Ceccarello *et al. Journal of Big Data* (2023) 10:49
https://doi.org/10.1186/s40537-023-00717-4

Journal of Big Data

RESEARCH Open Access

Scalable and space-efficient Robust Matroid Center algorithms

Matteo Ceccarello[1*], Andrea Pietracaprina[1], Geppino Pucci[1] and Federico Soldà[2]

*Correspondence:
matteo.ceccarello@unipd.it
[1] University of Padova, Padua, Italy
[2] ETH Zurich, Zurich, Switzerland

Abstract

Given a dataset V of points from some metric space, a popular robust formulation of the k-center clustering problem requires to select k points (centers) of V which minimize the maximum distance of any point of V from its closest center, excluding the z most distant points (outliers) from the computation of the maximum. In this paper, we focus on an important constrained variant of the robust k-center problem, namely, the Robust Matroid Center (RMC) problem, where the set of returned centers are constrained to be an independent set of a matroid of rank k built on V. Instantiating the problem with the partition matroid yields a formulation of the fair k-center problem, which has attracted the interest of the ML community in recent years. In this paper, we target accurate solutions of the RMC problem under general matroids, when confronted with large inputs. Specifically, we devise a coreset-based algorithm affording efficient sequential, distributed (MapReduce) and streaming implementations. For any fixed $\varepsilon > 0$, the algorithm returns solutions featuring a $(3+\varepsilon)$-approximation ratio, which is a mere additive term ε away from the 3-approximations achievable by the best known polynomial-time sequential algorithms. Moreover, the algorithm obliviously adapts to the intrinsic complexity of the dataset, captured by its doubling dimension D. For wide ranges of k, z, ε, D, our MapReduce/streaming implementations require two rounds/one pass and substantially sublinear local/working memory. The theoretical results are complemented by an extensive set of experiments on real-world datasets, which provide clear evidence of the accuracy and efficiency of our algorithms and of their improved performance with respect to previous solutions.

Introduction

Center-based clustering is a crucial primitive for data management. In general terms, given a dataset V, a distance function between pairs of points in V, and a value k, a solution for center-based clustering is a set of k representative points, called *centers*, which induce a partition of V into k subsets (clusters), each containing all points in V closest to the same center. One important formulation of center-based clustering is the *k-center* problem, where the set of centers must be chosen as a subset of V which minimizes the maximum distance of any point of V to its closest center. It is well known that k-center is NP-hard, that it admits a 2-approximation algorithm, and that for any $\varepsilon > 0$ it is not $(2-\varepsilon)$-approximable unless $P = NP$ [1].

Since the k-center objective function involves a maximum, the optimal solution is at risk of being severely influenced by a few "distant" points in V, called *outliers*. In fact, the presence of outliers is inherent in many datasets, since these points are often due to artifacts or errors in data collection. To cope with this issue, k-center admits the following *robust* formulation that takes into account outliers [2]: given an additional input parameter z, when computing the k-center objective function, the z points of V with the largest distances from their respective centers are disregarded when taking the maximum.

Some applications may need that the solution of the (robust) k-center problem satisfy additional constraints. For example, the set of centers may be required to be an independent set of a given matroid $M = (V, I)$, where V is the ground set and I is the family of independent sets. A robust formulation of the k-center problem under a general matroid constraint, referred to as *Robust Matroid Center* (RMC), has been recently studied in [3–5]. Matroid constraints arise naturally in the context of content distribution networks and facility location [6, 7]. Moreover, the special case of partition matroid can be used to model fairness constraints where the points of V are naturally subdivided into $m \leq k$ groups (e.g., demographic groups) and *fair* solutions to k-center are sought which include k_i points from the ith group, for given k_i's such that $\sum_{i=1}^{m} k_i = k$ [8, 9]. The k-center problem under a matroid constraint is NP-hard, and, for any $\varepsilon > 0$, it is not $(2 - \varepsilon)$-approximable unless $P = NP$ [3]. This hardness results hold even for points on a line and clearly extend to the robust version RMC as well.

Due to the ever increasing need of efficient tools for analyzing large datasets, it is paramount to devise clustering strategies amenable to the typical computational frameworks employed for big data processing, such as MapReduce and streaming [10]. *Coreset-based* strategies have recently emerged as ideal approaches for big data processing [11]. Informally, these strategies entail the (efficient) extraction of a very succinct summary T (dubbed *coreset*) of the dataset V, so that a solution for V can be obtained by running (suitable modifications of) the best sequential algorithm on T. Coreset constructions that can be either parallelized or streamlined efficiently, yield scalable and space-efficient algorithms in the big data realm. In this paper, we devise a novel coreset-based algorithm for the RMC problem, featuring efficient sequential, distributed and streaming implementations. Our distributed implementation is specified using the MapReduce framework which is recognized as one of the reference models for the efficient distributed processing of large datasets, but it can be immediately ported to other distributed frameworks (e.g., the Massively Parallel Computation model (MPC) [12]).

Previous work

For brevity, we only report on the works most closely related to the specific topic of this paper, and refer the interested reader to [13] for a more comprehensive overview of center-based clustering. The most accurate sequential algorithms to date for the RMC problem are the sequential 3-approximations presented in [4, 5], whose running times (not explicitly quantified in the papers) are polynomials of likely high degree due to the use of dynamic and linear programming. A simpler combinatorial algorithm, more amenable to practical implementation, is the 7-approximation of [3]. All of these algorithms are not immediately portable to the MapReduce or streaming settings. A coreset-based streaming algorithm for RMC has been recently devised by Kale in [14]. For $\varepsilon > 0$, the

algorithm computes a coreset of size $O(k(k+z)\log(1/\varepsilon)/\varepsilon)$, which contains a $(15+\varepsilon)$ -approximate solution, where z is the number of outliers and k is the rank of the matroid. This solution can then be extracted from the coreset using exhaustive search in time exponential in k. Alternatively, one of the sequential approximation algorithms in [4, 5] can be run on the coreset to yield a $(51+\varepsilon)$-approximate solution. Kale's strategy tests and updates, in parallel, several guesses on the optimal value of the objective function, and computes, for each guess, a suitable coreset through an adaptation of the k-center streaming strategy in [15].

The special case of RMC with the partition matroid and $z = 0$ (i.e., without outliers) has been recently studied in the context of *fair k-center clustering*, where the matroid constraint is used to enforce that given fractions of the centers be picked from the matroid categories. A streaming $(3+\varepsilon)$-approximation algorithm running in two passes, and a distributed $(17+\varepsilon)$-approximation algorithm running in two rounds and using sublinear local memory, are presented in [9], together with results of some experiments carried out, however, on very small datasets.

Coreset-based algorithms for the unconstrained (robust) k-center and related problems, suitable for the MapReduce and streaming settings, can be found in [16–20]. Coreset-based techniques have also found applications in other domains, such as graph processing [21, 22]. Useful techniques to deal with matroid constraints in big data scenarios have been introduced in [23, 24] in the realm of diversity maximization.

To the best of our knowledge, no distributed (e.g., MapReduce) algorithms for the general RMC problem have been presented in the open literature.

Our algorithms are analyzed in terms of the *doubling dimension* [25], which along with the *VC-dimension* [26], the *local intrinsic dimensionality* [27], the *expansion* [28], and the *relative contrast* [29] is a useful measure to characterize the behavior of algorithms.

Our contribution

We present a novel coreset-based algorithm for the RMC problem which features an approximation ratio close to the best one attainable sequentially, and admits efficient sequential, distributed (MapReduce) and streaming implementations, affording a dramatic performance improvement over the existing sequential approaches, hence proving suitable for dealing with massive inputs. By leveraging ideas introduced in [14, 23, 30], our algorithm makes pivotal use of the unconstrained k-center primitive to extract a small coreset T from the ground set V of the input matroid $M = (V, I)$, with the property that the distance between each point of V and the closest point of T is a small fraction of the cost of the optimal solution. In order to cope with the matroid constraint, T is built in such a way that for every independent set in I there is a "pointwise close" independent set in T. Consequently, T contains a good solution for the original problem on V, which can be computed by assigning a suitable multiplicity to each point of T, and feeding T to the best-known sequential algorithms for RMC, adapted to take multiplicities into account. The performance of the algorithm is analyzed in terms of the number of outliers z, the rank k of matroid M, an accuracy parameter ε capturing the approximation quality, and the *doubling dimension* D of the ground set V, a parameter that generalizes the notion of Euclidean dimension to arbitrary metric spaces.

The main contributions of our paper are the following.

- A coreset-based RMC algorithm called RobustMatroidCenter that, for any fixed $\varepsilon \in (0,1)$, attains an $(\alpha+\varepsilon)$-approximation ratio, where α is the approximation ratio of the sequential RMC algorithm (generalized to handle multiplicities) run on the coreset (Theorem 6).
- A sequential implementation of RobustMatroidCenter, which attains the same approximation ratio, and requires time $O\big(|V|\ \mathrm{poly}(k,z,(c/\varepsilon)^D)\big)$, for a suitable constant $c>0$. This running time is linear in $|V|$ for a wide range of parameters.
- A distributed implementation of RobustMatroidCenter in MapReduce, which attains the same approximation ratio, runs in 2 rounds, and requires $O\Big(\sqrt{|V|\ \mathrm{poly}(k,z)}(c/\varepsilon)^D\Big)$ memory at each worker, for a suitable constant $c>0$ (Theorem 9). This local memory bound is substantially sublinear in $|V|$ for a wide range of parameters.
- A streaming implementation of RobustMatroidCenter which attains the same approximation ratio, runs in 1 pass, and uses $O\big(\ \mathrm{poly}(k,z)(c/\varepsilon)^D\big)$ working memory, for a suitable constant $c>0$ (Theorem 11). This working memory bound is independent of $|V|$.
- A rich suite of experiments demonstrating the competitiveness of our algorithm, both in the distributed and in the streaming setting, when applied to large datasets of up to several million points, which are out of reach of the current sequential algorithms with best approximation ratios (section "Experiments").

We also show how to adapt the sequential RMC algorithm in [4] to handle multiplicities, retaining approximation ratio 3. Therefore, RobustMatroidCenter yields a $(3+\varepsilon)$-approximation in all computational frameworks. In the distributed setting, is proved in [9] that an approximation ratio less than 4 for unconstrained k-center (hence, for RMC) cannot be achieved with sublinear communication volume. In contrast, our MapReduce implementation achieves sublinear communication volume (matching the local memory requirement) for low dimensional spaces and small values of k and z, thus showing that the lower bound can be beaten in these cases.

We remark that, while the analysis is performed in terms of the doubling dimension D of V, RobustMatroidCenter is *oblivious to the value* D, in the sense that this value, which is hard to compute and expensive to approximate [31], is not used by the algorithm. We also observe that, as it is often the case in dimensionality-aware analyses, time and space requirements exhibit an exponential dependence on D, hence they are mostly effective for low-dimensional datasets. Nevertheless, in our experiments we did not observe the degradation exponential D suggested by the theory.

Our results improve upon the state of the art as follows.

- We provide the first distributed solution to RMC for general matroids, with an approximation guarantee comparable to one of the best sequential algorithm. Also, compared to the MapReduce algorithm for fair k-center clustering of [9] (i.e., $z=0$ and the partition matroid), we improve the approximation ratio from $(17+\varepsilon)$ to $(3+\varepsilon)$, at the expense of a modest memory blow-up, for ground sets of low doubling dimension.

