

An Efficient Massively Parallel Constant-Factor Approximation Algorithm for the k -Means Problem

Vincent Cohen-Addad
Google Research
cohenaddad@google.com

Fabian Kuhn
University of Freiburg
kuhn@cs.uni-freiburg.de

Zahra Parsaeian
University of Freiburg
zahra.parsaeian@cs.uni-freiburg.de

Abstract

In this paper, we present an efficient massively parallel approximation algorithm for the k -means problem. Specifically, we provide an MPC algorithm that computes a constant-factor approximation to an arbitrary k -means instance in $O(\log \log n \cdot \log \log \log n)$ rounds. The algorithm uses $O(n^\sigma)$ bits of memory per machine, where $\sigma > 0$ is a constant that can be made arbitrarily small. The global memory usage is $O(n^{1+\varepsilon})$ bits for an arbitrarily small constant $\varepsilon > 0$, and is thus only slightly superlinear. Recently, Czumaj, Gao, Jiang, Krauthgamer, and Veselý showed that a constant-factor bicriteria approximation can be computed in $O(1)$ rounds in the MPC model. However, our algorithm is the first constant-factor approximation for the general k -means problem that runs in $o(\log n)$ rounds in the MPC model.

Our approach builds upon the foundational framework of Jain and Vazirani. The core component of our algorithm is a constant-factor approximation for the related facility location problem. While such an approximation was already achieved in constant time in the work of Czumaj et al. mentioned above, our version additionally satisfies the so-called Lagrangian Multiplier Preserving (LMP) property. This property enables the transformation of a facility location approximation into a comparably good k -means approximation.

1 Introduction

The classic k -means objective is widely used to model clustering problems in data mining and machine learning applications. Given a set of points in a metric space the k -means problem asks to identify k points from the metric, called *centers*, so as to minimize the sum of the squared distances from the input points to their closest center. Since its introduction in the late 50s by Lloyd [Llo82] and Max [Max60] the k -means problem has received a tremendous amount of attention in a variety of communities to, e.g., model compression problems or uncover underlying structure in datasets.

In the overwhelming majority of these applications, the data lies in Euclidean space. In this context, the k -means problem has a very natural clustering formulation: The optimum centers are the means of the clusters they represent and the goal is to minimize some notion of *dispersion* within clusters. For this reason, a long line of work on approximating k -means on massive datasets has emerged. Since the early 2000s, people have studied the k -means problem in streaming [SYZ18, COP03, AJM09], dynamic [Mey01, CGKR21], and distributed/parallel settings [CMZ22, CGJ+24, CEM+22, CLN+21, BT10, BGT12, EIM11, BLK18, BEL13].

Arguably, the parallel and distributed setting remains the model for which upper and lower bounds for the k -means problem remain frustratingly loose. Hence, the last few years have witnessed several works aiming at filling this gap in our understanding of the complexity of the k -means problem in this context. Focusing on the Massively Parallel Computation (MPC) model [KSV10, IKL+23], people have studied the following question.

How well can the k -means problem be approximated in the MPC model?

The question is particularly interesting if we ask for algorithms with sublogarithmic time complexities and if we ask for fully scalable algorithms for which the memory per parallel machine can be made $O(n^\sigma)$ for an arbitrarily small constant $\sigma > 0$. If k is sufficiently small, then some progress was made in [EIM11]. The paper shows that if the memory per machine is at least $O(k^2 \cdot n^\varepsilon)$ for some constant $\varepsilon > 0$, the related k -median and k -center problems can be approximated within a constant factor in $O(1)$ MPC rounds.¹ The authors of [EIM11] conjecture that using similar techniques, a corresponding result can also be proved for the k -means problem. Even if we assume this to be true, the general problem where k can be large remains open. Unfortunately, modern data mining and machine learning applications focus on massive datasets and high-dimensional inputs, see e.g. [BW18, CGJ+24, CMZ22, CLN+21]. Additionally, widely-used privacy-preserving techniques such as s -anonymity often require to find a clustering of the input users into small size clusters (of size $s < 100$), which translates in a desired number of clusters k larger than $n/100$ [BKBL07].

Therefore recent work has considered the scenario where k may much be larger than the memory of each machine and also where the dimension d of the Euclidean space is $\Omega(\log n)$, where n is the number of input points. First, Cohen-Addad, Lattanzi, Norouzi-Fard, Sohler, and Svensson [CLN+21] showed how to obtain an $O(\log \Delta \cdot \log n)$ -approximate solution for the k -median problem in $O(1)$ parallel rounds in the MPC model for any value of k and d , where Δ is the ratio of the maximum distance to the minimum distance in the input. However, the method proposed in their paper heavily relies on quad-tree embeddings and seems tailored to the k -median objective. This was later followed by the work of Cohen-Addad, Mirrokni, and Zhong [CMZ22] who showed how to get a $(1 + \varepsilon)$ -approximation for so-called *perturbation resilient* k -means inputs, a notion of

¹The k -median problems asks to minimize the sum of the distances as opposed to the sum of the squared distances and the k -center problem asks to minimize the maximum distance. The k -median and k -center problems tend to be easier to tackle as not all techniques that apply to k -median or k -center also work for the k -means problem.

beyond worst-case inputs, in $o(\log n)$ parallel rounds. Other works that studied massively parallel algorithms for the k -means problem include [BW18, CEM⁺22].

A stronger result was later obtained by Czumaj, Gao, Jiang, Krauthgamer, and Veselý [CGJ⁺24] who obtained an $O(1)$ -approximation in $O(1)$ parallel rounds for the related facility location problem that can then be used to obtain a bicriteria $O_\epsilon(1)$ -approximate solution for k -means using $(1 + \epsilon)k$ centers also in a constant number of MPC rounds. Recently, there has also been some progress for the k -center problem. In [CGGJ25], the authors give fully scalable $O(1)$ -round MPC algorithms to obtain a $(2 + \epsilon)$ -approximation and a bicriteria $(1 + \epsilon)$ -approximation.

In the present paper, we make significant progress toward answering the question about the complexity of the general k -means problem in the MPC model. We show the following result.

Main Theorem. For any constants $\sigma, \epsilon > 0$ and any $k, d \geq 1$, an $O(1)$ -approximate solution to the k -means problem in \mathbb{R}^d can be computed in the MPC model in $O(\log \log n \cdot \log \log \log n)$ rounds with $O(n^\sigma)$ bits of memory per machine and with $O(n^{1+\epsilon})$ bits of global memory.

Note that in all of the above constant factor approximations and also in the present work, the total global memory is $n^{1+\epsilon}$ for some fixed constant $\epsilon > 0$, i.e., the total global memory is required to be superlinear by a small polynomial factor.

Our main technical contribution is a fully scalable algorithm for the facility location problem, which is closely related to the k -means problem. Our facility location algorithm is based on the classic primal-dual framework of Jain and Vazirani [JV01]. Since our facility location satisfies a property known as *Lagrangian Multiplier Preserving* (LMP), we can (almost) directly use the framework from [JV01] to apply our facility location algorithm to solve the k -means problem and prove our main theorem.

1.1 Further Related Work

We briefly review additional related literature on two techniques that turn out to be important for our work: locality-sensitive hashing (LSH) and ruling set algorithms in parallel and distributed models.

Locality-Sensitive Hashing. Locality-Sensitive Hashing (LSH) is a key technique for efficiently handling approximate nearest neighbor queries in high-dimensional spaces, which is central to many clustering algorithms in parallel and distributed settings. The foundational work by Andoni and Indyk [AI06] introduced near-optimal LSH-based algorithms for high-dimensional nearest neighbor search, which have since been widely adopted in scalable clustering approaches. Additionally, Indyk [Ind04] provided a broader survey of data structures for nearest neighbor search in general metric spaces. The construction of [AI06] was extended to the MPC model in [CEM⁺22]. In general, LSH and closely related techniques have been used in several recent works on clustering algorithms in space-restricted environments (e.g., [BW18, CEM⁺22, CGJ⁺24, CJK⁺22]).

Distributed and Parallel Ruling Set Algorithms. An (a, b) -ruling set of a graph (as introduced in [AGLP89]) is a set of nodes such that any two nodes in the set are at distance at least a , and every node not in the set is within distance b of some node in the set. Such sets are useful primitives in distributed computing, particularly in the context of symmetry breaking and locality-based computations. Ruling sets have therefore been studied extensively in the distributed setting (e.g., [Gha16, PPP⁺17, KP12, KMW18, BGKO23]).

To date, it remains unknown whether there exists a constant- or poly $\log \log n$ -round algorithm for computing an $(O(1), O(1))$ -ruling set in the distributed or sublinear-memory MPC setting.

The fastest known algorithm in the low-memory MPC model is the $O(\log \log n \cdot \log \log \log n)$ -round algorithm of [KPP20], which computes a $(2, O(\log \log \log n))$ -ruling set. A sublogarithmic-round algorithm for computing a maximal independent set (which is a $(2, 1)$ -ruling set) was given by Ghaffari and Uitto [GU19]. Faster algorithms are known in the setting where each machine has memory linear in the number of graph nodes. In this case, it has been shown that a $(2, 2)$ -ruling set can be computed in $O(1)$ MPC rounds. A randomized algorithm for this was presented in [CKPU23], followed by a deterministic algorithm by Giliberti and Parsaeian [GP24].

1.2 Organization of the Paper

The remainder of the paper is organized as follows. In Section 2, we introduce the necessary mathematical notation and we formally define the k -means problem. We also introduce the closely related facility location problem. Subsequently, in Section 3, we provide an informal overview over the most important challenges that we face and the main ideas that we use to overcome those challenges. In Section 4, we provide and analyze a high-level version of our k -means algorithm, which is tailored towards parallel implementation, but still independent of the concrete computational model we use. The main part of Section 4 is a constant-factor LMP-approximation of the facility location problem. In Section 5, we then show how the algorithm of Section 4 can be implemented efficiently in the sublinear memory MPC model. Finally, in Section 5.5, we combine everything to prove our main theorem.

2 Problem Definition and Preliminaries

2.1 Mathematical Definitions and Notation

When arguing about asymptotic complexities, we sometimes make use of the $\tilde{O}(\cdot)$ -notation. We use $\tilde{O}(x)$ to denote terms that are upper bounded by $x \cdot \text{poly} \log x$, i.e., $\tilde{O}(\cdot)$ hides factors that are polylogarithmic in the argument.

Our problems are defined for point sets in Euclidean space. For $x, y \in \mathbb{R}^d$, we use $\text{dist}(x, y) := \|x - y\|_2$ to denote the Euclidean distance between x and y . For $x \in \mathbb{R}$ and $Y \subset \mathbb{R}$, we define the distance between x and Y as $\text{dist}(x, Y) := \inf_{y \in Y} \text{dist}(x, y)$. Further, for $x \in \mathbb{R}^d$, $Y \subset \mathbb{R}^d$, and a radius $r \geq 0$, we use $B_Y(x, r) := \{y \in Y : \text{dist}(x, y) \leq r\}$ to denote the ball of radius r around x restricted to the points in Y .

