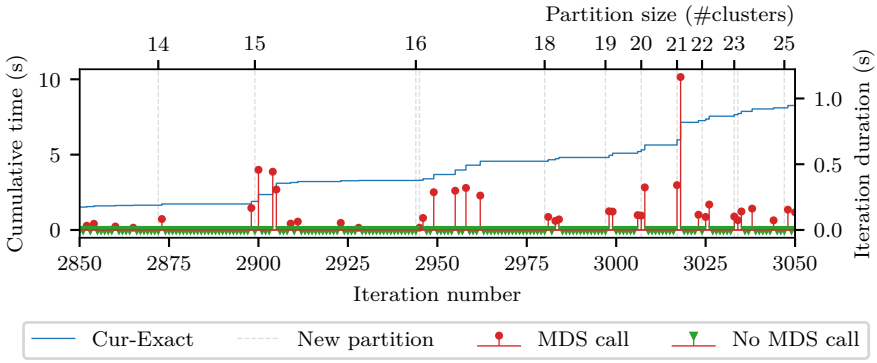


Fig. 4: Analysis of Curgraph-Exact iterations on the Iris dataset

Table 2: Curgraph computation statistics

	Curgraph-Exact			Curgraph-Approx		
Dataset	**# Iter.**	**% MDS calls**	**% MDS duration**	**# Iter.**	**% MDS calls**	**% MDS duration**
Iris	3,369	8.04	99.2	3,369	7.01	99.0
Wine	13,765	4.28	97.5	13,765	3.14	97.9
Glass	21,435	2.38	97.9	21,435	2.19	97.1
Ionosphere	51,668	4.67	98.8	51,668	1.79	95.4
WDBC	102,476[1]	0.26	97.4	157,419	0.85	93.1
Synt. Cont.	110,008[1]	0.09	97.5	142,423	0.85	90.2
Vehicle	85,156[1]	0.11	95.9	123,455	0.93	92.4

[1] Time limit reached.

a significant amount of successive partial graphs is correct. Moreover, results suggest that the larger the dataset, the more MDS computations are skipped relatively to the total number of iterations. This might be the basis for potential further optimization of the algorithm.

As shown in Figure 4, every optimal partition (depicted by a vertical line) requires an MDS call during the subsequent iteration. This step is necessary to guarantee that the previous radius is minimal. An optimal run would only require two MDS calls for each partition size: one at the threshold—or slightly before with the same dominating set—and one just after.

Finally, these results underline the effectiveness of Curgraph for solving the minimum radius clustering problem (Definition 3). The enhanced partition compactness achieved with this method over Protoclust offers a valuable trade-off despite the increased computation time.

6 Discussions and Conclusions

Optimal algorithms for clustering under radius constraints remain underexplored, with prior work using alternative paradigms like a diameter constraint, a mathematical norm, a dissimilarity threshold or a fixed cluster count, which are not aligned with our goals.

We introduce Curgraph, addressing these challenges to deliver optimal or near-optimal solutions for radius-based clustering. Experimental results show that Curgraph achieves optimal solutions at the cost of longer execution times compared to current state-of-the-art algorithms. Given the NP-hardness of the problem, this limitation was expected. Curgraph remains valuable for applications needing compact clusters and minimal parameter tuning, especially for smaller datasets

To address scalability, we developed an approximate approach, significantly reducing execution time while maintaining compact partitions, though this solution remains much slower than HAC with minimax linkage (Protoclust) on larger datasets. To the best of our knowledge, Curgraph is the first alternative solution to Protoclust to generate sequences of compact partitions w.r.t. their radius. Experimentally, both the exact and the approximate versions of Curgraph offer greater compactness over Protoclust. These better results come at the expense of higher computational costs, and the loss of the hierarchical structure of resulting partitions. For the most part, the execution time of Curgraph is due to the underlying MDS solver. Faster heuristics could make Curgraph compete with Protoclust.

Apart from work on the MDS solver, future directions include enhancing the parallelization process of Curgraph, taking into account the density distribution of dissimilarities to better divide the workload among threads. We might also rethink the implementation of the main loop in Algorithm 1 to reduce the overhead of non MDS calling iterations and improve the overall efficiency of Curgraph, while maintaining the quality of the partitions. This could be achieved, for example, by skipping some inconsequential iterations or by using more efficient data structures.