- In the streaming setting, we substantially improve upon the approximations attained in [14] for general matroids, by virtue of a different construction strategy that yields higher-precision coresets, at the expense of a modest blow-up in the coreset size, for ground sets of low doubling dimension. Also, for fair k-center clustering we obtain the same approximation ratio of [9] in one pass rather than two.
- To the best of our knowledge, we carry out the first experimentation to date of RMC algorithms for general matroids. The experiments demonstrate that confining the execution of an expensive accurate sequential strategy to small subsets of the ground set, affords the solution of large instances and yields dramatic improvements in running time, while maintaining roughly the same approximation quality

The rest of the paper is organized as follows. Section "Preliminaries" defines key properties of matroids, formally defines the RMC problem, and describes the MapReduce and streaming computational settings. Algorithm RobustMatroidCenter is presented and analyzed in section "Coreset-based strategy for the RMC problem", while its MapReduce and streaming implementations are described in section "MapReduce and streaming implementations". The results of the experiments are reported in section "Experiments". section "Concluding remarks" closes the paper with some concluding remarks. Finally, at the end of the paper a technical appendix describes the extension of the RMC algorithm of [4] to handle multiplicities, which is needed by our approach.

Preliminaries

Matroids

Let V be a ground set of elements from a metric space with distance function $d(\cdot,\cdot)$ satisfying the triangle inequality. A *matroid* [32] on V is a pair $M = (V, I)$, where I is a family of subsets of V, called *independent sets*, satisfying the following properties: (i) the empty set is independent; (ii) every subset of an independent set is independent (*hereditary property*); and (iii) if $A \in I$ and $B \in I$, and $|A| > |B|$, then there exist $x \in A \backslash B$ such that $B \cup \{x\} \in I$ (*augmentation property*). An independent set is *maximal* if it is not properly contained in another independent set. A basic property of a matroid M is that all of its maximal independent sets have the same size. The notion of maximality can be naturally extended to any subset of the ground set. Namely, for $V' \subseteq V$, an independent set $A \subseteq V'$ of maximum cardinality among all independent sets contained in V' is called a maximal independent set of V', and all maximal independent sets of V' have the same size. We let the *rank* of a subset $V' \subset V$, denoted by $rank(V')$ to be the size of a maximal independent set in V'. The rank of the matroid $rank(M)$ is then defined as $rank(V)$. An important property of the rank function is *submodularity*: for any $A, B \subseteq V$ it holds that $rank(A \cup B) + rank(A \cap B) \leq rank(A) + rank(B)$. The following lemma is an adaptation of [14, Lemma 3] and provides a useful property of matroids which will be exploited to derive the results of this paper.

Lemma 1 (Extended augmentation property) *Let $M = (V, I)$ be a matroid. Consider an independent set $A \in I$, a subset $V' \subseteq V$, and an independent set $B \subseteq V'$ which is maximal within V'. If there exists $y \in V' \backslash A$ such that $A \cup \{y\} \in I$, then there exists $x \in B \backslash A$ such that $A \cup \{x\} \in I$.*

Proof Since B is maximal in V', we have that $rank(B \cup \{y\}) = rank(B) = rank((B \cup \{y\}) \cap (A \cup B))$. Also, $rank((B \cup \{y\}) \cup (A \cup B)) \geq rank(A \cup \{y\}) \geq |A| + 1$, since $A \cup \{y\} \in I$. By applying the submodularity property to sets $B \cup \{y\}$ and $A \cup B$ we have the inequality $rank((B \cup \{y\}) \cup (A \cup B)) + rank((B \cup \{y\}) \cap (A \cup B)) \leq rank(B \cup \{y\}) + rank(A \cup B)$. Therefore, $rank(A \cup \{y\}) + rank(B \cup \{y\}) \leq rank((B \cup \{y\}) \cup (A \cup B)) + rank((B \cup \{y\}) \cap (A \cup B)) \leq rank(B \cup \{y\}) + rank(A \cup B)$, whence $rank(A \cup B) \geq rank(A \cup \{y\}) \geq |A| + 1$. So, there exists an independent set $C \subseteq A \cup B$ of $|A| + 1$ elements, and the lemma follows. □

Given a matroid $M = (V, I)$ and a subset $V' \subseteq V$, we define the *restriction of M to V'* as $M_{V'} = (V', I_{V'})$, where $I_{V'} = \{X \cap V' : X \in I\}$. It is easy to see that $M_{V'}$ is also a matroid.

Robust Matroid Center

The well-known *k-center problem* is defined as follows. Given a set V of points from a metric space with distance function $d(\cdot, \cdot)$, determine a subset $S \subseteq V$ of size k which minimizes $\max_{i \in V} d(i, S)$.[1] We let $\rho^*(V, k)$ denote the cost of the optimal solution. In this paper, we focus on the following variant of the k-center problem, defined below using the same terminology adopted in [5].

Definition 1 Let $M = (V, I)$ be a matroid defined over the set of points V, and let z be an integer, with $0 \leq z < |V|$. The **Robust Matroid Center (RMC) problem** on M with parameter z, requires to determine an independent set $S \in I$ minimizing

$$r(S, V, z) = \min_{X \subseteq V : |X| \geq |V| - z} \max_{i \in X} d(i, S).$$

We use the tuple $(M = (V, I), z)$ to denote an instance of RMC, and let $r^*(M, z)$ denote the cost of its optimal solution. It is immediate to see that the objective function $r(S, V, z)$ corresponds to the $(|V| - z)$-th smallest distance of a point of V from S. In other words, the best solution is allowed to ignore the contribution of the z most distant points, which can be regarded as *outliers*. Note that if the matroid (V, I) has rank k, any feasible solution $S \in I$ has size at most k. Also, note that the standard k-center problem is a special case of RMC problem where $z = 0$, and the set I of independent sets consists of all subsets of size at most k.

The state of the art on sequential solution for the problem is the 3-approximation algorithm presented in [4]. The coreset-based approaches developed in this paper require the solution of a generalized version of the problem, where each point $i \in V$ comes with a positive integer multiplicity m_i. Let $\mu_V = \sum_{i \in V} m_i$. The generalized version, dubbed **RMC problem with Multiplicities (RMCM problem)**, allows z to vary in $[0, \mu_V)$ and modifies the cost function as follows:

$$r(S, V, z) = \min_{X \subseteq V : \sum_{i \in X} m_i \geq \mu_V - z} \max_{i \in X} d(i, S).$$

Letting $\mathbf{m} = \left(m_1, \ldots m_i, \ldots m_{|V|}\right)$, we use the tuple $(M = (V, I), z, \mathbf{m})$ to denote instances of RMCM. To the best of our knowledge, prior to this work, no algorithms had been devised to solve the RMCM problem. However, we can show that the sequential

[1] For convenience, we use the notation $d(i, S) = \min_{c \in S} d(i, c)$.

algorithms in [4] can be adapted to solve the RMCM problem, featuring the same approximation guarantees as in the case without multiplicities. This is summarized in the following theorem (proof in Appendix).

Theorem 2 *There exists a 3-approximate polynomial-time sequential algorithm for the RMCM problem.*

Doubling dimension

The algorithm in this paper will be analyzed in terms of the dimensionality of the ground set V as captured by the well-established notion of doubling dimension. Formally, given a point $i \in V$, let the *ball of radius r centered at i* be the subset of points of V at distance at most r from i. The *doubling dimension* of V is the smallest value D such that any balls of radius r centered at a point $i \in V$ is contained in the union of at most 2^D balls of radius $r/2$ suitably centered at points of V. The algorithms that will be presented in this paper adapt automatically to the doubling dimension D of the input dataset and attain their best performance when D is small, possibly constant. This is the case, for instance, of ground sets V whose points belong to low-dimensional Euclidean spaces, or represent nodes of mildly-expanding network topologies under shortest-path distances.

The doubling dimension D of a ground set V allows to establish the following interesting relation between the radius of a clustering and its granularity, which will be crucially exploited in this paper.

Proposition 1 *Let $\varepsilon \in (0,1)$. Consider a set $S \subseteq V$, and let $\rho = \max_{i \in V} d(i,S)$. If V has doubling dimension D, there exists a set $S' \subseteq V$ of size $\leq |S|(2/\varepsilon)^D$ such that $\max_{i \in V} d(i,S') \leq \varepsilon\rho$.*

Proof By repeatedly applying the definition of doubling dimension, it is easily seen that each ball of radius ρ around a point in S can be covered with at most $(2/\varepsilon)^D$ smaller balls of radius at most $\varepsilon\rho$. The centers of all of these smaller balls provide the desired set S'. □

Computational settings

In recent years, *MapReduce* [10, 33, 34] has become one of the reference models for the efficient distributed processing of large datasets. In particular, MapReduce has proven to be an effective computational model for clustering problems [18, 35, 36]. A MapReduce algorithm executes as a sequence of parallel *rounds*. In a round, a multiset X of key-value pairs is first transformed into a new multiset X' of pairs by applying a given *map function* to each individual pair, and then into a final multiset Y of pairs by applying a given *reduce function* (referred to as *reducer*) independently to each subset of pairs of X' having the same key. The model is parametrized by total aggregate memory available to the computation, denoted with $\mathcal{M}_A$, and the maximum amount of memory locally available to each reducer, denoted with $\mathcal{M}_L$. We remark that, although we use the MapReduce abstraction for the high level design and analysis of our algorithms, the whole approach can be straightforwardly rephrased for other distributed models, such as the popular *Massively Parallel Computation* (MPC) model [12]. Furthermore, we implement our

distributed algorithms in the state-of-the-art *timely dataflow* system [37], which provides superior performance with respect to more established distributed frameworks supporting a MapReduce programming style.

The need to cope with data produced at high rates, which cannot be stored for offline processing, has led to the emergence of the *streaming* setting [10, 38] where the computation is performed by a single processor with a small working memory, and the input is provided as a continuous stream of items, usually too large to fit in the working memory. Typically, streaming strategies aim at a single pass on the input but in some cases few additional passes may be needed. Key performance indicators are the size of the working memory and the number of passes.

The holy grail of big data algorithmics is the development of MapReduce (resp., streaming) algorithms which work in as few rounds (resp., passes) as possible and require substantially sublinear local memory (resp., working memory) and linear aggregate memory.

Coreset-based strategy for the RMC problem

In this section, we present a two-phase strategy to solve the RMC problem based on the following simple high-level idea. In the first phase, a small *coreset* $T \subseteq V$ is extracted from the ground set V, with the property that each point $j \in V$ has a suitably "close" *proxy* $p(j)$ in T. While the algorithm does not explicitly store the proxy function, it stores, with each point in $i \in T$, its multiplicity m_i, defined as the number of distinct points $j \in V$ whose proxy is i. In the second phase, an approximate solution S to the RMCM problem is computed on T (efficiently, due to T's small size). The key ingredient of our strategy is that for each independent set X of the input matroid there is an independent set $X' \subseteq T$ whose elements are pointwise "close" to those of X. This fact will allow us to show that S is also a good solution for the RMC problem on V. The section is structured as follows. Section "Coreset construction" describes and analyzes the construction of T, while section "Extraction of the solution from the coreset" discusses how to extract an accurate solution for V from the coreset T with its multiplicities.