For graphs, we need the concept of ruling sets [AGLP89]. Given an undirected graph $G = (V, E)$ and two integers $\alpha \geq 1$ and $\beta \geq \alpha - 1$, an (α, β) -ruling set $S \subseteq V$ of G is a set of nodes such that for any two nodes $u, v \in S$, $d_G(u, v) \geq \alpha$, and for any $u \notin S$, there exists $v \in S$ such that $d_G(u, v) \leq \beta$. The special case where $\alpha = 2$ asks for an independent set S , where every node of the graph has an independent set node within hop distance at most β . Furthermore, a set S is an $(\alpha, \alpha \cdot \beta)$ -ruling set if and only if S is a $(2, \beta)$ -ruling set of the graph $G^{\alpha-1}$. Here, for any $t \geq 1$, G^t is defined as the graph on node set V with an edge between any two nodes at distance at most t in G .

Finally, we use the following notation throughout the paper:

$$\forall x \in \mathbb{R} : [x]^+ := \max\{0, x\}.$$

2.2 The k -Means Problem

Let $d \geq 1$ be an integer. In the k -means problem in \mathbb{R}^d , the input consists of a set of n points $P \subset \mathbb{R}^d$ and a positive integer $k \leq n$. The goal is to identify a set of k centers $Z \subset \mathbb{R}^d$ that minimizes the total squared Euclidean distance from each point in P to its closest center in Z .

Formally, the objective is to compute a set $Z \subseteq P$ with $|Z| = k$ such that the following cost function is minimized:

$$\text{cost}(Z) := \sum_{p \in P} \text{dist}^2(p, Z).$$

Note that we do not restrict the dimension d as a function of n . However, at the cost of a $(1 \pm \varepsilon)$ -factor in the approximation ratio (for an arbitrarily small constant $\varepsilon > 0$), we can apply a data-oblivious random projection to reduce the dimension of the input point set P to $\mathbb{R}^{O(\log k)}$ while approximately preserving the k -means cost. Therefore, we can assume without loss of generality that $d = O(\log k) = O(\log n)$ [BBCA⁺19]. Further, by scaling and sacrificing another $(1 + \varepsilon)$ -factor in the approximation ratio, we can assume that the minimum distance between any two points is at least 1, and the maximum distance is bounded polynomially in n (Lemma 4.1 in [ANFSW20]). Throughout the paper, we therefore assume that

$$P \subset \mathbb{R}^d \text{ for } d = O(\log k) \quad \text{and} \quad \forall x, y \in P : x \neq y \Rightarrow \text{dist}(x, y) \in [1, n^{O(1)}]. \quad (1)$$

A standard approach to approximate the k -means problem is by reducing it to a variant of the facility location problem. As we also follow this approach, we next formally define the facility location problem and we sketch the classic approximation algorithm by Jain and Vazirani [JV01].

2.3 The Facility Location Problem

The facility location problem is defined as follows: Given a set of facilities \mathcal{F} and a set of clients \mathcal{C} , the objective is to select a subset $\mathcal{F}' \subseteq \mathcal{F}$ of facilities to open and to assign each client $c \in \mathcal{C}$ to an open facility $f \in \mathcal{F}'$ such that the total cost is minimized. In the variant of the problem we study, opening a facility $f \in \mathcal{F}$ incurs a fixed opening cost $\lambda > 0$, and connecting a client $c \in \mathcal{C}$ to a facility f incurs a connection cost $\text{cost}(c, f) \geq 0$. For convenience, we use $\text{cost}(c, f)$ and $\text{cost}(f, c)$ interchangeably (with $\text{cost}(c, f) = \text{cost}(f, c)$). In the context of this paper, we assume that the connection cost satisfies a relaxed version of the triangle inequality. More precisely, for any integer $\ell \geq 2$ and any sequence x_0, x_1, \dots, x_ℓ of clients and facilities that alternates between clients and facilities (i.e., either x_0, x_2, \dots are clients and x_1, x_3, \dots are facilities or vice versa), we have

$$\sum_{i=1}^{\ell} \text{cost}(x_{i-1}, x_i) \leq \ell \cdot \text{cost}(x_0, x_s). \quad (2)$$

We note that this relaxation of the triangle inequality holds if the cost $\text{cost}(c, f)$ refers to the squared distance between c and f in some underlying metric space.

Linear Programming Formulation. The facility location problem can be formulated as an integer linear program as follows:

$$\begin{aligned}
\min \quad & \sum_{f \in \mathcal{F}} \lambda \cdot y_f + \sum_{f \in \mathcal{F}} \sum_{c \in \mathcal{C}} \text{cost}(c, f) \cdot x_{cf} \\
\text{s.t.} \quad & \sum_{f \in \mathcal{F}} x_{cf} \geq 1, \quad \forall c \in \mathcal{C}, \\
& x_{cf} \leq y_f, \quad \forall f \in \mathcal{F}, \forall c \in \mathcal{C}, \\
& x_{cf} \in \{0, 1\}, \quad \forall f \in \mathcal{F}, \forall c \in \mathcal{C}, \\
& y_f \in \{0, 1\}, \quad \forall f \in \mathcal{F}.
\end{aligned} \tag{3}$$

Here, y_f indicates whether facility f is open, and x_{cf} represents whether client c is connected to facility f . The first constraint ensures that each client is connected to at least one facility and the second constraint ensures that this facility has to be open. Relaxing y_f and x_{cf} to be in $[0, 1]$ yields the LP relaxation.

The *dual problem* of the LP relaxation of (3) can be stated as follows.

$$\begin{aligned}
\max \quad & \sum_{c \in \mathcal{C}} \alpha_c \\
\text{s.t.} \quad & \alpha_c - \beta_{cf} \leq \text{cost}(c, f), \quad \forall f \in \mathcal{F}, \forall c \in \mathcal{C}, \\
& \sum_{c \in \mathcal{C}} \beta_{cf} \leq \lambda, \quad \forall f \in \mathcal{F}, \\
& \alpha_c \geq 0, \quad \forall c \in \mathcal{C}, \\
& \beta_{cf} \geq 0, \quad \forall f \in \mathcal{F}, \forall c \in \mathcal{C}.
\end{aligned} \tag{4}$$

In this dual LP, α_c represents the total price paid by a client c , and β_{cf} represents the contribution of client c towards opening facility f . If a client c is connected to a facility f , then $\alpha_c - \beta_{cf} = \text{cost}(c, f)$; otherwise, it is zero. A facility is opened when $\sum_{c \in \mathcal{C}} \beta_{cf} = \lambda$.

Facility Location Algorithm by Jain and Vazirani In [JV01], Jain and Vazirani presented a centralized primal-dual algorithm to compute a 3-approximation for facility location problem if the cost function satisfies the (strict) triangle inequality. The algorithm achieves a 9-approximation in our case with the relaxed triangle inequality (2). In the following, we briefly sketch their algorithm.

The input is modeled as a (complete) bipartite graph, where one set of vertices represents the clients \mathcal{C} and the other set represents the facilities \mathcal{F} . The algorithm initializes all dual variables α_c and β_{cf} to 0, and initially classifies all clients as *unconnected*.

The first phase of the algorithm initially defines all clients as being *active* and it proceeds by uniformly increasing the α_c values of all active clients c at a fixed rate. Whenever $\alpha_c = \text{cost}(c, f)$ for some edge $\{c, f\}$ (i.e., for some client c and some facility f), the edge $\{c, f\}$ is declared as *tight*. At this point, client c has fully paid its connection cost to facility f and begins contributing toward the facility's opening cost. As a result, from now on, the dual variable β_{cf} is increased at the same rate as α_c .

A facility f is considered *temporarily open* once the total contribution from its adjacent clients is equal to the opening cost λ , i.e., if $\sum_{c \in \mathcal{C}} \beta_{cf} = \lambda$. Moreover, all unconnected clients that have a tight edge to f are then immediately connected to f and become *inactive* (ties about where to connect a client are broken arbitrarily in case several facilities become temporarily open at the same time). Throughout the remainder of the first phase of the algorithm, whenever a client c obtains tight edge with a facility f that is already temporarily open, c is connected to f and c

becomes inactive (note that β_{cf} in this case remains 0). The first phase terminates when all clients are connected to a temporarily open facility, i.e., when all clients are inactive. Note that the above construction ensures that the variables α_c and β_{cf} form a feasible solution of the dual LP (4).

At the end of the first phase, a client may have contributed to opening multiple facilities. To ensure each client contributes to at most one facility, we define a conflict graph $H = (\mathcal{F}_T, E_H)$, where \mathcal{F}_T is the set of temporarily open facilities. There is an edge $\{f, f'\} \in E_H$ between two temporarily open facilities f and f' if and only if there exists a client c such that $\beta_{cf} > 0$ and $\beta_{cf'} > 0$, i.e., c contributes to opening both f and f' . The final set of open facilities \mathcal{F}' is chosen as a maximal independent set of H . In this way, each client contributes to at most one open facility. By using the relaxed triangle inequality (2), one can show that

$$\sum_{c \in \mathcal{C}} \text{cost}(c, \mathcal{F}') \leq 9 \cdot \left(\sum_{c \in \mathcal{C}} \alpha_c - |\mathcal{F}'| \cdot \lambda \right) \leq 9 \cdot (\text{OPT} - |\mathcal{F}'| \cdot \lambda), \quad (5)$$

where OPT denotes the objective value of an optimal solution to the given facility location problem. The above inequality implies that the objective value $\sum_{c \in \mathcal{C}} \text{cost}(c, \mathcal{F}') + |\mathcal{F}'| \cdot \lambda$ achieved by the algorithm is within a factor at most 9 of OPT . Moreover the solution has the additional property known as *Langrangian multiplier preserving (LMP)* [JV01, CAGLS23]. It was already shown by Jain and Vazirani [JV01] that a constant-factor LMP approximation algorithm for the facility location problem can be used to develop a constant approximation for the k -means problem.

2.4 The Massively Parallel Computation Model

The *massively parallel computation* (MPC) model was introduced in [KSV10]. It is an abstract model that captures essential aspects of coarse-grained parallelism in large-scale data processing systems. In an MPC algorithm for an input of size n , we have a number of machines, each with n^σ bits of local memory for some constant $\sigma > 0$. An algorithm is called *fully scalable* if the constant σ can be made arbitrarily small. The machines can communicate with each other in synchronous rounds. In each round, every machine may send and receive up to $O(S)$ bits of data and perform arbitrary local computation. The total number of machines and thus the global memory can also be bounded. Ideally, one usually aims for a global memory of $\tilde{O}(n)$ bits. Sometimes, this is not possible. Another popular assumption that we also make in our paper is that the global memory is restricted to $O(n^{1+\varepsilon})$ bits, where $\varepsilon > 0$ is a constant that can be chosen arbitrarily small, possibly at some cost regarding the guarantees of an algorithm.