Overall, Curgraph offers a promising direction for radius-constrained clustering, although more work is needed to close the efficiency gap.

Acknowledgments. This work was supported by the @LIENOR ANR LabCom (ANR-19-LCV2-0006) and the French region Nouvelle-Aquitaine.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

Bibliography

[1] Andersen, J., Chardin, B., Tribak, M.: Clustering to the Fewest Clusters Under Intra-Cluster Dissimilarity Constraints. In: 2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI), pp. 209–216 (2021), https://doi.org/10.1109/ICTAI52525.2021.00036

[2] Ao, S.I., et al.: CLUSTAG: hierarchical clustering and graph methods for selecting tag SNPs. Bioinformatics pp. 1735–1736 (2005), https://doi.org/10.1093/bioinformatics/bti201

[3] Berndt, D.J., Clifford, J.: Using dynamic time warping to find patterns in time series. In: Proceedings of the 3rd international conference on knowledge discovery and data mining, pp. 359–370 (1994)

[4] Bien, J., Tibshirani, R.: Hierarchical Clustering With Prototypes via Minimax Linkage. Journal of the American Statistical Association pp. 1075–1084 (2011), https://doi.org/10.1198/jasa.2011.tm10183

[5] Casado, A., Bermudo, S., López-Sánchez, A.D., Sánchez-Oro, J.: An iterated greedy algorithm for finding the minimum dominating set in graphs. Mathematics and Computers in Simulation pp. 41–58 (2023), https://doi.org/10.1016/j.matcom.2022.12.018

[6] Dao, T.B.H.e.a.: Constrained clustering by constraint programming. Artificial Intelligence pp. 70–94 (2017), https://doi.org/10.1016/j.artint.2015.05.006

[7] Davidson, I., Ravi, S.: Clustering with constraints: Feasibility issues and the k-means algorithm. In: Proceedings of the 2005 SIAM international conference on data mining, pp. 138–149, SIAM (2005)

[8] Dinler, D., Tural, M.K.: A Survey of Constrained Clustering. In: Unsupervised Learning Algorithms, pp. 207–235 (2016), https://doi.org/10.1007/978-3-319-24211-8_9

[9] Gordon, A.D.: A survey of constrained classification. Computational Statistics & Data Analysis pp. 17–29 (1996), https://doi.org/10.1016/0167-9473(95)00005-4

[10] Haenn, Q., Chardin, B., Baron, M.: Clustering Under Radius Constraints Using Minimum Dominating Sets. In: 27th International Symposium on Methodologies for Intelligent Systems (2024), https://doi.org/10.1007/978-3-031-62700-2_2

[11] Hansen, P., Delattre, M.: Complete-Link Cluster Analysis by Graph Coloring. Journal of the American Statistical Association pp. 397–403 (1978), https://doi.org/10.1080/01621459.1978.10481589

[12] Hansen, P., Jaumard, B.: Cluster analysis and mathematical programming. Mathematical programming **79**(1), 191–215 (1997)

[13] Hubert, L.J.: Some applications of graph theory to clustering. Psychometrika pp. 283–309 (1974), https://doi.org/10.1007/BF02291704

[14] Jiang, H., Zheng, Z.: An Exact Algorithm for the Minimum Dominating Set Problem. In: Proceedings of the 32nd International Joint Conference

on Artificial Intelligence, pp. 5604–5612 (2023), https://doi.org/10.24963/ijcai.2023/622
[15] Shi, M., Hua, K., Ren, J., Cao, Y.: Global optimization of k-center clustering. In: Proceedings of the 39th International Conference on Machine Learning, vol. 162, pp. 19956–19966 (2022)
[16] Tai, X.H., Frisoli, K.: Benchmarking Minimax Linkage (2019)
[17] Vanschoren, J., van Rijn, J.N., Bischl, B., Torgo, L.: Openml: networked science in machine learning. SIGKDD Explorations pp. 49–60 (2013), https://doi.org/10.1145/2641190.2641198