Coreset construction

Let $(M = (V, I), z)$ be an instance of the RMC problem. As in previous works, we assume that constant-time oracles are available to compute the distance between two elements of V and to check whether a subset of V is an independent set (see e.g., [39]). We let k be the rank of matroid M, and make the reasonable assumption that k is provided in input together with the instance.

In order to construct the coreset T, we first determine a β-approximate solution S to the unconstrained $(k + z)$-center problem on V, for some constant $\beta > 0$, and compute its cost $\rho = \max_{i \in V} d(i, S)$. Hence, $\rho \leq \beta \rho^*(V, k + z)$. The value β will depend on the employed approximation algorithm, which, in turn, depends on the computational setting. For instance, in the sequential setting, Gonzalez's algorithm [1] provides $\beta = 2$, while in the streaming setting, Gonzalez's algorithm cannot be used, and a larger value of β will be needed [15, 40].

As a next step, we fix a suitable accuracy parameter $\varepsilon' \in (0, 1)$ and determine a set T' of points of V such that $d(i_1, i_2) > (\varepsilon'/(2\beta))\rho$, for every $i_1 \neq i_2 \in T'$, and

$d(j, T') \leq (\varepsilon'/(2\beta))\rho$, for every $j \in V \setminus T'$. We define $r_{T'} = \max_{i \in V} d(i, T')$, hence, $r_{T'} \leq (\varepsilon'/(2\beta))\rho$. T' can be computed greedily by performing a linear scan of V, starting with an initially empty T', and iteratively adding to T' each point that is at distance greater than $(\varepsilon'/(2\beta))\rho$ from the current T' [41].

Let $T' = \{i_1, i_2, \ldots, i_\tau\}$, for some value τ which is a function of ρ, β, and ε'. For $1 \leq \ell \leq \tau$, define the cluster $C_\ell = \{j \in V : d(j, i_\ell) = d(j, T')\}$ (ties broken arbitrarily for points $j \in V$ equidistant from two or more points of T_τ). From each C_ℓ we extract a maximum independent set[2] Y_ℓ and define

$$T = \cup_{1 \leq \ell \leq \tau} Y_\ell.$$

For every $1 \leq \ell \leq \tau$ and every point $j \in C_\ell$, we set the proxy $p(j) = i \in Y_\ell$, where $d(j, i) = d(j, Y_\ell)$ (ties broken arbitrarily). Moreover, for each $i \in T$, we compute its multiplicity as $m_i = |\{j \in V : p(j) = i\}|$. The above construction is implemented by Procedure CoresetConstruction in the pseudocode provided as Algorithm 1.

In the rest of this subsection, we show that T is a good representative for the ground set V, and provide a bound on its size, in terms of the doubling dimension of V. We first determine sufficient conditions on *any coreset* $Q \subseteq V$, which guarantee that a good solution to the RMCM problem on Q is also a good solution for the RMC problem on V, and then we prove that the coreset T built above satisfies these conditions. Consider a coreset $Q \subseteq V$ with proxy function $p : V \to Q$, and let $m_i = |\{j \in V : p(j) = i\}|$, for every $i \in Q$. Let $M_Q = (Q, I_Q)$ denote the restriction of matroid $M = (V, I)$ to the coreset Q, where $I_Q = \{X \cap Q : X \in I\}$. Finally, let $(M_Q, z, \mathbf{m})$ denote the RMCM instance defined by M_Q, z and $\mathbf{m} = \{m_i : i \in Q\}$. The following lemma holds

Lemma 3 *Suppose that the coreset Q with proxy function $p : V \to Q$ satisfies the following conditions, for a given $\varepsilon' \in (0, 1)$:*

C1 *For each $j \in V$, $d(j, p(j)) \leq \varepsilon' r^*(M, z)$;*
C2 *For each independent set $X \in I$ there exists an injective mapping $\pi_X : X \to Q$ such that:*

- *$\{\pi_X(i) : i \in X\} \subseteq Q$ is an independent set;*
- *for each $i \in X$, $d(i, \pi_X(i)) \leq \varepsilon' r^*(M, z)$.*

Then:

P1 *There exists a solution to $(M_Q = (Q, I_Q), z, \mathbf{m})$ of cost at most $(1 + 2\varepsilon')r^*(M, z)$;*
P2 *Every solution S to $(M_Q, z, \mathbf{m})$ of cost r_S is also a solution to $(M = (V, I), z)$ of cost at most $r_S + \varepsilon' r^*(M, z)$.*

Proof Let us first show P1. Let X_V^* be the optimal solution to the RMC instance $(M = (V, I), z)$ and let $Y = \{\pi_{X_V^*}(o) : o \in X_V^*\} \subseteq Q$. We will show that Y is a solution for

[2] For ease of presentation, we are assuming that C_ℓ always contains a non-empty independent set. If this were not the case, it would be sufficient to set Y_ℓ to a singleton consisting of an arbitrary element of C_ℓ, so that C_ℓ is represented in the final coreset.

$(M_Q, z, \mathbf{m})$ of cost at most $(1+2\varepsilon')r^*(M,z)$. By C2, $|Y| = |X_V^*|$ and Y is an independent set in I_Q. Consider now a point $j \in V$ such that $\exists o \in X_V^*$ with $d(j,o) \leq r^*(M,z)$ and observe that there are at least $|V| - z$ such points (e.g., all nonoutliers). We have that

$$\begin{aligned} d(p(j), Y) \leq & d(p(j), \pi_{X_V^*}(o)) \\ \leq & d(p(j), j) + d(j, o) + d(o, \pi_{X_V^*}(o)) \\ & \text{(by triangle inequality)} \\ \leq & \varepsilon' r^*(M,z) + r^*(M,z) + \varepsilon' r^*(M,z) \\ & \text{(by C1 and C2)} \\ \leq & (1+2\varepsilon') r^*(M,z). \end{aligned}$$

Let $\mu_Q = \sum_{i \in Q} m_i$ and observe that $\mu_Q = |V|$. We have that

$$\begin{aligned} & \sum_{i \in Q : d(i,Y) \leq (1+2\varepsilon')r^*(M,z)} m_i \geq \\ & \geq \sum_{i \in Q : \exists j \in V : (i = p(j)) \wedge (d(j, X_V^*) \leq r^*(M,z))} m_i \\ & \geq |\{j \in V : d(j, X_V^*) \leq r^*(M,z)\}| \\ & \geq |V| - z, \end{aligned}$$

which concludes the proof of P1. In order to prove P2, let S be a solution to $(M_Q, z, \mathbf{m})$ of cost r_S. Clearly, S is an independent set in I. Consider a generic point $i \in Q$ such that $d(i,S) \leq r_S$ and let a be the point of S closest to i. Observe that the m_i points $j \in V$ with $i = p(j)$ are such that $d(j,S) \leq d(j,a) \leq d(j,i) + d(i,a) \leq \varepsilon' r^*(M,z) + r_S$. Since $\sum_{i \in Q : d(i,S) \leq r_S} m_i \geq \mu_Q - z$, there are at least $\mu_Q - z = |V| - z$ points of V that are within a distance $\varepsilon' r^*(M,z) + r_S$ from S. □

We have:

Lemma 4 *The coreset T returned by Procedure* CoresetConstruction$(M = (V, I), z, \varepsilon')$ *satisfies Conditions C1 and C2 of Lemma 3, hence, it exhibits Properties P1 and P2 of that lemma.*

Proof First, we prove C1. Consider an arbitrary point $j \in V$, and suppose that j belongs to cluster C_ℓ, for some ℓ. Thus, $p(j)$ belongs to $Y_\ell \subseteq C_\ell$ and $d(j, p(j)) \leq 2r_{T'} \leq (\varepsilon'/\beta)\rho$. Since any solution to the $(M = (V, I), z)$ instance of RMC, with the addition of the z outlier points as extra centers, is a solution to $(k+z)$-center on V, it is easy to see that $\rho^*(V, k+z) \leq r^*(M,z)$. Now, by using the fact that ρ is the cost of a β-approximate solution to $(k+z)$-center on V, we have

$$d(j, p(j)) \leq (\varepsilon'/\beta)\rho \leq \varepsilon' \rho^*(V, k+z) \leq \varepsilon' r^*(M,z),$$

thus proving C1. As for C2, we reason as follows. Consider an arbitrary independent set $X \in I$. We now show that there exists an injective mapping π_X which transforms X into an independent set contained in T, and such that, for each $j \in X$, if j belongs to cluster C_ℓ, for some $1 \leq \ell \leq \tau$, then also $\pi_X(j) \in C_\ell$. This will immediately imply that $d(j, \pi_X(j)) \leq 2r_{T'} \leq \varepsilon' r^*(M,z)$. Let $X = \{x_a : 1 \leq a \leq |X|\}$. We

define the mapping π_X incrementally one element at a time. Suppose that we have fixed the mapping for the first $h \geq 0$ elements of X, and assume, inductively, that $W(h) = \{\pi_X(x_a) : 1 \leq a \leq h\} \cup \{x_a : h < a \leq |X|\}$ is an independent set of size $|X|$ and that x_a and $\pi_X(x_a)$ belong to the same cluster, for every $1 \leq a \leq h$. Consider now x_{h+1}, and suppose that $x_{h+1} \in C_\ell$, for some ℓ. We distinguish among the following two cases:

- **Case 1.** If $x_{h+1} \in Y_\ell$, we set $\pi_X(x_{h+1}) = x_{h+1}$, hence $W(h+1) = W(h)$.
- **Case 2.** If $x_{h+1} \notin Y_\ell$, we apply the extended augmentation property stated in Lemma 1 with $A = W(h) \setminus \{x_{h+1}\}$, $y = x_{h+1}$, $V' = C_\ell$, and $B = Y_\ell$ to conclude that there exists a point $\pi_X(x_{h+1}) \in B \setminus A = Y_\ell \setminus (W(h) \setminus \{x_{h+1}\})$ such that $W(h+1) = (W(h) \setminus \{x_{h+1}\}) \cup \pi_X(x_{h+1})$ is an independent set.

After $|X|$ iterations of the above inductive argument, we have that the mapping π_X is completely specified and exhibits the following properties: it is injective, $\{\pi_X(x_a) : 1 \leq a \leq |X|\}$ is an independent set, and, for $1 \leq a \leq |X|$, if $x_a \in C_\ell$ then also $\pi_X(x_a) \in C_\ell$, hence $d(x_a, \pi_X(x_a)) \leq \varepsilon' r^*(M, z)$. This proves C2. □

The size of coreset T can be conveniently bounded as a function of the doubling dimension of the ground set V.

Theorem 5 *If V has doubling dimension D, then the coreset T obtained with the above construction has size at most $k(k+z)(8\beta/\varepsilon')^D$.*

Proof Since the matroid $M = (V, I)$ has rank k, we have that $|T| \leq k\tau$, hence we are left to bound τ. Consider the first set S of $k+z$ centers computed by the coreset construction algorithm (i.e., the one that provided the value $\rho = \max_{i \in V} d(i, S) \leq \beta \rho^*(V, k+z)$). Proposition 1 implies that there exists a set S' of at most $h = (k+z)(8\beta/\varepsilon')^D$ points such that $\max_{i \in V} d(i, S') \leq (\varepsilon'/(4\beta))\rho$, hence V can be covered with h balls of radius at most $(\varepsilon'/(4\beta))\rho$. It is easy to see that the greedy strategy used to construct T', picks at most one point from each such ball. Hence, $\tau = |T'| \leq h$, and the theorem follows. □

Extraction of the solution from the coreset

Once the coreset T with the multiplicities $\mathbf{m} = \{m_i : i \in T\}$ is computed from V, the final solution S to instance $(M = (V, I), z)$ of RMC is obtained by running an approximation algorithm $\mathcal{A}$ for RMCM on instance $(M_T = (T, I_T), z, \mathbf{m})$. (The pseudocode for both the coreset construction and the extraction of the final solution is provided by Algorithm 1.)