3 Technical Overview

In the following, we discuss the main technical challenges in devising an efficient MPC algorithm for the k -means problem and sketch the approach we used to resolve these challenges. As discussed, we assume that the input to the k -means problem is given by n points $P \subset \mathbb{R}^d$ for $d = O(\log k)$, and that for any $x, y \in P$ with $x \neq y$, their Euclidean distance satisfies $1 \leq \text{dist}(x, y) \leq \text{poly}(n)$ (cf. Eq. (1)). We solve the given k -means instance by reducing it to several instances of the facility location problem with some fixed opening cost $\lambda \geq 1$ per facility. In this facility location instance, the facilities \mathcal{F} and clients \mathcal{C} are also points in \mathbb{R}^d (in fact, in our reduction, we will have $\mathcal{F} = P$ and $\mathcal{C} = P$). For any $x, y \in \mathcal{F} \cup \mathcal{C}$, we therefore also have a well-defined Euclidean distance $\text{dist}(x, y)$. It is convenient to define the cost function not only between a client and a facility, but between

any pair of clients and facilities. The cost between any two $x, y \in \mathcal{C} \cup \mathcal{F}$ is defined as

$$\text{cost}(x, y) := (\text{dist}(x, y))^2. \quad (6)$$

Note that by applying the triangle inequality for the Euclidean distances and the Cauchy-Schwarz inequality, this implies that the cost function satisfies the relaxed triangle inequality given by Equation (2).

Challenge 1: Sparse Representation of Pairwise Distances. Since we assume that the distances are in $[1, n^{O(1)}]$, we can (approximately) represent each point in P by $d = O(\log k) = O(\log n)$ coordinates of $O(\log n)$ bits each. The whole input can therefore be stored using $O(n \log^2 n) = \tilde{O}(n)$ bits. However, we need to be able to make efficient non-trivial queries. In particular, we need to be able to find (approximate) nearest neighbors and (approximately) aggregate over different neighborhoods. Note that we cannot simply store all the distances as a weighted graph, as this would require $\Omega(n^2)$ memory. To approximately store the distances in a structured way, we use *locality-sensitive hashing (LSH)*, a technique that has been used in different contexts, in particular to perform approximate nearest neighbor queries [CGJ⁺24, AI06, DIIM04]. The technique has also been used in a recent MPC algorithm that achieves a constant bicriteria k -means approximation in $O(1)$ time [CGJ⁺24]. In this paper, we make use of a construction from [AI06], which allows storing the n points as a weighted graph G with the following properties. The nodes of the graph are the n points. For an arbitrary constant $\varepsilon > 0$, the graph has at most $n^{1+\varepsilon}$ edges and

$$\forall x, y \in P : d_G^{(2)}(x, y) \leq \text{dist}(x, y) \leq O(\sqrt{1/\varepsilon}) \cdot d_G^{(2)}(x, y),$$

where $d_G^{(2)}(x, y)$ is the length of a shortest path in G with hop-length at most 2 between x and y . The graph can be computed in $O(1)$ time in the MPC model, and it can then be used to perform efficient approximate aggregation over local neighborhoods. The details appear in Sections 5.1 and 5.2.

Challenge 2: Parallel LMP Approximation of Facility Location. As our k -means algorithm is based on the Jain/Vazirani framework, the main step of the algorithm is to compute a constant-factor LMP-approximation of the facility location problem. Our main challenge therefore is to design a parallel variant of the primal-dual algorithm that is both efficient and satisfies the LMP property. We start by describing a natural (but too slow) parallel algorithm for an idealized setting, where we can make exact queries about the underlying distances.

In the primal-dual algorithm, we initially set the dual variables α_c for clients c to 0 and we gradually increase the α_c -values at a fixed rate until some facility becomes paid. We can initially set $\alpha_c = \alpha_0 > 0$ for some sufficiently small α_0 . In order to be fast in a parallel setting, one can then try to be more aggressive than in the sequential algorithm and increase α_c in $O(\log n)$ discrete steps, where in each step, the current value is multiplied by some constant (for simplicity, say we double α_c for each active client c in each step). If we freeze each α_c value as soon as c contributes to the opening cost of some facility f that becomes paid or overpaid, we obtain an approximately feasible dual solution. Note that a facility f is called paid w.r.t. to dual values α_c if $\sum_{c \in \mathcal{C}} [\alpha_c - \text{cost}(c, f)]^+ \geq \lambda$ and it is called overpaid if this sum is strictly larger than λ .

The approximately feasible dual solution produced in this way can indeed be used to obtain a constant-factor approximation for the facility location problem. Unfortunately, however, it does not yield a solution that satisfies the LMP property. Slightly simplified, we can guarantee the LMP property if we ensure that the final dual solution is feasible and that the facilities we open are exactly paid for—but not overpaid—by the adjacent clients. This, however, suggests that increasing

the α_c values in parallel in discrete steps will not work.

We can make the approach work by using the following observation about the sequential primal-dual algorithm. At the cost of losing a constant factor in the approximation quality, one can increase the α_c -variables of different active clients at different rates. One merely has to make sure that at all times, the α_c -variables of all the active clients are within a constant factor of each other. Consider some intermediate state in the algorithm, where the current solution is still dual feasible and for all active clients c , we have $\alpha_c = \alpha$ for some value $\alpha > 0$. We would now like to increase the dual variables of the active clients to 2α . If we did this, some subset S of the facilities would become paid or overpaid. Instead of directly increasing all the dual values of active clients to 2α , we proceed as follows. We say that two facilities $f, f' \in S$ are in conflict if there exists an active client c for which $2\alpha \geq \text{cost}(c, f)$ and $2\alpha \geq \text{cost}(c, f')$. This relation defines a conflict graph H on the potentially paid facilities S . We now select a subset $S_0 \subseteq S$ in such a way that any two $f, f' \in S_0$ are at distance at least 4 in H and for any $f' \in S \setminus S_0$, there (ideally) exists an $f \in S_0$ at distance $O(1)$ from f' in H . Note that such a set S_0 is known as a $(4, O(1))$ -ruling set of the H (cf. Section 2). We then build clusters of facilities and clients around each facility in S_0 and we will eventually open exactly one facility in each cluster. For each $f \in S_0$, let C_f be the set of active clients for which $2\alpha > \text{cost}(c, f)$. By gradually increasing only the α_c -values for $c \in C_f$, we make sure that some facility \bar{f} (either f or one of its neighboring facilities in the conflict graph) becomes paid before the α_c -values for $c \in C_f$ exceed 2α . We can open \bar{f} and assign active clients c for which $\text{cost}(c, \bar{f}) = O(\alpha)$ to \bar{f} . Because any two facilities in S_0 have distance at least 4 in the conflict graph, the client sets C_f and $C_{f'}$ for $f, f' \in S_0$ are disjoint and they also contribute to a disjoint set of facilities in S . The increase of the dual values in C_f for different $f \in S_0$ can therefore be done completely independently and therefore in a ‘local’ manner. Moreover, since for each $f \in S_0$, exactly one facility will be opened, one can show that even for proving the LMP property, it is sufficient to just open a facility \bar{f} that approximately minimizes $\sum_{c \in C_f} \text{cost}(c, \bar{f})$.

The sketched parallel facility location algorithm has two main problems. First, because our sparse representation of the points (and thus of facilities and clients) only allows us to make approximate queries about neighborhoods, we cannot exactly implement the algorithm in our setting. By adding sufficient slack in all steps, it is, however, possible to adapt the algorithm to the setting where all steps can only be carried out in an approximate manner. More importantly, even though we increase the dual values quite aggressively, the algorithm is still too slow. In the end, the ratio between the largest and the smallest dual value α_c can be polynomial in n . We therefore need $\Omega(\log n)$ doubling steps, which, of course, implies that the round complexity of the algorithm will also be at least $\Omega(\log n)$. However, we aim for an algorithm with a round complexity that is at most $\text{poly log log } n$ and thus almost exponentially smaller than what we can achieve with the above algorithm. We next discuss how we can use the same basic idea without going through the logarithmically many doubling steps of the dual variables.

Challenge 3: Fast Parallel Primal-Dual Algorithm. As even exponentially increasing the dual client values is far too slow, we need to find a way to fix the dual variables much more directly. Also note that a reasonable approximate solution to the dual LP might require different clients c to use up to $\Theta(\log n)$ distinct values for α_c . We therefore also cannot increase the α_c variables together in a synchronized way. Instead, we aim to define a value α_c^* for every client, where α_c^* is essentially chosen as the maximum value such that, if all clients set their dual variable to α_c^* , no facility f to which c contributes (i.e., for which $\text{cost}(c, f) > \alpha_c^*$) is overpaid. Note that α_c^* is a lower bound on the final value of α_c in the standard sequential primal-dual algorithm. Concretely, the values α_c^* are computed as follows.

First, every facility f computes a radius r_f such that

$$r_f := \max \left\{ r \in \mathbb{R} : \sum_{c \in \mathcal{C}} [r^2 - \text{cost}(c, f)]^+ \leq \lambda \right\}. \quad (7)$$

Note that r_f is chosen such that, if all clients c set $\alpha_c = r_f^2$, then the facility f is exactly paid for. We now set α_c^* such that $\alpha_c^* \leq r_f^2$ for every facility f for which $\alpha_c^* > \text{cost}(c, f)$:

$$\alpha_c^* := \min_{f \in \mathcal{F}} \max \{ r_f^2, \text{cost}(c, f) \}. \quad (8)$$

Those definitions directly imply that by setting all the dual variables of clients c to α_c^* (and setting $\beta_{cf} = \alpha_c^* - \text{cost}(c, f)$), we obtain a dual feasible solution. Interestingly, one can also show that by setting $\alpha_c = C_A \cdot \alpha_c^*$ for a sufficiently large constant $C_A \geq 1$, every client c has a facility f within distance $O(\sqrt{\alpha_c^*})$ such that f is at least fully paid for by the dual variables α_c . One could therefore try to adapt the approach of the above algorithm as follows: consider the set S of facilities that are fully paid for by the dual values $C_A \cdot \alpha_c^*$, define the conflict graph H between facilities in S w.r.t the dual values $C_A \cdot \alpha_c^*$, select a set of centers $S_0 \subseteq S$ as a $(4, O(1))$ -ruling set of H , and open one facility close to every node in S_0 . However, this does not work directly, particularly because choosing the wrong centers S_0 might result in missing the opening of some facilities f with small radius r_f , causing some clients to be connected to a facility that is too far away.

The problem, in particular, occurs if for a client c , there exists another client c' within distance $O(\sqrt{\alpha_c^*})$ such that $\alpha_{c'}^* \ll \alpha_c^*$. Instead of increasing the dual variable of c until some facility becomes tight, we should, in this case, simply connect c to the facility that we open for client c' . We resolve this in the following way. We define a set $\mathcal{C}' \subset \mathcal{C}$ of *problematic clients*. Essentially, a client is problematic if there exists another client c' within distance $O(\sqrt{\alpha_c^*})$ such that $\alpha_{c'}^* \ll \alpha_c^*$. For every client c , we define two dual values $\alpha_{c,0}$ and $\alpha_{c,1}$, where $\alpha_{c,0}$ is smaller than α_c^* by some constant factor. For problematic clients, we set $\alpha_{c,1} = \alpha_{c,0}$, and for all other clients, we set $\alpha_{c,1} = C_A \cdot \alpha_{c,0}$, where C_A is a sufficiently large constant. One can then show that, when choosing the constants appropriately, for all clients c , there exists a facility f' within distance $\alpha_{c,0}$ such that f' is paid w.r.t the dual values $\alpha_{c',1}$ (formally proven in [Lemma 4.2](#)). It is now also true that whenever two facilities f, f' are in conflict (i.e., if there exists a client c that contributes to both of them), then $r_f = \Theta(r_{f'})$ and $\alpha_{c,1} = O(r_f)$, and thus $\text{dist}(f, f') = O(r_f)$ (cf. [Lemmas 4.3 and 4.4](#)). This is now sufficient for our general approach described above to work.

One of the challenges in implementing the above algorithm in the MPC model is that we only have approximate access to the underlying Euclidean metric through our sparse LSH approximation. Therefore, for example, we cannot compute r_f for facilities f or α_c^* for clients c exactly. However, we can efficiently compute approximations that differ from the actual values by at most a constant factor. We can also only approximately determine whether a facility is paid for by some given dual client values. Consequently, we must compute a conflict graph H that includes all facilities that are approximately paid. Fortunately, the above algorithm is robust enough to work even when all quantities are computed only up to a constant-factor error.

Challenge 4: Computing a Ruling Set in the MPC Model. One of the most challenging tasks in implementing the sketched algorithm in the MPC model is the computation of a ruling set to determine cluster centers \mathcal{S}_0 among the (approximately) paid facilities. The problem can be broken down to a standard graph algorithms problem in the MPC model. Given an n -node and m -edge graph $G = (V, E)$, one needs a fully scalable algorithm to compute a (a, b) -ruling set for some constants $a > 1$ and $b \geq a - 1$. In the conflict graph H , we need to compute an

$(4, O(1))$ -ruling set. However, our algorithm will compute a sparse graph G such that G^2 serves as the conflict graph. We therefore have to compute an $(7, O(1))$ -ruling set on G . Unfortunately, there currently are no fully scalable MPC algorithm to compute such sets in time at most $\text{poly log log } n$. The fastest fully scalable MPC algorithm for computing a ruling set is an algorithm that computes a $(2, O(\log \log \log n))$ -ruling set in time $O(\log \log n \cdot \log \log \log n)$ [KPP20]. The algorithm uses $O(m + n^{1+\varepsilon})$ bits of global memory (for ε an arbitrarily small constant), which is fine for us. However, the algorithm does not directly compute an (a, b) -ruling set for any $a > 2$ and even more problematically, it only computes an (a, b) -ruling set for $b = O(\log \log \log n)$. When using such a ruling set in our algorithm, we would only achieve an approximation ratio that is exponential in $O(\log \log \log n)$ and thus $\text{poly log log } n$. We still use the ruling set algorithm of [KPP20], but we have to combine it with an additional idea.

Let us first discuss, how one can adapt any $(2, b)$ -ruling set algorithm to compute an $(a, (a-1)b)$ -ruling set on some given graph G . Note that a node set X is $(a, (a-1)b)$ -ruling set on G if and only if X is a $(2, b)$ -ruling set on G^{a-1} . We cannot directly apply an algorithm on G^{a-1} because G^{a-1} might be dense even if G is sparse and we therefore would need too much global memory. We can however again exploit the fact that we have $n^{1+\varepsilon}$ bits of global memory. By sampling each node with a sufficiently small probability, one can make sure that the subgraph of G^{a-1} induced by the sampled nodes has at most $n^{1+\varepsilon}$ edges. When computing a $(2, b)$ -ruling set on the sampled subgraph of G^{a-1} and remove all nodes that have a ruling set node within distance $(a-1)b$ in G , the maximum degree of the remaining subgraph of G^{a-1} drops by a factor $n^{\Omega(\varepsilon)}$. We can therefore compute an $(a, (a-1)b)$ -ruling set of G in $O(1/\varepsilon) = O(1)$ phases, where in each phase, we run a $(2, b)$ -ruling set algorithm on a sufficiently sparse graph that we can construct explicitly.

Let us now discuss how we cope with the fact that we do not know how to efficiently compute an (a, b) -ruling set for $b = O(1)$. In [BGKO23], it is shown that if the basic randomized Luby algorithm [ABI86, Lub86] for computing a maximal independent set is adapted to compute a $(2, 2)$ -ruling set, then in each step of the algorithm, a constant fraction of the nodes of a graph becomes covered. In $O(\log \log n)$ iterations of the algorithm, one can therefore guarantee that a $(1 - 1/\text{poly log } n)$ -fraction of the nodes has a ruling set node within distance 2. This idea can easily be adapted to the MPC model and also to computing (a, a) -ruling sets instead of $(2, 2)$ -ruling sets. By slightly adjusting the quality of the ruling sets and running the algorithm for several weight classes, one can also get a weighted version of this. Given a node-weighted graph, in $O(\log \log n)$ rounds, one can compute a set X of centers so that any two nodes in X are at distance at least a (for a given $a > 1$) and such that the total weight of nodes that are not within distance $O(1)$ of a node in X is at most $1/\text{poly log } n$ -fraction of the total node weight. If one uses the algorithm of [KPP20] to cover the remaining nodes within distance $O(\log \log \log n)$, we can use the computed set of centers to find a constant approximate facility location solution.

Challenge 5: From Facility Location to k -Means. Our reduction from the k -means problem to facility location follows the framework described by Jain and Vazirani for the k -median problem [JV01]. Given a k -median instance with point set P , one can define a facility location instance by setting both the facilities \mathcal{F} and the clients \mathcal{C} equal to P . If there exists an opening cost λ for which our LMP approximation algorithm opens exactly k facilities, we can directly use this solution for the k -means instance. Of course, this may not always be possible, so we aim to approximate this scenario. As a first step, we compute facility location solutions for two opening costs, λ_1 and λ_2 , such that $\lambda_2 \leq \lambda_1 \leq 2\lambda_2$, where the solution corresponding to λ_1 opens $k_1 \leq k$ facilities and the one corresponding to λ_2 opens $k_2 \geq k$ facilities.

Jain and Vazirani describe a randomized rounding procedure that interpolates between the two facility location solutions corresponding to λ_1 and λ_2 , such that the resulting solution opens exactly

k facilities. Provided that the facility location algorithm satisfies the LMP property, the resulting k -means solution achieves a constant-factor approximation with respect to the original instance. Using the LSH-based approximation of Euclidean distances, this randomized rounding procedure can be implemented in the MPC model in a relatively straightforward manner.

4 High-Level k -Means Algorithm

We are now ready to formally define and analyze our algorithm for computing a constant-factor approximation to the k -means problem. Instead of directly presenting an MPC algorithm, we first describe a high-level version that is (mostly) independent of implementation details. In [Section 5](#), we then show that the high-level algorithm presented in this section can be implemented efficiently and in a fully scalable way within the MPC model. The high-level algorithm consists of two parts. In [Section 4.1](#), we describe the main component: a constant-factor LMP approximation algorithm for the facility location problem. In [Sections 4.3](#) and [4.4](#), we show how to leverage this facility location algorithm to compute an approximate solution for a given k -means instance.

4.1 High-Level Facility Location Algorithm Description

As discussed at the beginning of [Section 3](#), we assume that we are given a set \mathcal{F} of facilities, a set \mathcal{C} of clients, and an opening cost $\lambda \geq 1$. We further assume that \mathcal{F} and \mathcal{C} are represented by points in \mathbb{R}^d for $d = O(\log k)$, and that for any two distinct points $x, y \in \mathcal{F} \cup \mathcal{C}$, we have $\text{dist}(x, y) \geq 1$. The connection cost between a client c and a facility f is defined as $\text{cost}(c, f) = \text{dist}^2(c, f)$. The details of our high-level facility location algorithm are provided in the pseudo-code of [Algorithm 1](#). In the following, we also discuss each step of the algorithm in further detail.

The algorithm relies on several constants that determine the accuracy with which each individual step can be efficiently executed. We specify all of these constants as a function of a sufficiently large constant $\Gamma \geq 1$, which, in turn, depends on the quality of our sparse LSH graph in approximating the distances in \mathbb{R}^d . We use the following definition to characterize when a facility is paid for or approximately paid for.

Definition 4.1 (Paid and Approximately Paid Facilities). *Consider a facility location instance with facilities \mathcal{F} and clients \mathcal{C} , and assume that each client $c \in \mathcal{C}$ is assigned a dual value $\alpha_c > 0$. We say that:*

- A facility f is paid w.r.t the dual values α_c if

$$\sum_{c \in \mathcal{C}} [\alpha_c - \text{cost}(c, f)]^+ \geq \lambda.$$

- A facility f is κ -approximately paid w.r.t the dual values α_c and a constant $\kappa > 1$ if it is paid w.r.t the scaled dual values $\kappa \cdot \alpha_c$.

We further use the following terminology to describe when a client contributes to paying for a facility. We say that a client c *contributes to a facility f* w.r.t the dual value α_c if $\alpha_c > \text{cost}(c, f)$. For $\kappa \geq 1$, we say that a client c *κ -approximately contributes to a facility f* w.r.t the dual value α_c if $\kappa \cdot \alpha_c > \text{cost}(c, f)$.