The following theorem establishes an upper bound to the approximation obtainable for the RMC problem.

Theorem 6 *Let $\varepsilon \in (0, 1)$ and suppose that an α-approximation algorithm $\mathcal{A}$ for RMCM is available. If the coreset T exhibits Properties P1 and P2 of Lemma 3 with $\varepsilon' = \varepsilon/(2\alpha+1)$ then the solution, S returned by running $\mathcal{A}$ on instance $(M_T = (T, I_T), z, \mathbf{m})$ of RMCM, is an $(\alpha+\varepsilon)$-approximate solution to instance $(M = (V, I), z)$ of RMC.*

Proof By Property P1 of Lemma 3, we know that the optimal solution to $(M_T = (T, I_T), z, \mathbf{m})$ has cost at most $(1 + 2\varepsilon')r^*(M, z) = (1 + 2\varepsilon/(2\alpha + 1))r^*(M, z)$. Hence, S has cost $r_S \leq (\alpha + 2\alpha\varepsilon/(2\alpha + 1))r^*(M, z)$. By Property P2 of Lemma 3, S is also a solution to instance $(M = (V, I), z)$ of RMC with cost $r_S + \varepsilon' r^*(M, z) \leq (\alpha + \varepsilon)r^*(M, z)$. □

Using the 3-approximation algorithm for RMCM from Theorem 2 as $\mathcal{A}$ in the above theorem, immediately yields the following corollary.

Corollary 7 *Algorithm* RobustMatroidCenter *can be used to compute a* $(3 + \varepsilon)$ *-approximate solution to any instance* $(M = (V, I), z)$*, for any fixed* $\varepsilon \in (0, 1)$.

Algorithm 1: RobustMatroidCenter$(M = (V, I), k, z, \varepsilon))$

```
Let 𝒜 be an α-approximation algorithm for RMCM;
ε' ← ε/(2α + 1);
(T, m) ← CoresetConstruction(M, k, z, ε');
return 𝒜(M_T = (T, I_T), z, m);

Procedure CoresetConstruction(M, k, z, ε')
k ← rank(M)
S ← β-approximate solution to (k + z)-center on V;
ρ ← max_{i∈V} d(i, S); T' ← ∅;
foreach i ∈ V do
    if (d(i, T') > (ε'/(2β))ρ) then T' ← T' ∪ {i};
end
τ ← |T'|;
Let 𝒞 = {C_1, ..., C_τ} be the clustering of V induced by T';
T ← ∅;
foreach C_ℓ ∈ 𝒞 do
    Y_ℓ ← maximal independent set in C_ℓ;
    T ← T ∪ Y_ℓ;
    foreach i ∈ Y_ℓ do
        m_i ← |{j ∈ C_ℓ : d(j, Y_ℓ) = d(j, i)}|
        {ties broken arbitrarily}
    end
end
return (T, m = {m_i : i ∈ T})
```

Remarks A straightforward consequence of Theorems 2 and 5, and of Corollary 7, is that there is a sequential $(3 + \varepsilon)$-approximation algorithm for the RMC problem which runs in time $O\big(|V| \text{ poly}(k, z, (c/\varepsilon)^D)\big)$ when given in input an instance $(M = (V, I), z, \varepsilon))$, where c is a suitable constant and D is the doubling dimension of V. Therefore, if k, z, ε and D are constants, the running time is linear in $|V|$. Moreover, an exhaustive search on the coreset can yield a tighter $(1 + \varepsilon)$-approximate solution while maintaining the running time linear in $|V|$, although exponential in the other parameters. It is also important to observe that, while the analysis of our algorithm is performed in terms of the doubling dimension D of V, the algorithm itself is **oblivious to the value** D , in the sense that D is not explicitly used by the algorithm, and becomes very efficient for spaces of low doubling dimension.

MapReduce and streaming implementations

In this section, we present efficient implementations of RobustMatroidCenter in the MapReduce (section "MapReduce implementation") and streaming (section "Streaming implementation") settings.

MapReduce implementation

In MapReduce, RobustMatroidCenter can be implemented in two rounds, where the first round computes the coreset T and the second round extracts the solution S from T. While, in the second round, the small size of T affords computing S by running a sequential RMCM algorithm on T in one reducer, the computation of T in the first round involves the whole (possibly very large) ground set V, and thus requires a careful implementation able to exploit parallelism while keeping memory requirements suitably low.

A key feature the RMC coreset construction presented in the previous section is its *composability* [42], a property referring to the fact that the coreset for the entire set V can be obtained as the union of partial coresets, one for each subset of an arbitrary partition of V. The next lemma establishes this property.

We have:

Lemma 8 *Given a matroid $M = (V, I)$, consider an arbitrary partition of V into ℓ disjoint subsets $V_1, \ldots, V_\ell$, with $\ell \geq 1$. For $1 \leq q \leq \ell$, let T_q be the coreset returned by* CoresetConstruction $(M_{V_q} = (V_q, I_{V_q}), z, \varepsilon'/2)$, *with proxy function $p_q : V_q \to T_q$, where M_{V_q} is the restriction of M to V_q. Then, the coreset $T = \cup_{1 \leq q \leq \ell} T_q$, with proxy function $p : V \to T$ defined as $p(i) = p_q(i)$ for $i \in V_q$, satisfies the conditions C1 and C2 of Lemma 3*

Proof The proof follows by repeating the same argument used in the proof of Lemma 4, with the only modification that, for any subset V_q, a simple application of the triangle inequality suffices to show that $\rho^*(V_q, k+z) \leq 2\rho^*(V, k+z)$. □

As an immediate consequence of the above lemma, a coreset T satisfying Conditions C1 and C2 of Lemma 3 can be constructed in one MapReduce round as follows. Partition V evenly but arbitrarily into ℓ disjoint subsets $V_1, \ldots, V_\ell$, and assign each V_q to a distinct reducer, which builds a coreset T_q for V_q by invoking CoresetConstruction $(M_q = (V_q, I_{V_q}), z, \varepsilon'/2)$, instantiated with the $(\beta = 2)$-approximation algorithm by Gonzalez [1] to find the initial solution to $(k+z)$-center. In a second round, the T_q's are gathered into the final coreset $T = \cup_{1 \leq q \leq \ell} T_q$, and a solution can be computed from T using a single reducer running $\mathcal{A}(M_T = (T, I_T), z, \mathbf{m})$, where $\mathcal{A}$ is a sequential approximation algorithm for the RMCM problem.

Setting $\ell = \sqrt{|V|/(k(k+z))}$ and applying Theorem 5 (with $\varepsilon'/2$) we have that $|T| = O(\sqrt{|V|k(k+z)}(16/\varepsilon')^D)$. Observe that for a large range of values of k and z, the size of each V_q and the size of T are substantially sublinear in $|V|$. The following theorem is an immediate consequence of the above discussion and of the results of Theorem 6.

Theorem 9 *Let $\varepsilon \in (0, 1)$ and suppose that a sequential α-approximation algorithm for RMCM is available, for some constant $\alpha > 0$. Then, there exists a 2-round MapReduce algorithm that for the RMC instance $(M = (V, I), z)$ computes an $(\alpha + \varepsilon)$-approximate solution using $\mathcal{M}_A = O(|V|)$ and $\mathcal{M}_L = O(\sqrt{|V|k(k+z)}(c/\varepsilon)^D)$, for a suitable constant c.*

By using the result of Theorem 2, the value $\alpha = 3$ can be plugged in the above theorem.

Streaming implementation

In the streaming setting, the coreset construction devised in section Coreset-based strategy for the RMC problem can be easily implemented in two passes, where the first pass computes the initial solution to $(k+z)$-center on V, using the scaling algorithm of [15], and the second pass computes the coreset T, together with the multiplicities, through an adaptation of the greedy linear scan of Lines 9–11 of Algorithm 1, so to compute concurrently the points of T' and their associated independent sets. At the end of the second pass, the final solution is obtained by running the sequential approximation algorithm $\mathcal{A}$ on T. We describe below how the two passes can be reduced to one, by exploiting ideas similar to those in [24, 40].

Let V_t denote the first t points of the ground set stream V, and let $\delta, \varepsilon' \in (0,1)$ be two fixed accuracy parameters. For each $t \geq k+z+1$, our implementation maintains the following data:

- A set $S_t \subseteq V_t$ of $k+z$ centers and a value $\rho'_t \leq (2+\delta)\rho^*(V_t, k+z)$ such that for every $i \in V_t$, $d(i, S_t) \leq \rho'_t$.
- A set $T'_t \subseteq V_t$ with the property that for every $i \neq j \in T'_t$: $d(i,j) > (\varepsilon'/(2\beta))R$, and for every $i \in V_t$: $d(i, T'_t) \leq (\varepsilon'/\beta)R$, where R is a suitable value in $[\rho'_t/2, \rho'_t]$ and $\beta = 2 \cdot (2+\delta)$.
- An implicit partition V_t into clusters, $C_1, C_2, \ldots, C_{|T'_t|}$, such that all points of any cluster C_ℓ are at distance at most $(\varepsilon'/\beta)R$ from the ℓ-th point of T'_t (call it i_ℓ), which is regarded as the center of C_ℓ. The partition is not explicitly stored, but for each C_ℓ, the algorithm maintains its size and a maximal independent set $Y_\ell \subseteq C_\ell$, with multiplicities assigned to its points so that they add up to $|C_\ell|$.

For any t, the set S_t and the value ρ'_t are maintained through the scaling algorithm of [15]. For $t = k+z+1$, R is set equal to ρ'_t, while T'_t, the independent sets Y_ℓ, and the related multiplicities, are initialized by performing a simple scan of V_t. For $t > k+z+1$, let i be the point arriving at time t. If $d(i, T'_{t-1}) > (\varepsilon'/(2\beta))R$ then $T'_t = T'_{t-1} \cup \{i\}$ and i will make a singleton cluster and independent set with multiplicity 1. Otherwise, $T'_t = T'_{t-1}$ and i will be implicitly added to a cluster C_ℓ such that the distance between i and the center i_ℓ of C_ℓ is at most $(\varepsilon'/\beta)R$. Also, i is added to Y_ℓ if $Y_\ell \cup \{i\}$ stays independent. In both cases, the multiplicities in Y_ℓ are updated to reflect that there is a new point in C_ℓ.