Step I: Facility Radius Assignment. As described in [Section 3](#), we define a radius r_f for each facility f such that f is exactly paid for if all clients set $\alpha_c = r_f^2$. Since in our MPC implementation

Algorithm 1 High-Level Facility Location Algorithm

- 1: **Step I: Assign Facility Radii.** Each facility $f \in \mathcal{F}$ computes a radius \hat{r}_f such that

$$\frac{r_f}{C_R} \leq \hat{r}_f \leq r_f \quad \text{for } r_f = \max \left\{ r \in \mathbb{R} : \sum_{c \in \mathcal{C}} [r^2 - \text{cost}(c, f)]^+ \leq \lambda \right\} \text{ and } C_R > 1. \quad (9)$$

- 2: **Step II: Assign Client Dual Values.** Each client $c \in \mathcal{C}$ computes a value $\alpha_{c,0}$ such that

$$\frac{\alpha_c^*}{C_D^+} \leq \alpha_{c,0} \leq \frac{\alpha_c^*}{C_D^-} \quad \text{for } \alpha_c^* = \min_{f \in \mathcal{F}} \max \{r_f^2, \text{cost}(c, f)\}, \quad C_D^+ > C_D^- \geq 2. \quad (10)$$

- 3: **Step III: Determine Problematic Clients.** Compute $\mathcal{C}' \subseteq \mathcal{C}$ such that for all $c \in \mathcal{C}$ and appropriate constants $\gamma_1 > 1$, $\gamma_2 > 1$, and $Q > 1$,

$$\exists c' \in B_{\mathcal{C}}(c, \gamma_1 \cdot \sqrt{\alpha_c^*}) \text{ such that } \alpha_{c',0} \leq \frac{\alpha_{c,0}}{Q} \implies c \in \mathcal{C}', \quad (11)$$

$$c \in \mathcal{C}' \implies \exists c' \in B_{\mathcal{C}}(c, \gamma_2 \cdot \sqrt{\alpha_c^*}) \text{ such that } \alpha_{c',0} \leq \frac{\alpha_{c,0}}{Q}. \quad (12)$$

- 4: Define the upper dual values as:

$$\forall c \in \mathcal{C}, \quad \alpha_{c,1} := \begin{cases} \alpha_{c,0}, & \text{if } c \in \mathcal{C}', \\ C_A \cdot \alpha_{c,0}, & \text{with } C_A > 1 \text{ otherwise.} \end{cases} \quad (13)$$

- 5: **Step IV: Approximately Paid Facilities.** Compute a set $\mathcal{S} \subseteq \mathcal{F}$ of facilities such that for the dual values $\alpha_{c,1}$ and $\kappa > 1$, a) \mathcal{S} contains all paid facilities of \mathcal{F} and b) all facilities in \mathcal{S} are κ -approximately paid.

- 6: **Step V: Dependency Graph.** The dependency graph $H = (\mathcal{S}, E_H)$ on \mathcal{S} is defined as

$$E_H := \left\{ \{f, f'\} \in \binom{\mathcal{S}}{2} : \exists c \in \mathcal{C} \text{ s.t. } \kappa \cdot \alpha_{c,1} > \max \{ \text{cost}(c, f), \text{cost}(c, f') \} \right\}.$$

- 7: Compute an undirected graph $H' = (\mathcal{S}, E_{H'})$ such that $E_H \subseteq E_{H'}$ and for every edge $\{f, f'\} \in E_{H'}$ we have $r_f/C_{H,1} \leq r_{f'} \leq C_{H,1} \cdot r_f$, $\text{dist}(f, f') \leq C_{H,2} \cdot \min \{r_f, r_{f'}\}$, and appropriate constants $C_{H,1}$ and $C_{H,2}$.

- 8: **Step VI: Compute Clustering.** Determine a set of cluster centers $\mathcal{S}_0 \subseteq \mathcal{S}$ such that

- Any two facilities in \mathcal{S}_0 are at distance at least 4 in H .
- For every client $c \in \mathcal{C}$, determine one facility $f_c \in \mathcal{S}$ for which $\max \{r_f^2, \text{cost}(c, f)\} = O(\alpha_{c,0})$.
- Let $\mathcal{C}^+ \subseteq \mathcal{C}$ be the clients for which f_c is within hop dist. $O(1)$ of a node $f_0 \in \mathcal{S}_0$ in H' .
- For all clients $c \in \mathcal{C} \setminus \mathcal{C}^+$, f_c is within distance $o(\log \log n)$ of a node in \mathcal{S}_0 in H' .
- Define $A := \sum_{c \in \mathcal{C}} \alpha_{c,0}$ and $A^+ := \sum_{c \in \mathcal{C}^+} \alpha_{c,0}$. We have $(A - A^+)/A \leq 1/\log n$.

- 9: Assign each facility to a nearest center in H' and each client $c \in \mathcal{C}$ to the cluster of f_c .

- 10: **Step VII: Opening Facilities.** In every cluster, open one facility that approximately (within a constant factor) minimizes the distance to all clients of the cluster.
-

the radii r_f cannot be computed exactly, the algorithm instead computes an approximate value \hat{r}_f for each facility f . The approximate radius \hat{r}_f is chosen to satisfy

$$\frac{r_f}{C_R} \leq \hat{r}_f \leq r_f,$$

for some constant $C_R > 1$.

Step II: Assign Client Dual Values. For each client $c \in \mathcal{C}$, we initialize an approximate dual value $\alpha_{c,0}$ that approximates the minimal value $\alpha_c^* := \min_{f \in \mathcal{F}} \max\{r_f^2, \text{cost}(c, f)\}$ required to connect c to some facility in \mathcal{F} (cf. the description in [Section 3](#)). The assigned dual value $\alpha_{c,0}$ is required to satisfy

$$\frac{\alpha_c^*}{C_D^+} \leq \alpha_{c,0} \leq \frac{\alpha_c^*}{C_D^-},$$

where C_D^+ and C_D^- are constants such that $C_D^+ > C_D^- > 1$. This range ensures that $\alpha_{c,0}$ remains within a constant factor of α_c^* . We will later show that initializing all dual values $\alpha_{c,0}$ in this way guarantees that no facility is fully paid, i.e.,

$$\forall f \in \mathcal{F}, \quad \sum_{c \in \mathcal{C}} [\alpha_{c,0} - \text{cost}(c, f)]^+ < \lambda.$$

Step III: Identification of Problematic Clients In this step, we identify clients in \mathcal{C} that may become *problematic* if their dual values are increased beyond their initial assignment. We define a subset $\mathcal{C}' \subseteq \mathcal{C}$ consisting of clients whose scaled dual values could lead to inconsistencies in the facility-opening process.

Informally, a client $c \in \mathcal{C}$ is classified as problematic if, upon increasing its dual value by a sufficiently large constant factor, it contributes to two facilities f and f' such that the radius of f' is significantly smaller than the radius of f . Formally, a client c must be classified as problematic if

$$\exists c' \in B_{\mathcal{C}} \left(c, \gamma_1 \cdot \sqrt{\alpha_c^*} \right) \text{ such that } \alpha_{c',0} \leq \frac{\alpha_{c,0}}{Q},$$

where γ_1 and Q are sufficiently large constants. Since determining this condition exactly is not efficiently possible, the algorithm is allowed to include some additional clients as problematic. In particular, a client c may be marked as problematic whenever

$$\exists c' \in B_{\mathcal{C}} \left(c, \gamma_2 \cdot \sqrt{\alpha_c^*} \right) \text{ such that } \alpha_{c',0} \leq \frac{\alpha_{c,0}}{Q},$$

where $\gamma_2 \gg \gamma_1$ is another sufficiently large constant. For each client $c \in \mathcal{C}$, we then define an upper dual value $\alpha_{c,1}$ that guarantees that for every client c , there is a facility f that is paid w.r.t. the dual values $\alpha_{c,1}$ and that is within distance $O(\sqrt{\alpha_{c,0}}) = O(\sqrt{\alpha_{c,1}})$ of c . To guarantee this, for all clients $c \in \mathcal{C}'$, i.e., for all clients that are classified as problematic, we set $\alpha_{c,1} := \alpha_{c,0}$ and for all other clients $c \in \mathcal{C} \setminus \mathcal{C}'$, we set $\alpha_{c,1} := C_A \cdot \alpha_{c,0}$, where C_A is a sufficiently large constant.

Step IV: Approximately Paid Facilities. In this step, the algorithm selects a set of facilities \mathcal{S} that are candidates to be opened because they are at least approximately paid for by the computed dual client values. Formally, \mathcal{S} must contain all facilities that are paid w.r.t the dual values $\alpha_{c,1}$, and it may include any facility that is κ -approximately paid w.r.t the dual values $\alpha_{c,1}$, for some sufficiently large constant $\kappa \geq 1$.

Step V: Dependency Graph. The *dependency graph* $H = (\mathcal{S}, E_H)$ between the paid and approximately paid facilities \mathcal{S} is defined as follows. There is an edge $\{f, f'\} \in E_H$ between two facilities $f, f' \in \mathcal{S}$ if and only if there exists a client c such that c κ -approximately contributes to both f and f' . Here, κ is the same constant as in Step IV. In this way, as long as we do not increase the dual values beyond $\kappa \cdot \alpha_{c,1}$, any facilities that receive contributions from the same client are neighbors in H .

In the algorithm, we cannot compute H exactly and therefore compute a supergraph $H' = (\mathcal{S}, E_{H'})$. The graph H' contains all edges of H , i.e., $E_H \subseteq E_{H'}$, and it may include additional edges. In our analysis, we show that if a client c κ -approximately contributes to two facilities f and f' , then it must hold that $r_f = \Theta(r_{f'})$ and $\alpha_{c,1} = O(r_f^2)$ (cf. [Lemmas 4.3](#) and [4.4](#)). In the approximation analysis, this is the only property we require for neighboring nodes in the extended dependency graph H' . We therefore allow the algorithm to include any edge $\{f, f'\}$ in $E_{H'}$ for which the radii satisfy $r_f/C_H \leq r_{f'} \leq C_H \cdot r_f$, and the distance condition $\text{dist}(f, f') \leq 2C_H^2 \cdot \max\{r_f, r_{f'}\}$ holds, where C_H is a constant defined as a fixed multiple of Γ .

Step VI: Compute Clustering. In this step, we utilize the extended dependency graph H' from Step V to define clusters of facilities. The clustering process begins with the selection of a subset of facilities, denoted by $\mathcal{S}_0 \subseteq \mathcal{S}$, which serve as cluster centers. The selection criterion ensures that any two facilities in \mathcal{S}_0 are at least four hops apart in the graph H' (and thus also in H). This separation condition guarantees that the clusters formed around these centers are well-separated. In particular, it implies that if, for each $f \in \mathcal{S}_0$, either f or one of its neighboring facilities f' in H' is opened, then—as long as the dual values are set to at most $\kappa \cdot \alpha_{c,1}$ —each client contributes to at most one open facility.

Ideally, we would like to select the centers \mathcal{S}_0 as a $(4, O(1))$ -ruling set of H' , i.e., in such a way that every facility in \mathcal{S} has a facility in \mathcal{S}_0 within constant distance in H' . Unfortunately, there is no sufficiently efficient MPC algorithm known that computes such a set \mathcal{S}_0 , and we therefore have to compute a set \mathcal{S}_0 that is sufficiently close to being a $(4, O(1))$ -ruling set. We assign every client c to some facility $f_c \in \mathcal{S}$ at distance at most $\text{cost}(c, f) = O(\alpha_{c,0})$ (such a facility exists by [Lemma 4.2](#)). We then assign a weight to each facility $f \in \mathcal{S}$, which is equal to the sum of the dual values $\alpha_{c,0}$ of all clients for which $f_c = f$. Given this, we ensure that a $(1 - 1/\log n)$ -fraction of the total weight of the facilities in \mathcal{S} lies within a constant hop distance in H' of a facility in \mathcal{S}_0 . We also ensure that all remaining facilities in \mathcal{S} are within distance $o(\log \log n)$ of \mathcal{S}_0 .

Given the set \mathcal{S}_0 , we now define clusters as follows: each facility is assigned to its nearest center in H' , and each client $c \in \mathcal{C}$ is assigned to the cluster of its selected facility f_c .

Step VII: Opening Facilities. Within each cluster, we now open one facility that approximately (within a constant factor) minimizes the sum of squared Euclidean distances to the clients in that cluster. This ensures that the selected facility serves as a cost-effective center for the clients in its assigned area, which turns out to be sufficient to guarantee a constant-factor approximation that satisfies the LMP property.

4.2 Analysis of the High-Level Facility Location Algorithm

In the following, we formally establish the correctness of [Algorithm 1](#). We begin by proving that the upper dual client values computed in Step III of [Algorithm 1](#) ensure that each client contributes toward at least one paid facility ([Lemma 4.2](#)). Next, in [Lemma 4.3](#), we show that if a client contributes to a facility, there exists a well-defined relationship between the client's dual value and the facility's radius. [Lemma 4.4](#) then establishes that if a client contributes to multiple facilities, those facilities have asymptotically equal radii. Following this, in [Lemma 4.5](#), we demonstrate

that the facilities that are actually opened are sparse enough that each client contributes toward the opening cost of at most one opened facility. Finally, [Theorem 4.7](#) analyzes the approximation quality of [Algorithm 1](#), proving that the algorithm achieves a constant-factor approximation and satisfies the LMP property.

[Algorithm 1](#) uses a number of constants, many of which are interdependent. Specifically, the pseudocode references the constants C_R , C_D^- , C_D^+ , γ_1 , γ_2 , Q , C_A , κ , and $C_{H,1}$, $C_{H,2}$. In the analysis, we additionally introduce constants η , ζ , and ρ . All of these constants are deterministically defined as functions of a single parameter $\Gamma \geq 5$, which is the approximation factor guaranteed by the graph construction in [Lemma 5.4](#). The precise definitions of these constants are provided in [\(14\)](#).

$$\begin{aligned} C_R &= 9\Gamma, & C_D^- &= 2\Gamma^2, & C_D^+ &= 8\Gamma^4, & \gamma_1 &= 4\Gamma^4, \\ \gamma_2 &= 9\Gamma^4, & Q &= 8\Gamma^4, & C_A &= 8\Gamma^8, & \kappa &= \Gamma^2, \\ \eta &= 8000\Gamma^{12}, & \zeta &= 1 + 16\sqrt{2}\Gamma^7, & \rho &= 128\Gamma^{12}, & C_{H,1} &= 4 \cdot C_R^2 \cdot \zeta^2, \\ C_{H,2} &= 12\sqrt{\kappa \cdot \rho} \cdot C_R^3 \cdot \zeta^2 \end{aligned} \tag{14}$$

If the constants are fixed as described above, the following analysis holds for any $\Gamma \geq 5$. The exact value of Γ depends on the implementation of the algorithm in the MPC model, which requires Γ to be sufficiently large. For the remainder of [Section 4.2](#), we assume that the constants are set according to Equation [\(14\)](#).

Lemma 4.2. *For every client $c \in \mathcal{C}$, there exists a facility $f \in \mathcal{F}$ that is paid w.r.t the dual values $\alpha_{c,1}$ and that satisfies*

$$\max \{r_f^2, \text{cost}(c, f)\} \leq \eta \cdot \alpha_{c,0}.$$

Proof. Recall that a facility f is considered paid w.r.t the dual values $\alpha_{c,1}$ if

$$\sum_{c \in \mathcal{C}} [\alpha_{c,1} - \text{cost}(c, f)]^+ \geq \lambda. \tag{15}$$

Suppose, for the sake of contradiction, that there exists a client $c \in \mathcal{C}$ such that no facility $f \in \mathcal{F}$ is both paid and satisfies [\(15\)](#). Without loss of generality, assume that c is a client with minimum $\alpha_{c,0}$ -value for which this is the case.

We distinguish two cases: 1) when c is a non-problematic client ($c \in \mathcal{C} \setminus \mathcal{C}'$), and 2) when c is a problematic client ($c \in \mathcal{C}'$).

Case 1 ($c \in \mathcal{C} \setminus \mathcal{C}'$): Let $f \in \mathcal{F}$ be a facility that minimizes $\max\{r_f^2, \text{cost}(c, f)\}$. By the definition of α_c^* (cf. [\(10\)](#)), we have

$$\alpha_c^* = \max\{r_f^2, \text{cost}(c, f)\}.$$

This in particular implies that $\max\{r_f^2, \text{cost}(c, f)\} \leq \alpha_c^*$. For each client $c' \in B_{\mathcal{C}}(f, r_f)$, the triangle inequality therefore yields

$$d(c, c') \leq d(c, f) + d(f, c') \leq \sqrt{\text{cost}(c, f)} + r_f \leq 2\sqrt{\alpha_c^*}. \tag{16}$$

Since c is not classified as problematic, every client $c' \in B_{\mathcal{C}}(c, 2\sqrt{\alpha_c^*}) \subseteq B_{\mathcal{C}}(c, \gamma_1 \cdot \sqrt{\alpha_c^*})$ satisfies $\alpha_{c',0} > \frac{\alpha_{c,0}}{Q}$. For the sake of contradiction, assume that there are no problematic clients within the

ball $B_C(f, r_f)$. Under this assumption, we have:

$$\begin{aligned}
\sum_{c' \in \mathcal{C}} [\alpha_{c',1} - \text{cost}(c', f)]^+ &\stackrel{(B_C(f, r_f) \subseteq \mathcal{C}, [x]^+ \geq x)}{\geq} \sum_{c' \in B_C(f, r_f)} (\alpha_{c',1} - \text{cost}(c', f)) \\
&\stackrel{(c' \notin \mathcal{C}')}{\stackrel{=}{=}} \sum_{c' \in B_C(f, r_f)} (C_A \cdot \alpha_{c',0} - \text{cost}(c', f)) \\
&\stackrel{(c \notin \mathcal{C}')}{\geq} \sum_{c' \in B_C(f, r_f)} \left(\frac{C_A}{Q} \cdot \alpha_{c,0} - \text{cost}(c', f) \right) \\
&\stackrel{(\alpha_{c,0} \geq \alpha_c^*/C_D^+)}{\geq} \sum_{c' \in B_C(f, r_f)} \left(\frac{C_A}{Q \cdot C_D^+} \cdot \alpha_c^* - \text{cost}(c', f) \right) \\
&\stackrel{(r_f^2 \leq \alpha_c^*)}{\geq} \sum_{c' \in B_C(f, r_f)} \left(\frac{C_A}{Q \cdot C_D^+} \cdot r_f^2 - \text{cost}(c', f) \right) \\
&\stackrel{(C_A \geq Q \cdot C_D^+, \text{ Eq. (9)})}{\geq} \lambda.
\end{aligned}$$

Therefore, f is a paid facility w.r.t the dual values $\alpha_{c,1}$. Because $\alpha_{c,0} \geq \alpha_c^*/C_D^+ = \alpha_c^*/(8\Gamma^4)$ and $\eta = 8000\Gamma^{12}$, we also have $\text{cost}(c, f) \leq \alpha_c^* \leq \eta \cdot \alpha_{c,0}$. Since we assumed that there is no facility f that is paid and satisfies (15), this contradicts the assumption that there is no problematic client $c' \in B_C(f, r_f)$. Consequently, there must exist some $c' \in \mathcal{C}' \cap B_C(f, r_f)$. Since $c' \in B_C(f, r_f)$, it follows that $\alpha_{c'}^* \leq r_f^2 \leq \alpha_c^*$. This further implies that

$$C_D^- \cdot \alpha_{c',0} \leq \alpha_{c'}^* \leq \alpha_c^* \leq C_D^+ \cdot \alpha_{c,0}.$$

Since c' is problematic, there exists a client $c'' \in B_C(c', \gamma_2 \cdot \sqrt{\alpha_{c'}^*})$ such that $\alpha_{c'',0} \leq \frac{\alpha_{c',0}}{Q}$. Consequently, we obtain

$$\alpha_{c'',0} \leq \frac{C_D^+}{C_D^- \cdot Q} \cdot \alpha_{c,0}. \quad (17)$$

Because for $\Gamma \geq 5$, we have $Q = 8\Gamma^4 > C_D^+/C_D^- = 8\Gamma^4/(2\Gamma^2)$, it follows that $\alpha_{c'',0} < \alpha_{c,0}$. Since we assumed that c has the smallest $\alpha_{c,0}$ among all clients c for which there is no paid facility f with $\text{cost}(c, f) \leq \eta \cdot \alpha_{c,0}$, there must exist a paid facility f such that $\text{cost}(c'', f) \leq \eta \cdot \alpha_{c'',0}$. Consequently, we derive the following bound:

$$\begin{aligned}
\text{cost}(c, f) &\stackrel{(\text{Eq. (2)})}{\leq} 3 \cdot (\text{cost}(c, c') + \text{cost}(c', c'') + \text{cost}(c'', f)) \\
&\leq 12\alpha_c^* + 12\gamma_2^2\alpha_{c'}^* + 3\eta \cdot \alpha_{c'',0} \\
&\stackrel{(\alpha_{c'}^* \leq \alpha_c^*, \text{ Eq. (17)})}{\leq} 12C_D^+ \cdot \alpha_{c,0} + 12\gamma_2^2 \cdot C_D^+ \cdot \alpha_{c,0} + 3\eta \cdot \frac{C_D^+}{C_D^- \cdot Q} \alpha_{c,0} \\
&\leq \eta \cdot \alpha_{c,0}
\end{aligned}$$

The second inequality follows from (16), from $c'' \in B_C(c', \gamma_2 \cdot \sqrt{\alpha_{c'}^*})$, and from $\text{cost}(c'', f) \leq \eta \cdot \alpha_{c'',0}$. The last inequality follows from the definitions of the constants C_D^+ , C_D^- , γ_2 , Q , and η , and from $\Gamma \geq 5$ (cf. Eq. (14)). This contradicts the assumption that there is no paid facility f for which (15) holds, and it thus concludes Case 1.