As soon as the current estimate ρ'_t becomes greater than $2R$, the algorithm sets R equal to ρ'_t (thus R increases by a factor at least 2) and "shrinks" T'_t by iteratively eliminating each point that is within distance $(\varepsilon'/(2\beta))R$ from another point, so to re-establish the separation property stated above. When a point $i_\ell \in T'_t$ is eliminated from T'_t because of another point $i_{\ell'} \in T'_t$, with $d(i_\ell, i_{\ell'}) \leq (\varepsilon'/(2\beta))R$, the cluster C_ℓ corresponding to i_ℓ is implicitly merged with Cluster $C_{\ell'}$. To reflect such a merging, the old subset $Y_{\ell'}$ is reassigned to a maximal independent set obtained from $Y_\ell \cup Y_{\ell'}$, and the multiplicities of

the points in the new $Y_{\ell'}$ are chosen to add up to $|C_\ell| + |C_{\ell'}|$. The correctness of this procedure (i.e., the fact that the new subset $Y_{\ell'}$ is a maximal independent set of $C_\ell \cup C_{\ell'}$) follows by the extended augmentation property of Lemma 1.

At the end of the stream, the coreset T is set equal to the union of the Y_ℓ's, with multiplicities inherited from those of the Y_ℓ's. Finally, the solution to the $(M = (V, I), z)$ instance to the problem is computed as $\mathcal{A}(M_T = (T, I_T), z, \mathbf{m})$, where $\mathcal{A}$ is a sequential approximation algorithm for the RMCM problem.

Lemma 10 *The coreset T computed at the end of the stream has size at most $k(k+z)(32(2+\delta)/\varepsilon')^D$, where D is the doubling dimension of V. Moreover, it satisfies Conditions C1 and C2 of Lemma 3, hence, it exhibits Properties P1 and P2 of that lemma.*

Proof Let $n = |V|$. By construction, $|T| \leq k|T'_n|$, hence, it is sufficient to upper bound $|T'_n|$. At the end of the stream, S_n is a set of $k+z$ centers with the guarantee that $d(i, S_n) \leq \rho'_n \leq (2+\delta)\rho^*(V_n, k+z)$ for every $i \in V_n = V$, and for any two points $i \neq j \in T'_n$ we have that $d(i,j) > (\varepsilon'/(2\beta))R \geq (\varepsilon'/(4 \cdot (2+\delta)))(\rho'_n/2) \geq (\varepsilon'/(8(2+\delta)))\rho'_n$. By Proposition 1 we have that V can be covered with at most $(k+z)(32(2+\delta)/\varepsilon')^D$ balls of radius at most $(\varepsilon'/(16(2+\delta)))\rho'_n$. Clearly, no two points of T'_n can reside in any such ball, which implies $|T'_n| \leq (k+z)(32(2+\delta)/\varepsilon')^D$. In order to show that T satisfies Conditions C1 and C2 of Lemma 3, we first observe that, for every $i \in V$, if i belongs to cluster C_ℓ with center $i_\ell \in T'_n$, we have that the proxy $p(i)$ of i is a point of Y_ℓ, hence $d(i, p(i)) \leq 2d(i, i_\ell) \leq (\varepsilon'/(2+\delta))R \leq (\varepsilon'/(2+\delta))\rho'_n \leq \varepsilon'\rho^*(V_n, k+z) \leq \varepsilon' r^*(M, z)$. Then, the same proof of Lemma 4 can be repeated. □

Note that the proof of the above lemma can be immediately generalized to show that at any time t, the aggregate size of all Y_ℓ's is at most $k(k+z)(32(2+\delta)/\varepsilon')^D$. Since, the scaling algorithm used to maintain S_t requires working memory $O\big((k+z)(1/\delta)\log(1/\delta)\big)$ [15], we have that the dominant factor in the working memory requirements is the aggregate size of the Y_ℓ's. For a fixed constant δ, the following theorem is an immediate consequence of the above discussion and of the results of Lemma 10 and Theorem 6.

Theorem 11 *Let $\varepsilon \in (0,1)$ and suppose that a sequential α-approximation algorithm for RMCM is available, for some constant $\alpha > 0$. There exists a 1-pass streaming algorithm that for the RMC instance $(M = (V, I), z)$ computes an $(\alpha + \varepsilon)$-approximate solution using working memory of size $O\big(k(k+z)(c/\varepsilon)^D\big)$, for a suitable constant c.*

Again, a $(3+\varepsilon)$-approximation can be attained by using the RMCM algorithm of Theorem 2.

Experiments

We carry out an experimental evaluation of RobustMatroidCenter and its sequential, MapReduce and streaming implementations, aimed at answering the following questions:

- What is the impact of the coreset size on performance, both in terms of quality and running time?

- How does the coreset-based strategy used by RobustMatroidCenter compare against existing sequential and streaming baselines?
- To what extent does the distributed implementation of the coreset construction benefit from parallelism?
- How expensive is to build the coreset, compared to finding a good solution to the RMCM problem on the coreset itself?

Our experiments focus on the robust formulation of the problem under general matroids, and for this reason we do not compare with the algorithms from [8, 9], which are restricted to the non-robust case under the partition matroid.

Experimental setup We implemented all algorithms using Rust 1.53.0-nightly (`ca075d268` 2021-04-28). The MapReduce implementation of our coreset approach runs on top of Timely Dataflow [37]. All experiments have been executed on a cluster with 8 machines equipped with a Intel®Xeon®CPU (E5-2670 v2 @ 2.50GHz), and 16 Gb of RAM each. Each result is the average over at least 10 runs.

Algorithms For RobustMatroidCenter, we developed a sequential implementation (dubbed SeqCoreset), a streaming algorithm (dubbed StreamingCoreset), and a MapReduce implementation (dubbed MRCoreset). As baselines to compare with, we implemented the sequential 7-approximation algorithm from [3] (dubbed ChenEtAl), which we have straightforwardly extended to handle multiplicities, and the streaming algorithm from [14] (dubbed KaleStream). We also used the multiplicity-enhanced ChenEtAl implementation in all coreset-based algorithms to extract the final solution from the coreset. In fact, we did not make use of the algorithm of [4] which, although theoretically more accurate, does not admit an efficient implementation due to its recourse to the ellipsoid method, while ChenEtAl features a simpler and faster implementation. All our implementations are available as open source code[3] and adopt a framework which easily allows to tweak parameters [43]. We stress that the main takeaway from our experiments is that coresets provide a dramatic reduction in the size of the data fed to expensive sequential strategies, at the cost of a limited loss in approximation quality. This holds irrespective of whether the algorithms in [3] or [4] are used.

In all experiments, rather than setting the parameter ε, which in the previous sections relates the quality of the solution to the size of the coreset, we control directly the number of cluster centers around which the coresets are built. We denote this parameter with τ. This is to have greater control on the size of the coreset, which will be $\leq \tau \cdot \text{rank}(\mathcal{M})$, with the goal of observing the effect on the performance both in terms of time and quality. While adapting the sequential and MapReduce coreset constructions to this end is straightforward (we just run the algorithm from [1] aiming for τ clusters), adapting the streaming construction requires more care. To this end, rather than maintaining the guess on the radius by means of several instances run in parallel, as in the scaling algorithm of [15], we fix the number of centers to τ and update the radius guess following the schedule of the doubling algorithm of [40]. Each cluster center is associated to a maximal independent set of the points closest to the center seen so far, and such sets are

[3] https://github.com/Cecca/macaco.

Table 1 Datasets used in this experimental evaluation

Dataset	n	Dim	Matroid	Rank
`Higgs`	11,000,000	7	Partition	20
`Phones`	13,062,475	3	Partition	35
`Wiki`	4,976,753	10	Transversal	50

merged whenever the radius guess is updated, dropping points as needed so to maintain independence.

Datasets We consider three datasets in this experimental evaluation, whose characteristics are summarised in Table 1. The scripts to automatically download and preprocess the datasets are available in the code repository.

`Higgs` [44] is a 7-dimensional dataset of 11 M simulated readings from a particle detector. Each reading is characterized by 28 attributes, where 7 of them are a function of the other 21. We used only the 7 derived attributes. Each reading is classified as being either *signal* or *background*. We define a rank-20 partition matroid on these two classes by allowing up to 10 points from each class in any independent set.

`Phones` [45] is a 3-dimensional dataset of over 13 M sensor readings from phones, each tagged with one of seven activities (stand, sit, walk, bike, stairs up, stairs down, *null*). We define a partition matroid on these activities by allowing up to 5 points from each activity in any independent set.

`Wiki` is derived from a recent snapshot of about 5 M pages of the English Wikipedia [46]. Each page is mapped to a 10-dimensional vector using GloVe [47]. Furthermore, we use Latent Dirichlet Allocation [48] to derive a set of 50 categories, together with a probability distribution over these categories for each page. We then assign each page to the most likely categories, so to obtain a transversal matroid of rank 50. The use of LDA is motivated by the fact that the original categories of Wikipedia are over one million, and using all of them to define a transversal matroid would make the matroid constraint immaterial.

For each dataset, we set the number of allowed outliers to 50, 100, or 150. Allowing for more outliers would make the set of non-outlier points more cohesive, so that simply sampling a random independent set would yield a competitive solution compared to the one obtained with ChenEtAl). In the following, we report the results for 50 outliers. The results for 100 and 150 outliers are very similar and are not reported for brevity.

Influence of coreset size

First, we consider the effect of varying the size of the coreset on accuracy and the running time of SeqCoreset and StreamingCoreset. We vary τ from 1 to 10: note that $\tau = 1$ is a degenerate configuration where the coreset is an arbitrary independent set. Lacking exact solutions, we evaluate the quality of the solution in terms of the ratio between the returned radius and the smallest radius found on that dataset *by any tested configuration*.

Figure 1 reports the results of this experiment. First, the top plots of the figure show that increasing τ quickly improves the solution before hitting a plateau, where building larger coresets does not improve the quality of the solution. On `Higgs` this effect is

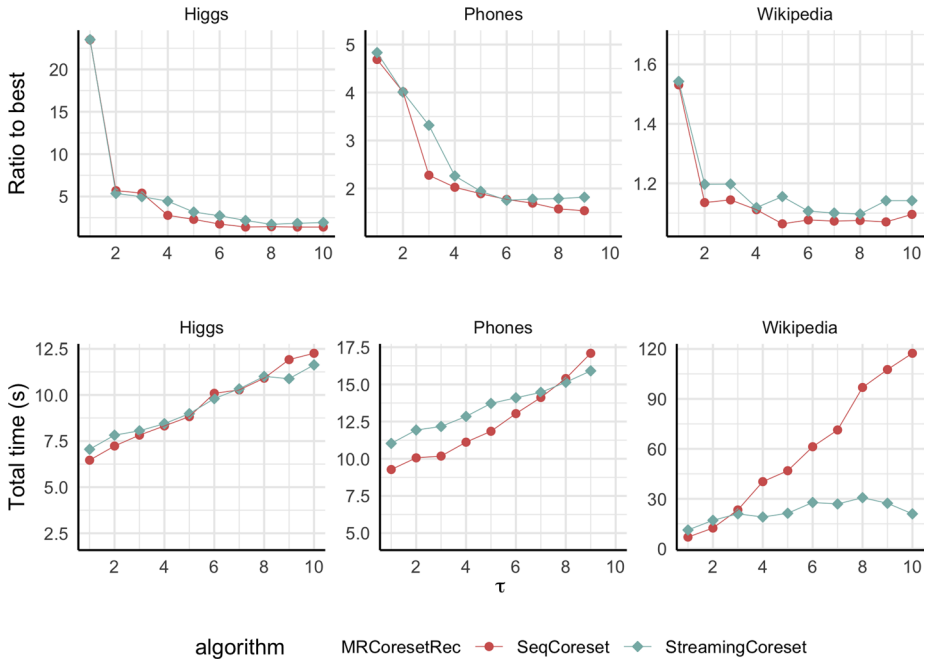

Fig. 1 Effect of the parameter τ on the solution quality (top) and the running time (bottom) for SeqCoreset and StreamingCoreset

more pronounced, whereas for `Wiki` already using a small coreset yields a good-quality solution. Furthermore, we notice that StreamingCoreset and SeqCoreset have comparable quality, with the former performing slightly worse for the same value of τ, as predicted by the theory.