Case 2 ($c \in \mathcal{C}'$): Suppose c is a problematic client. By definition, there then exists a client $c' \in B_{\mathcal{C}}(c, \gamma_2 \cdot \sqrt{\alpha_c^*})$ for which $\alpha_{c',0} \leq \frac{\alpha_{c,0}}{Q} < \alpha_{c,0}$. Since we assumed that c has the smallest $\alpha_{c,0}$ among all clients for which there is no paid facility such that (15) holds, this implies that there exists a paid facility f such that $\text{cost}(c', f) \leq \eta \cdot \alpha_{c',0}$. Using this, we obtain

$$\begin{aligned}
\text{cost}(c, f) &\stackrel{\text{Eq. (2)}}{\leq} 2 \cdot (\text{cost}(c, c') + \text{cost}(c', f)) \\
&\leq 8\gamma_2^2 \cdot \alpha_c^* + 2\eta \cdot \alpha_{c',0} \\
&\stackrel{(\alpha_{c',0} \leq \alpha_{c,0}/Q)}{\leq} 8\gamma_2^2 \cdot C_D^+ \cdot \alpha_{c,0} + 2\eta \cdot \frac{\alpha_{c,0}}{Q} \\
&\leq \eta \cdot \alpha_{c,0}.
\end{aligned}$$

The second inequality follows from $c' \in B_{\mathcal{C}}(c, \gamma_2 \cdot \sqrt{\alpha_c^*})$ and from $\text{cost}(c', f) \leq \eta \cdot \alpha_{c',0}$. The last inequality follows from the definitions of the constants C_D^+ , γ_2 , Q , and η , and from $\Gamma \geq 5$ (cf. Eq. (14)). This again contradicts the assumption that there is no paid facility f for which (15) holds, and it thus concludes Case 2 and therefore the proof of the lemma. \square

Lemma 4.3. *Let $c \in \mathcal{C}$ be a client and $f \in \mathcal{F}$ a facility. If client c κ -approximately contributes to facility f w.r.t the dual values $\alpha_{c,1}$, then*

$$\alpha_{c,1} \leq \rho \cdot r_f^2, \quad \text{where } \rho := \frac{C_A \cdot C_D^+ \cdot Q}{(C_D^-)^2} = 128\Gamma^{12}.$$

Proof. We again distinguish two cases: 1) when c is a non-problematic client ($c \in \mathcal{C} \setminus \mathcal{C}'$) and 2) when c is a problematic client ($c \in \mathcal{C}'$).

Case 1 ($c \in \mathcal{C} \setminus \mathcal{C}'$): Assume, for the sake of contradiction, that $\alpha_{c,1} > \rho \cdot r_f^2$. This directly implies that

$$\alpha_c^* \geq C_D^- \cdot \alpha_{c,0} \stackrel{(c \in \mathcal{C})}{=} \frac{C_D^-}{C_A} \cdot \alpha_{c,1} > \frac{C_D^-}{C_A} \cdot \rho \cdot r_f^2. \quad (18)$$

By the definition of α_c^* , we know that $\alpha_c^* \leq \max\{r_f^2, \text{cost}(c, f)\}$. This implies that

$$\text{cost}(c, f) \geq \alpha_c^* > \frac{C_D^-}{C_A} \cdot \rho \cdot r_f^2 \geq 9 \cdot r_f^2. \quad (19)$$

The last inequality follows because $\rho \geq 9C_A/C_D^-$. Since $B_{\mathcal{C}}(f, r_f)$ is non-empty by the definition of r_f , there exists a client $c' \in B_{\mathcal{C}}(f, r_f)$. Using this, we can derive the following bound:

$$\begin{aligned}
\text{dist}(c, c') &\leq \text{dist}(c, f) + \text{dist}(c', f) \\
&\leq \text{dist}(c, f) + r_f \\
&\stackrel{(\text{Eq. (19)})}{\leq} \text{dist}(c, f) + \frac{\text{dist}(c, f)}{3} \\
&\leq \frac{4}{3} \cdot \sqrt{\kappa \cdot \frac{C_A}{C_D^-} \cdot \alpha_c^*} \\
&\leq \gamma_1 \cdot \sqrt{\alpha_c^*}.
\end{aligned} \quad (20)$$

The second-to-last inequality follows because c κ -approximately contributes to f , and thus $\kappa \cdot \alpha_{c,1} \geq \text{cost}(c, f)$, and from the fact that for $c \in \mathcal{C} \setminus \mathcal{C}'$, $\alpha_{c,1} = C_A \cdot \alpha_{c,0} \leq \frac{C_A}{C_D^-} \cdot \alpha_c^*$. The last inequality holds

because $\gamma_1 \geq \sqrt{2 \cdot \kappa \cdot C_A / C_D^-}$.

Note that because $c' \in B_C(f, r_f)$, we have $\max\{r_f^2, \text{cost}(c', f)\} = r_f^2$ and therefore $\alpha_{c'}^* \leq r_f^2$. This implies

$$\alpha_{c',0} \leq \frac{\alpha_{c'}^*}{C_D^-} \leq \frac{r_f^2}{C_D^-} \leq \frac{C_A}{\rho \cdot (C_D^-)^2} \cdot \alpha_c^* \leq \frac{C_A \cdot C_D^+}{\rho \cdot (C_D^-)^2} \cdot \alpha_{c,0} \leq \frac{\alpha_{c,0}}{Q}.$$

The last inequality follows because $\rho \geq C_A \cdot C_D^+ \cdot Q / (C_D^-)^2$. Thus, client c is a problematic client, contradicting the assumption that c is non-problematic, i.e., $c \in \mathcal{C} \setminus \mathcal{C}'$. This concludes Case 1.

Case 2 ($c \in \mathcal{C}'$): In this case, we have $\alpha_{c,1} = \alpha_{c,0} \leq \alpha_c^* / C_D^-$. By the definition of α_c^* (Eq. (10)), we know that $\alpha_c^* \leq \max\{r_f^2, \text{cost}(c, f)\}$, which implies

$$\alpha_{c,1} \leq \frac{\max\{r_f^2, \text{cost}(c, f)\}}{C_D^-}.$$

Since client c approximately contributes to facility f , we also have $\text{cost}(c, f) \leq \kappa \cdot \alpha_{c,1}$. Substituting into the above inequality, we obtain

$$\text{cost}(c, f) \leq \frac{\kappa}{C_D^-} \cdot \max\{r_f^2, \text{cost}(c, f)\}.$$

Given that $\kappa < C_D^-$, it follows that $\max\{r_f^2, \text{cost}(c, f)\} = r_f^2$. We can therefore conclude that

$$\alpha_{c,1} \leq \frac{r_f^2}{C_D^-} \leq \rho \cdot r_f^2,$$

which concludes Case 2 and thus the proof. \square

Lemma 4.4. *Let $c \in \mathcal{C}$ be a client, and let $f, f' \in \mathcal{F}$ be two facilities. If the client c κ -approximately contributes to both facilities f and f' w.r.t the dual values $\alpha_{c,1}$, then the radii of these facilities satisfy*

$$\max\{r_f, r_{f'}\} \leq \zeta \cdot \min\{r_f, r_{f'}\}, \quad \text{where } \zeta = 1 + 2 \cdot \frac{\sqrt{C_A \cdot C_D^+ \cdot Q \cdot \kappa}}{C_D^-}. \quad (21)$$

We further also have $\text{dist}(f, f') < \frac{\zeta}{2} \cdot \min\{r_f, r_{f'}\}$.

Proof. Without loss of generality, assume that $r_f \leq r_{f'}$. By Lemma 4.3, we have

$$\alpha_{c,1} \leq \rho \cdot r_f^2, \quad \text{where } \rho = \frac{C_A \cdot C_D^+ \cdot Q}{(C_D^-)^2}. \quad (22)$$

Since, c κ -approximately contributes to f and f' , we further have

$$\text{cost}(c, f) \leq \kappa \cdot \alpha_{c,1} \quad \text{and} \quad \text{cost}(c, f') \leq \kappa \cdot \alpha_{c,1}.$$

Combining the two inequalities gives

$$\text{dist}(c, f) \leq \sqrt{\kappa \cdot \rho} \cdot r_f \quad \text{and} \quad \text{dist}(c, f') \leq \sqrt{\kappa \cdot \rho} \cdot r_f$$

and therefore (by the triangle inequality)

$$\text{dist}(f, f') \leq 2 \cdot \sqrt{\kappa \cdot \rho} \cdot r_f. \quad (23)$$

Note that the ball of radius r_f around facility f is contained within the ball of radius $r_f + \text{dist}(f, f')$ around facility f' . We define a new radius r' as

$$r' := r_f + 2 \text{dist}(f, f').$$

Now consider the ball of radius r' around facility f' . We derive

$$\begin{aligned} \sum_{\bar{c} \in B_C(f', r')} (r'^2 - \text{cost}(\bar{c}, f')) &\geq \sum_{\bar{c} \in B_C(f, r_f)} (r'^2 - \text{cost}(\bar{c}, f')) \\ &= \sum_{\bar{c} \in B_C(f, r_f)} (r_f^2 + 4r_f \text{dist}(f, f') + 4 \text{dist}^2(f, f') - \text{cost}(\bar{c}, f')) \\ &\geq \sum_{\bar{c} \in B_C(f, r_f)} (r_f^2 + 4r_f \text{dist}(f, f') + 4 \text{dist}^2(f, f') - (\text{dist}(\bar{c}, f) + \text{dist}(f, f'))^2) \\ &\geq \sum_{\bar{c} \in B_C(f, r_f)} (r_f^2 - \text{cost}(\bar{c}, f) + 4r_f \text{dist}(f, f') - 2r_f \text{dist}(f, f')) \\ &= \sum_{\bar{c} \in B_C(f, r_f)} (r_f^2 - \text{cost}(\bar{c}, f) + 2r_f \text{dist}(f, f')) \\ &\geq \sum_{\bar{c} \in B_C(f, r_f)} (r_f^2 - \text{cost}(\bar{c}, f)) \\ &= \lambda. \end{aligned}$$

The first inequality follows because $B_C(f, r_f) \subseteq B_C(f', r')$. The equation in the second line follows from the definition of r' . The second inequality (in line 3) follows from the triangle inequality. The third inequality follows because $\bar{c} \in B_C(f, r_f)$ and thus $\text{dist}(\bar{c}, f) \leq r_f$. By (9), the fact that $\sum_{\bar{c} \in B_C(f', r')} (r'^2 - \text{cost}(\bar{c}, f')) \geq \lambda$ holds implies that $r_{f'} \leq r'$. Inequality (23) therefore implies that

$$r_{f'} \leq r_f + 2d(f, f') \leq r_f + 2\sqrt{\rho\kappa} \cdot r_f = (1 + 2\sqrt{\rho\kappa}) \cdot r_f.$$

Because we assumed that $r_f \leq r_{f'}$, together with (22) this directly implies that statement of the lemma. \square

Before analyzing the approximation guarantee of the exact algorithm as stated in [Algorithm 1](#), we first analyze an alternative algorithm that is closer to the sequential primal-dual algorithm, but which cannot be easily implemented in the MPC model. In the alternative algorithm, only Step VII of [Algorithm 1](#) is modified, where we decide which facilities to open. We now first describe this alternative Step VII.