The bottom part of Fig. 1 reports the running times of these experiments, showing that SeqCoreset scales linearly with τ, as expected. As for StreamingCoreset, its performance is comparable with SeqCoreset with some interesting behavior: for `Wiki`, increasing τ beyond 4 does not imply a linear degradation of the performance. The reason is that StreamingCoreset in practice builds coresets smaller than the maximum allowed $\tau \cdot \text{rank}(\mathcal{M})$: after the last doubling of the radius guess the $< \tau$ cluster centers are able to accommodate all the remaining points in the stream, thus building less clusters than the maximum allowed budget. Furthermore, we stress that StreamingCoreset works on unbounded streams of data in a single pass using limited memory (more on this in what follows).

Second, we compare the performance of SeqCoreset with ChenEtAl. Due to the high complexity of ChenEtAl, we are unable to run it on the full datasets, hence we also consider samples of 10,000 points from each dataset. Table 2 reports the result obtained by SeqCoreset with large coresets ($\tau \approx 10$) with ChenEtAl both in terms of approximation quality and running time. First and foremost, we note that SeqCoreset is over two orders of magnitude faster than ChenEtAl. At the same time, the approximation quality is comparable, if not better (e.g. on `Phones`). In fact, recall that SeqCoreset actually runs ChenEtAl on the coreset to obtain the output solution. Results in Table 2 suggest that using a coreset to reduce the size of the input fed to the slower

Table 2 Comparison of SeqCoreset and ChenEtAl on a sample of 10,000 points from each dataset

	Total time (s)		Ratio to best	
Dataset	ChenEtAl	SeqCoreset	ChenEtAl	SeqCoreset
Higgs	1215	3.17	1.000	1.022
Phones	2640	5.13	1.135	1.045
Wiki	5758	18.11	1.039	1.073

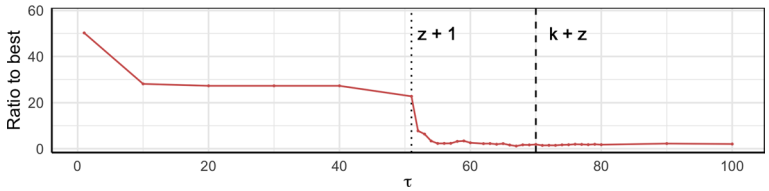

Fig. 2 Ratio to best solution of SeqCoreset on an instance of Higgs with artificially inserted outliers

approximation algorithm not only does not impact quality, but, in some cases, it may help the greedy choices of the slower algorithm, thus leading to better approximations.

The reader might have noticed that we have set τ, i.e. the parameter controlling the number of clusters out of which to build the coreset, to values much smaller than the minimum $k + z$ mandated by the theory (see section "Coreset construction). Nonetheless, we observed that in the real datasets that we considered, the quality of the approximation improves quickly with τ, reaching a plateau for values of τ much smaller than $k + z$. We now explore a setting in which using larger values of τ is paramount to achieving a good approximation quality. Consider the Higgs dataset (for other datasets the results are similar and omitted for brevity), and let c_{MEB} and r_{MEB} be the center and radius of the minimum enclosing ball of the dataset, respectively. Now, sample $z = 50$ points at random, and relocate them on the sphere centered at c_{MEB} of radius $100 \cdot r_{MEB}$: clearly, these z points are extreme outliers. Note that each of these z points still belongs to its original category in the partition matroid.

We run SeqCoreset on this artificial dataset, reporting the results in Fig. 2. We observe that for $\tau < z + 1$ the approximation ratio is very high. In this configuration, some of the outliers are included in the clusters defining the coreset along with points from the bulk of the dataset: they might therefore be part of the independent set representing the cluster, thus receiving a multiplicity larger than 1, making them non-outliers in the final execution of ChenEtAl on the coreset. Starting from $\tau = z + 1$ we have a rapid improvement of the approximation quality, similar to what we observed in Fig. 1, but with values of τ shifted by z. Indeed, in these configurations, the coreset construction will place each outlier in its own cluster, allowing ChenEtAl to handle it appropriately when building the solution on the coreset.

The intuition for this behavior is the following: when the outliers are very far from the bulk of the dataset, assigning them too high a multiplicity in the coreset is detrimental, so we need to put them in their own cluster, so that they have unit multiplicity. On the other hand, when outliers are not so extreme (like in the case of the original datasets), in practice a much smaller τ allows to achieve very good approximations.

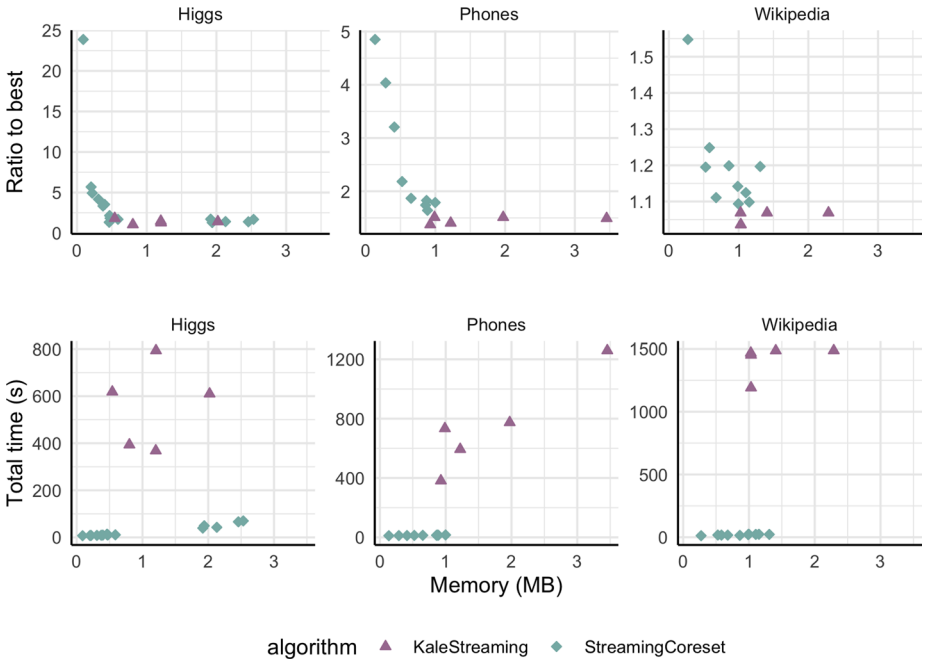

Fig. 3 Approximation ratio and total running time for StreamingCoreset and KaleStream against memory usage. The time scale is logarithmic

Comparison of streaming algorithms

We now compare our streaming algorithm StreamingCoreset with KaleStream. Since the two algorithms are parameterized differently, we study their performance in terms of approximation ratio and running time parameterized by the amount of memory used. Note that for StreamingCoreset a larger τ implies a higher memory usage, for KaleStream the memory increases by decreasing ε. Figure 3 reports on the results of these experiments, fixing the number of outliers to 50. We observe that in general, for comparable amounts of memory, StreamingCoreset is able to find approximations comparable to KaleStream in much less time. The memory used by both algorithms is negligible and, most importantly, not related to the size of the dataset.

Scalability of the MapReduce coreset construction

We now focus on the cost of building the coreset using the MRCoreset algorithm, compared to SeqCoreset. We test 2, 4, and 8 workers, and values of $\tau \in [1, 9]$, but plot the results only for $\tau = 3, 6, 9$, for brevity. Note that for the same value of τ, the size of the aggregated coreset will change depending on the number of workers. As for the number of allowed outliers, we fix it to 100, noting that it does not influence the time required to build the coreset itself.

Figure 4 reports the results of this experiment, focusing just on the corset construction time, which is the part of the algorithm carried out in parallel. For fixed values of τ, the MRCoreset coreset construction scales linearly with the number of processors. Furthermore, in most cases (except $\tau = 3, 6$ for Phones) MRCoreset is faster than SeqCoreset when using 2 processors, i.e., it has a COST metric of 2 [49]. This is due to the

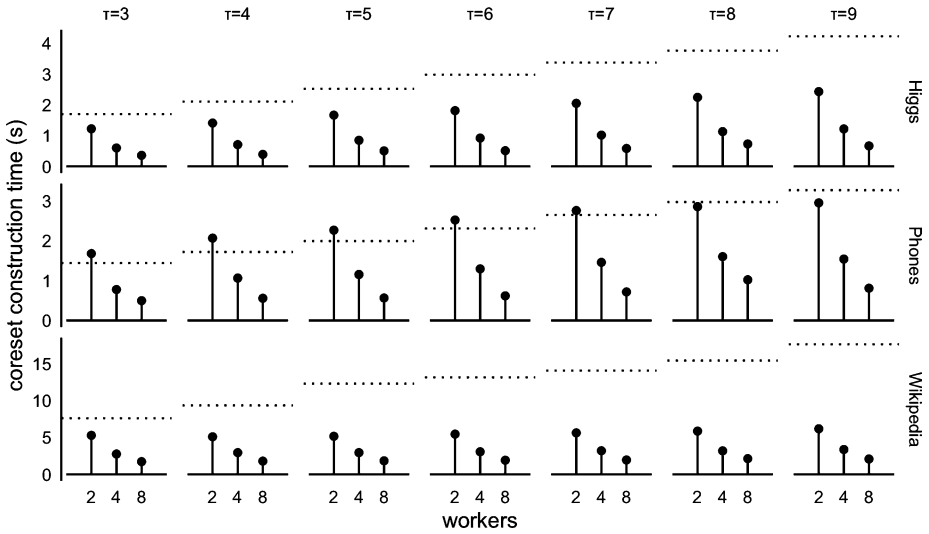

Fig. 4 Time to build the coreset with the MRCoreset algorithm, for varying number of workers. Each column of plots corresponds to a different value of τ. The dotted line represents the performance of the SeqCoreset algorithm in constructing the coreset. The time scale is linear

overheads introduced by the communication required to distribute data to processors located on different machines.

Balance between the coreset construction and solution computation

We now discuss the balance between the time required to build the coreset and the time to compute the solution on the coreset. In MRCoreset we fix the number of workers to 8. The results are reported in Fig. 5, where each line corresponds to a value of τ for a given combination of algorithm and dataset. Each line is divided in two: the left (red) part represents the time to compute the solution, the right (blue) part is the time to build the coreset. Therefore, the span of each line, from dot to dot, is the total running time of a particular configuration.

In general, building a larger coreset clearly tilts the balance towards the more expensive sequential computation of the final solution. This is to be expected, since the coreset construction scales linearly in τ, whereas the computation of the solution is at least quadratic in the coreset size. Interestingly, for any given value of τ, the time to compute the solution on the coreset built by StreamingCoreset is less then the that for SeqCoreset. The reason lies in the behavior of the streaming construction algorithm. Adjusting its guess of the radius to remain within the allotted budget of centers τ, it may end up not using it completely, thus building smaller coresets compared to SeqCoreset.

For MRCoreset, we have that the computation of the solution dominates the running time by far. This is due to the size of the coreset built in parallel, which, for the same τ, is larger than the one computed by SeqCoreset, by a factor (roughly) equal to the number of workers. This problem can be ameliorated as follows. After building the coreset in parallel, SeqCoreset can be applied to reduce the size of the resulting set of points. While this might incur into a worsening of the approximation ratio, it would dramatically speed up the computation of the final solution, thus amplificating the benefits of

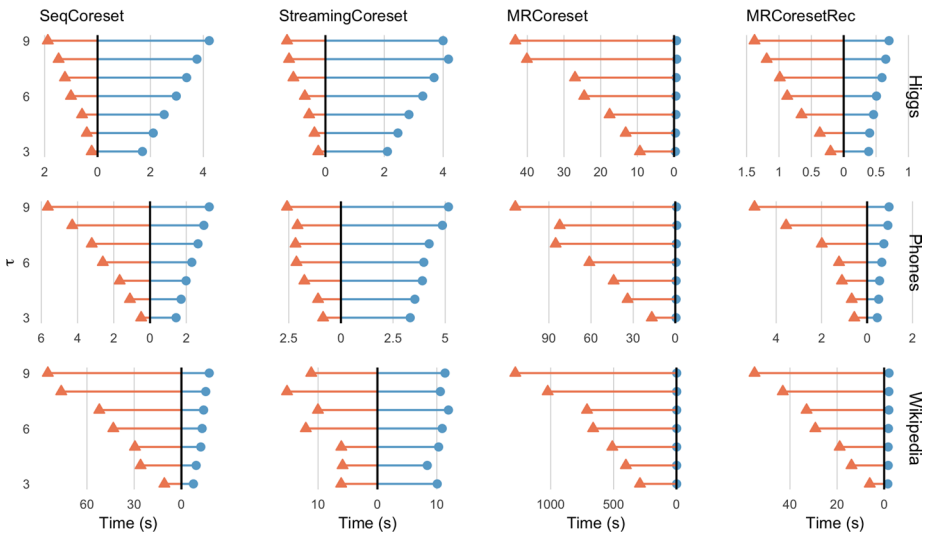

Fig. 5 Time spent building the coreset and time spent computing the solution. To the left the time for the solution ▲, to the right the time for the coreset ●. The x axis reports the running time (in seconds), the y axis reports the values of τ

the scalable coreset construction. We deem this solution MRCoresetRec, and report its performance as the last column of plots of Fig. 5. We have that the time to compute the solution is now comparable to SeqCoreset, but the coreset construction benefits from running on 8 processors. The approximation quality of this approach is in practice comparable to the one obtained by SeqCoreset: we observed a maximum difference between the final radius of the cluster obtained with MRCoresetRec and SeqCoreset of 25%, with the average difference being $\approx 1.5\%$.

For MRCoreset, we have that the computation of the solution dominates the running time by far. This is due to the size of the coreset built in parallel, which, for the same τ, is larger than the one computed by SeqCoreset, by a factor (roughly) equal to the number of workers. This problem can be ameliorated as follows. After building the coreset in parallel, SeqCoreset can be applied first to compute the "coreset of the coreset", (of size comparable to the coreset obtained by directly running SeqCoreset on the entire dataset), and then to obtain the final solution from this smaller coreset. While this approach might incur into a worsening of the approximation ratio, it dramatically speeds up the computation of the final solution, thus amplifying the benefits of the scalable coreset construction. We deem this solution MRCoresetRec, and report its performance as the last column of plots of Fig. 5. We have that, as expected, the time to compute the solution is now comparable to SeqCoreset, but the coreset construction benefits from running on 8 processors. The approximation quality of this approach is in practice comparable to the one obtained by SeqCoreset: we observed a maximum difference between the final radius of the cluster obtained with MRCoresetRec and SeqCoreset of about 25%, but the average difference is much more contained (approximately 1.5%).

Concluding remarks

In this paper, we presented a coreset-based strategy for computing approximate solutions to the RMC problem under general matroids, and provided efficient sequential, distributed (MapReduce) and streaming implementations of the strategy. The theoretical analysis shows that our strategy attains approximation guarantees comparable to the one of the best sequential strategies, with a dramatic improvement in efficiency for datasets of low doubling dimension. A rich suite of experiments on large inputs demonstrates the practical viability of our approach.

It is not difficult to show that the techniques employed for the RMC problem can also be used to extend the algorithms presented in [18, 24] for diversity maximization under partition and transversal matroid constraints, to work for all possible matroids, also improving their space requirements. Moreover, some preliminary theoretical results (see manuscript [50]) indicate that similar coreset-based approaches can be profitably employed to solve the *Robust Knapsack Center problem* [3], a variant of the robust unconstrained k-center problem where each input carries a weight, and the aggregate weight of the returned centers cannot exceed a certain budget.

Appendix

Proof of Theorem 2

A polynomial-time 3-approximation sequential algorithm for the RMCM problem can be obtained through a technical generalization of the approach of [4] to handle multiplicities. For ease of exposition, we restate the RMCM problem as an instantiation of the Robust $\mathcal{F}$-Supplier problem of [4] for the case where $\mathcal{F}$ is the family of independent sets of a matroid. Specifically, we define the *Robust Matroid Supplier problem with Multiplicities* (RMSM), whose generic instance is a tuple $\mathcal{I} = (F \cup C, d, m, \mathcal{F}, \mu)$, where $(F \cup C, d)$ is a metric space, m is an integer parameter, $(F, \mathcal{F})$ is a matroid, and $\mu : C \to \mathbb{N}$ is a function that associates a multiplicity to each point of C. The objective is to find $S \in \mathcal{F}$ and $T \subseteq C$ such that $\sum_{u \in T} \mu(u) \geq m$ and $\max_{u \in T} d(u, S)$ is minimized. An instance $(M = (V, I), z, \mathbf{m})$ of the RMCM problem can be regarded as instance of RMSM where: $V = F = C$, $I = \mathcal{F}$, $m = |V| - z$, and $\mu = \mathbf{m}$.

Following the lines of [4], a good approximate solution to the RMSM problem can be obtained by resorting to the auxiliary *Matroid-maximization under Partition Constraint* (Matroid-PCM) problem (which corresponds to the $\mathcal{F}$-PCM problem of [4], when $\mathcal{F}$ is the family of independent sets of a matroid). An instance of Matroid-PCM is a tuple $\mathcal{I} = (F, \mathcal{F}, \mathcal{P}, val)$, where $(F, \mathcal{F})$ is a matroid, $\mathcal{P}$ is a sub-partition of F, and $val : F \to \{0, 1, 2, \dots\}$ is integer valued function consistent with $\mathcal{P}$, in the sense that for each $A \in \mathcal{P}$ and for each pair $f_1, f_2 \in A$, $val(f_1) = val(f_2)$. For a set $S \in \mathcal{P}$, we let $val(S) = \sum_{f \in S} val(f)$. The objective of $\mathcal{F}$-PCM problem is to compute

$$\max\{val(S) : (S \in \mathcal{F}) \wedge (\forall A \in \mathcal{P} : |S \cap A| \leq 1)\}.$$

We remark that solving an instance of Matroid-PCM problem is equivalent to solving the weighted matroid intersection problem, where we need to find an intersection between the matroid $(F, \mathcal{F})$ and the partition matroid defined by $\mathcal{P}$. As a consequence this problem can be solved in polynomial time [51].

In order to describe the algorithm for RMSM, we make algorithm $\mathcal{F}$-PCM INSTANCE CONSTRUCTION (Algorithm 1 in [4, Section 3.1]) conscious to multiplicities by substituting Line 10 with the following line:

$$val(f) \leftarrow \sum_{u \in Chld(v)} \mu(u) \quad \forall f \in B_F(v, 1).$$

Similarly, we modify the definition of the polytope $\mathcal{P}^{\mathcal{I}}_{\text{cov}}$ in [4, section "Extraction of the solution from the coreset"] by substituting the constraint ($\mathcal{P}^{\mathcal{I}}_{\text{cov}}$.1) in the definition of the polytope with

$$\sum_{v \in C} \mu(v) cov(v) \geq m.$$

We sketch here the algorithm, which is the same as the one given in the proof of [4, Theorem 1], except for the changes to $\mathcal{F}$-PCM INSTANCE CONSTRUCTION and to polytope $\mathcal{P}^{\mathcal{I}}_{\text{cov}}$ described above. The algorithm starts by making a guess $\widehat{\text{opt}}$ on the optimal value of the solution. Then it considers the polytope $\mathcal{P}^{\mathcal{I}}_{\text{cov}}$, which is non-empty if the guess $\widehat{\text{opt}}$ is at least the cost of the optimal solution. It is possible to check whether $\mathcal{P}^{\mathcal{I}}_{\text{cov}}$ is empty using the ellipsoid algorithm [51]. When the ellipsoid algorithm asks if a given point is in the polytope, we run algorithm $\mathcal{F}$-PCM INSTANCE CONSTRUCTION to construct a Matroid-PCM instance that we solve using the weighted matroid intersection algorithm. If the algorithm finds a point inside $\mathcal{P}^{\mathcal{I}}_{\text{cov}}$, then we obtain a 3-approximate solution, otherwise if $\mathcal{P}^{\mathcal{I}}_{\text{cov}}$ is empty the algorithm proceeds with a new guess $\widehat{\text{opt}}$.

We omit the proof of the correctness of the algorithm since it is a straightforward adaptation of the proof in [4].

Abbreviations

RMC	Robust Matroid Center
RMCM	Robust Matroid Center with multiplicities
MPC	Massively Parallel Computation model

Acknowledgements
Not applicable.

Author contributions
AP, GP and FS developed the algorithmic ideas and the theoretical analysis. MC implemented the algorithms and carried out the experimental evaluation. All authors contributed to the writing of the manuscript. All authors read and approved the final manuscript.

Funding
This work was supported, in part, by MUR, the Italian Ministry of University and Research, under Projects PRIN 20174LF3T8 (AHeAD: Efficient Algorithms for HArnessing Networked Data), and PNRR CN00000013 (National Centre for HPC, Big Data and Quantum Computing), and by the University of Padova under Project SID 2020 (RATED-X: Resource-Allocation TradEoffs for Dynamic and eXtreme data).

Availability of data and materials
All our implementations are available as open source code which can be accessed at https://github.com/Cecca/macaco. The `Phones` and `Higgs` datasets are publicly available from the UCI Machine Learning Repository https://archive.ics.uci.edu/ml/index.php. The Wikipedia dataset can be downloaded from [46]. All the preprocessing required to prepare the datasets for usage with our implementation is implemented in the script `experiments/datasets.py` available in the source code repository.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 25 January 2022 Accepted: 16 March 2023
Published online: 17 April 2023

References

1. Gonzalez TF. Clustering to minimize the maximum Intercluster distance. Theor Comput Sci. 1985;38:293–306. https://doi.org/10.1016/0304-3975(85)90224-5.
2. Charikar M, Khuller S, Mount DM, Narasimhan G. Algorithms for facility location problems with outliers. In: Proc. of the 12th Annual Symposium on Discrete Algorithms, (SODA). ACM/SIAM; 2001. p. 642–651. http://dl.acm.org/citation.cfm?id=365411.365555.
3. Chen DZ, Li J, Liang H, Wang H. Matroid and Knapsack center problems. Algorithmica. 2016;75(1):27–52.
4. Chakrabarty D, Negahbani M. Generalized center problems with outliers. ACM Trans Algorithms. 2019;15(3):41:1-41:14. https://doi.org/10.1145/3338513.
5. Harris DG, Pensyl TW, Srinivasan A, Trinh K. A lottery model for center-type problems with outliers. ACM Trans Algorithms. 2019;15(3):36:1-36:25. https://doi.org/10.1145/3311953.
6. Krishnaswamy R, Kumar A, Nagarajan V, Sabharwal Y, Saha B. The Matroid Median Problem. In: Proc. of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA. SIAM; 2011. p. 1117–1130. https://doi.org/10.1137/1.9781611973082.84.
7. Hajiaghayi MT, Khandekar R, Kortsarz G. Local search algorithms for the red-blue median problem. Algorithmica. 2012;63(4):795–814.
8. Kleindessner M, Awasthi P, Morgenstern J. Fair k-Center Clustering for Data Summarization. In: Proc. of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA. vol. 97 of Proc. of Machine Learning Research. PMLR; 2019. p. 3448–3457. http://proceedings.mlr.press/v97/kleindessner19a.html.
9. Chiplunkar A, Kale SS, Ramamoorthy SN. How to Solve Fair k-Center in Massive Data Models. In: Proc. of the 37th International Conference on Machine Learning, ICML 2020, 13—18 July 2020, Virtual Event. vol. 119 of Proc. of Machine Learning Research. PMLR; 2020. p. 1877–1886. http://proceedings.mlr.press/v119/chiplunkar20a.html.
10. Leskovec J, Rajaraman A, Ullman JD. Mining of massive datasets. 2nd ed. Cambridge: Cambridge University Press; 2014.
11. Badoiu M, Har-Peled S, Indyk P. Approximate clustering via core-sets. In: Proc. on 34th Annual ACM Symposium on Theory of Computing, STOC. ACM; 2002. p. 250–257. https://doi.org/10.1145/509907.509947.
12. Beame P, Koutris P, Suciu D. Communication steps for parallel query processing. In: Proc. of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS. ACM; 2013. p. 273–284. https://doi.org/10.1145/2463664.2465224.
13. Awasthi P, Balcan MF. Center based clustering: a foundational perspective. In: Handbook of cluster analysis. CRC Press; 2015.
14. Kale S. Small Space Stream Summary for Matroid Center. In: Proceedings of the Workshop on Approximation, Randomization, and Combinatorial Optimization, Algorithms and Techniques, APPROX/RANDOM. vol. 145 of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik; 2019. p. 20:1–20:22. https://doi.org/10.4230/LIPIcs.APPROX-RANDOM.2019.20.
15. McCutchen RM, Khuller S. Streaming Algorithms for k-Center Clustering with Outliers and with Anonymity. In: Proc. of the 11th International Workshot on Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques, APPROX-RANDOM. vol. 5171 of Lecture Notes in Computer Science. Springer; 2008. p. 165–178. https://doi.org/10.1007/978-3-540-85363-3_14.
16. Malkomes G, Kusner MJ, Chen W, Weinberger KQ, Moseley B. Fast Distributed k-Center Clustering with Outliers on Massive Data. In: Proc. of the Annual Conference on Neural Information Processing Systems, NIPS; 2015. p. 1063–1071. https://proceedings.neurips.cc/paper/2015/hash/8fecb20817b3847419bb3de39a609afe-Abstract.html.
17. Ceccarello M, Pietracaprina A, Pucci G, Upfal E. MapReduce and streaming algorithms for diversity maximization in metric spaces of bounded doubling dimension. Proc VLDB Endow. 2017;10(5):469–80.
18. Ceccarello M, Pietracaprina A, Pucci G. Solving k-center clustering (with outliers) in MapReduce and streaming, almost as accurately as sequentially. Proc VLDB Endow. 2019;12(7):766–78.
19. Ding H, Yu H, Wang Z. Greedy Strategy Works for k-Center Clustering with Outliers and Coreset Construction. In: Proc. of the 27th Annual European Symposium on Algorithms, ESA. vol. 144 of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik; 2019. p. 40:1–40:16. https://doi.org/10.4230/LIPIcs.ESA.2019.40.
20. de Berg M, Monemizadeh M, Zhong Y. k-Center Clustering with Outliers in the Sliding-Window Model. In: Proc. of the 29th Annual European Symposium on Algorithms, ESA. vol. 204 of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik; 2021. p. 13:1–13:13. https://doi.org/10.4230/LIPIcs.ESA.2021.13.
21. Ceccarello M, Pietracaprina A, Pucci G, Upfal E. A practical parallel algorithm for diameter approximation of massive weighted graphs. In: Proc. of the International Parallel and Distributed Processing Symposium, IPDPS. IEEE Computer Society; 2016. p. 12–21. https://doi.org/10.1109/IPDPS.2016.61.
22. Ceccarello M, Fantozzi C, Pietracaprina A, Pucci G, Vandin F. Clustering uncertain graphs. Proc VLDB Endow. 2017;11(4):472–84.

23. Ceccarello M, Pietracaprina A, Pucci G. Fast coreset-based diversity maximization under matroid constraints. In: Proc. of the 11th ACM International Conference on Web Search and Data Mining, WSDM. ACM; 2018. p. 81–89. https://doi.org/10.1145/3159652.3159719.
24. Ceccarello M, Pietracaprina A, Pucci G. A general coreset-based approach to diversity maximization under matroid constraints. ACM Trans Knowl Discov Data. 2020;14(5):60:1-60:27. https://doi.org/10.1145/3402448.
25. Gupta A, Krauthgamer R, Lee JR. Bounded geometries, fractals, and low-distortion embeddings. In: Proc. of the 44th Symposium on Foundations of Computer Science, FOCS. IEEE Computer Society; 2003. p. 534–543. https://doi.org/10.1109/SFCS.2003.1238226.
26. Vapnik VN, Chervonenkis AY. In: Vovk V, Papadopoulos H, Gammerman A, editors. On the uniform convergence of relative frequencies of events to their probabilities. Cham: Springer International Publishing; 2015. p. 11–30. https://doi.org/10.1007/978-3-319-21852-6_3.
27. Aumüller M, Ceccarello M. The Role of Local Intrinsic Dimensionality in Benchmarking Nearest Neighbor Search. In: Proc. of the 12th International Conference on Similarity Search and Applications, SISAP. vol. 11807 of Lecture Notes in Computer Science. Springer; 2019. p. 113–127. https://doi.org/10.1007/978-3-030-32047-8_11.
28. Ahle TD, Aumüller M, Pagh R. Parameter-free Locality Sensitive Hashing for Spherical Range Reporting. In: Proc. of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA. SIAM; 2017. p. 239–256. https://doi.org/10.1137/1.9781611974782.16.
29. He J, Kumar S, Chang S. On the difficulty of nearest neighbor search. In: Proc. of the 29th International Conference on Machine Learning, ICML. icml.cc / Omnipress; 2012. http://icml.cc/2012/papers/580.pdf.
30. Mazzetto A, Pietracaprina A, Pucci G. Accurate MapReduce algorithms for k-Median and k-Means in general metric spaces. In: Proc. of the 30th International Symposium on Algorithms and Computation, ISAAC. vol. 149 of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik; 2019. p. 34:1–34:16. https://doi.org/10.4230/LIPIcs.ISAAC.2019.34.
31. Gottlieb L, Krauthgamer R. Proximity algorithms for nearly doubling spaces. SIAM J Discret Math. 2013;27(4):1759–69.
32. Oxley JG. Matroid Theory. Oxford graduate texts in mathematics. Oxford: Oxford University Press; 2006.
33. Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters. In: Proc. OSDI; 2004. p. 137–150.
34. Pietracaprina A, Pucci G, Riondato M, Silvestri F, Upfal E. Space-round tradeoffs for MapReduce computations. In: Proc. of the International Conference on Supercomputing, ICS. ACM; 2012. p. 235–244. https://doi.org/10.1145/2304576.2304607.
35. Sreedhar C, Kasiviswanath N, Reddy PC. Clustering large datasets using K-means modified inter and intra clustering (KM-I2C) in Hadoop. J Big Data. 2017;4:27. https://doi.org/10.1186/s40537-017-0087-2.
36. Bakhthemmat A, Izadi M. Decreasing the execution time of reducers by revising clustering based on the futuristic greedy approach. J Big Data. 2020;7(1):6.
37. Murray DG, McSherry F, Isard M, Isaacs R, Barham P, Abadi M. Incremental, iterative data processing with timely dataflow. Commun ACM. 2016;59(10):75–83.
38. Henzinger MR, Raghavan P, Rajagopalan S. Computing on data streams. In: Proc. DIMACS Workshop on External Memory Algorithms; 1998. p. 107–118.
39. Abbassi Z, Mirrokni VS, Thakur M. Diversity maximization under matroid constraints. In: Proc. of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD. ACM; 2013. p. 32–40. https://doi.org/10.1145/2487575.2487636.
40. Charikar M, Chekuri C, Feder T, Motwani R. Incremental clustering and dynamic information retrieval. SIAM J Comput. 2004;33(6):1417–40.
41. Hochbaum DS, Shmoys DB. A Best possible heuristic for the *k*-center problem. Math Oper Res. 1985;10(2):180–4.
42. Indyk P, Mahabadi S, Mahdian M, Mirrokni VS. Composable core-sets for diversity and coverage maximization. In: Proc. of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS. ACM; 2014. p. 100–108. https://doi.org/10.1145/2594538.2594560.
43. Aumüller M, Ceccarello M. Running experiments with confidence and sanity. In: Proc. of the 13th International Conference on Similarity Search and Applications, SISAP. vol. 12440 of Lecture Notes in Computer Science. Springer; 2020. p. 387–395. https://doi.org/10.1007/978-3-030-60936-8_31.
44. Higgs Dataset; https://archive.ics.uci.edu/ml/datasets/HIGGS. Accessed 20 Jan 2021.
45. Phone Activity Recognition Dataset; https://archive.ics.uci.edu/ml/datasets/Heterogeneity+Activity+Recognition. . Accessed 20 Jan 2021.
46. Wikipedia XML dump; https://dumps.wikimedia.org/enwiki/20210120/enwiki-20210120-pages-articles-multistream.xml.bz2. Accessed 20 Jan 2021.
47. Pennington J, Socher R, Manning CD. Glove: global vectors for word representation. In: Proc. of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP. ACL; 2014. p. 1532–1543. https://doi.org/10.3115/v1/d14-1162.
48. Blei DM, Ng AY, Jordan MI. Latent Dirichlet allocation. J Mach Learn Res. 2003;3:993–1022.
49. McSherry F, Isard M, Murray DG. Scalability! But at what COST? In: Proc. of the 15th Workshop on Hot Topics in Operating Systems, HotOS. USENIX Association; 2015. https://www.usenix.org/conference/hotos15/workshop-program/presentation/mcsherry.
50. Pietracaprina A, Pucci G, Soldà F. Coreset-based Strategies for Robust Center-type Problems. CoRR. 2020; arXiv:abs/2002.07463.
51. Schrijver A. Combinatorial optimization polyhedra and efficiency. Berlin: Springer; 2003.

Publisher's Note