Alternative Step VII: Opening Facilities. Let $f \in \mathcal{S}_0$ be a cluster center. We define \mathcal{C}_f to be the set of clients c that κ -approximately contribute to f w.r.t the dual values $\alpha_{c,1}$. Note that if there exists a client that κ -approximately contributes to two facilities $f_1, f_2 \in \mathcal{S}$, then there is an edge between f_1 and f_2 in the dependency graph H and thus also in the extended graph H' . Therefore, the clients in \mathcal{C}_f can only κ -approximately contribute to f and its direct neighbors in H (and possibly to facilities that are not κ -approximately paid and may not be in \mathcal{S}).

The algorithm now computes a dual solution (α_c, β_{cf}) to the LP (4) as follows. For all clients c that are not in the set \mathcal{C}_f for any $f \in \mathcal{S}_0$ (i.e., clients that do not κ -approximately contribute to any cluster center), we set $\alpha_c := \alpha_{c,0}$. Furthermore, for each $f \in \mathcal{S}_0$, let $N_H^+(f)$ denote the set consisting of f and all its neighbors in H .

For each $f \in \mathcal{S}_0$, we determine a value $\xi_f \in \mathbb{R}$, defined as the minimum $\xi \in \mathbb{R}$ for which there exists a facility $f' \in N_H^+(f)$ such that

$$\sum_{c \in \mathcal{C}_f} [\alpha_{c,0} + \xi \cdot (\kappa \cdot \alpha_{c,1} - \alpha_{c,0}) - \text{cost}(c, f')]^+ + \sum_{c \in \mathcal{C} \setminus \mathcal{C}_f} [\alpha_{c,0} - \text{cost}(c, f')]^+ = \lambda.$$

For each $c \in \mathcal{C}_f$, we then set $\alpha_c := \alpha_{c,0} + \xi_f \cdot (\kappa \cdot \alpha_{c,1} - \alpha_{c,0})$. Further, for all c and f , we define $\beta_{cf} := [\alpha_c - \text{cost}(c, f)]^+$. For each $f \in \mathcal{S}_0$, we open exactly one facility $f' \in N_H^+(f)$ that satisfies

$$\sum_{c \in \mathcal{C}_f} [\alpha_c - \text{cost}(c, f')]^+ = \sum_{c \in \mathcal{C}_f} \beta_{cf} = \lambda.$$

We denote the set of open facilities by \mathcal{F}' . By construction, we have $|\mathcal{F}'| = |\mathcal{S}_0|$. The following lemma shows that the computed solution is dual feasible and satisfies all properties required for the analysis of the approximation factor.

Lemma 4.5. *The dual solution computed by the above algorithm is feasible for the LP (4). Moreover, the algorithm guarantees that for every client $c \in \mathcal{C}$:*

(I) *We have $\alpha_{c,0} \leq \alpha_c \leq \kappa \cdot \alpha_{c,1}$, and*

(II) *There is at most one open facility $f \in \mathcal{F}'$ for which $\beta_{cf} = [\alpha_c - \text{cost}(c, f)]^+ > 0$.*

Proof. We first show that for every client $c \in \mathcal{C}$, we have $\alpha_{c,0} \leq \alpha_c \leq \kappa \cdot \alpha_{c,1}$. To show that $\alpha_c \geq \alpha_{c,0}$, we need to show that $\xi_f \geq 0$ for every $f \in \mathcal{S}_0$. This follows from the fact that if all clients choose $\alpha_{c,0}$ as their dual value, then no facility is paid. That is, we need to show that $\sum_{c \in \mathcal{C}} [\alpha_{c,0} - \text{cost}(c, f)]^+ < \lambda$ for every $f \in \mathcal{F}$. If for all $c \in \mathcal{C}$, $\alpha_{c,0} \leq \text{cost}(c, f)$, we have $\sum_{c \in \mathcal{C}} [\alpha_{c,0} - \text{cost}(c, f)]^+ = 0$, which is clearly less than λ . Let us therefore consider a facility $f \in \mathcal{F}$ for which there exists at least one client c with $\alpha_{c,0} > \text{cost}(c, f)$:

$$\begin{aligned} \sum_{c \in \mathcal{C}} [\alpha_{c,0} - \text{cost}(c, f)]^+ &< \sum_{c \in \mathcal{C}} [\alpha_c^* - \text{cost}(c, f)]^+ \\ &\stackrel{\text{Eq. (10)}}{\leq} \sum_{c \in \mathcal{C}} [\max\{r_f^2, \text{cost}(c, f)\} - \text{cost}(c, f)]^+ \\ &= \sum_{c \in \mathcal{C}} [r_f^2 - \text{cost}(c, f)]^+ \\ &\stackrel{\text{Eq. (9)}}{=} \lambda. \end{aligned}$$

The first inequality follows because for all clients c , $\alpha_{c,0} < \alpha_c^*$, and because there exists a client c for which $\alpha_{c,0} > \text{cost}(c, f)$. In order to have a paid facility within $N_H^+(f)$ for some $f \in \mathcal{S}_0$, one therefore needs to choose $\xi_f > 0$.

To show that $\xi_f \leq 1$, observe that every facility $f \in \mathcal{S}$ —and therefore also every facility $f \in \mathcal{S}_0$ —is κ -approximately paid w.r.t the dual values $\alpha_{c,1}$. This directly implies that if all clients $c \in \mathcal{C}_f$ choose $\kappa \cdot \alpha_{c,1}$ as their dual value, then f is paid. The value of ξ_f can therefore not exceed 1.

It remains to show that the computed solution is feasible for the dual LP (4) and that Property (II) holds. To prove dual feasibility, first note that we clearly have $\alpha_c \geq 0$ and $\beta_{cf} \geq 0$ for all $c \in \mathcal{C}$ and $f \in \mathcal{F}$. The inequalities $\alpha_c - \beta_{cf} \leq \text{cost}(c, f)$ also directly hold for all $c \in \mathcal{C}$ and $f \in \mathcal{F}$ because we set the β_{cf} -variables to $\beta_{cf} = [\alpha_c - \text{cost}(c, f)]^+$. It therefore remains to show that for all $f \in \mathcal{F}$, we have $\sum_{c \in \mathcal{C}} \beta_{cf} \leq \lambda$. If we set all dual client variables to $\alpha_{c,0}$, this directly follows from the above calculations. Note that we set $\alpha_c > \alpha_{c,0}$ only for clients $c \in \mathcal{C}_f$ for some $f \in \mathcal{S}_0$. Because for each $f \in \mathcal{S}_0$ and each client $c \in \mathcal{C}_f$, the set $N_H^+(f)$ contains all the facilities for which $\kappa \cdot \alpha_{c,1} \geq \text{cost}(c, f)$, increasing the values of the clients in \mathcal{C}_f can only affect the value of $\sum_{c \in \mathcal{C}} \beta_{cf'}$ for facilities $f' \in N_H^+(f)$. Further, because facilities in \mathcal{S}_0 are at distance at least 4 in H , the sets $N_H^+(f)$ for different $f \in \mathcal{S}_0$ are disjoint. For a specific $f \in \mathcal{S}_0$, the choice of the value ξ_f guarantees that no facility in $N_H^+(f)$ becomes overpaid, and thus the computed solution is feasible for (4).

Let us now prove that Property (II) holds, i.e., that for each client $c \in \mathcal{C}$, there is at most one open facility f for which $\alpha_c > \text{cost}(c, f)$. Note that we can only have $\alpha_c > \text{cost}(c, f)$ if $\kappa \cdot \alpha_{c,1} > \text{cost}(c, f)$, and thus only if c κ -approximately contributes to f w.r.t the dual values $\alpha_{c,1}$. Therefore, if $\alpha_c > \text{cost}(c, f_1) > 0$ and $\alpha_c > \text{cost}(c, f_2) > 0$ for two facilities f_1 and f_2 , then f_1 and f_2 must be neighbors in H . Because the cluster centers \mathcal{S}_0 are at distance at least 4 from each other, and all the open facilities are either cluster centers or direct neighbors of cluster centers, the open facilities of two different clusters are at least two hops apart in H . It is therefore not possible that a client contributes toward the opening cost of two open facilities (i.e., Property (II) holds). \square

We are now ready to prove that [Algorithm 1](#), with the alternative Step VII, computes a constant-factor approximate solution to the facility location problem that satisfies the LMP property.

Lemma 4.6. *Algorithm 1, with the alternative Step VII, opens a set of facilities \mathcal{F}' and connects each client $c \in \mathcal{C}$ to an open facility $f'_c \in \mathcal{F}'$ such that*

$$\sum_{c \in \mathcal{C}} \text{cost}(c, f'_c) \leq \Lambda \cdot (\text{OPT} - |\mathcal{F}'| \cdot \lambda)$$

for some constant $\Lambda \geq 1$, where OPT denotes the objective value of an optimal solution to the given facility location instance.

Proof. We partition the clients \mathcal{C} into three sets: \mathcal{C}_O , \mathcal{C}_N , and \mathcal{C}_F . The clients $c \in \mathcal{C}_O$ are those for which there exists an open facility $f \in \mathcal{F}'$ such that $\alpha_c > \text{cost}(c, f)$. The clients in \mathcal{C}_N are those not in \mathcal{C}_O and are labeled \mathcal{C}^+ in Step VI of [Algorithm 1](#). That is, \mathcal{C}_N consists of clients c for which the facility f_c (as determined in Step VI of the algorithm) is within constant distance in H' from its cluster center. For concreteness, assume that [Algorithm 1](#) guarantees that for clients $c \in \mathcal{C}_N$, the facility f_c is within at most $d_0 = O(1)$ hops in H' from its cluster center. Finally, the clients in \mathcal{C}_F are those that are far from their cluster center. According to [Algorithm 1](#), for clients $c \in \mathcal{C}_F$, the facility f_c is guaranteed to be within $o(\log \log n)$ hops from its cluster center. We analyze the connection costs of the three sets of clients separately.

Let us first focus on the clients in \mathcal{C}_O , which contribute to the opening cost of some open facility. By [Lemma 4.5](#), for every $c \in \mathcal{C}_O$, there is exactly one open facility $f \in \mathcal{F}'$ for which $\alpha_c > \text{cost}(c, f)$.

We have:

$$\begin{aligned}
\sum_{c \in \mathcal{C}_O} \text{cost}(c, f'_c) &= \sum_{c \in \mathcal{C}_O} (\alpha_c - (\alpha_c - \text{cost}(c, f'_c))) \\
&= \sum_{c \in \mathcal{C}_O} (\alpha_c - [\alpha_c - \text{cost}(c, f'_c)]^+) \\
&= \sum_{c \in \mathcal{C}_O} \alpha_c - \sum_{f \in \mathcal{F}'} \sum_{c \in \mathcal{C}} [\alpha_c - \text{cost}(c, f)]^+ \\
&= \sum_{c \in \mathcal{C}_O} \alpha_c - |\mathcal{F}'| \cdot \lambda.
\end{aligned}$$

For clients $c \in \mathcal{C} \setminus \mathcal{C}_O$, the connection cost depends in particular on how far the facility $f_c \in \mathcal{S}$ is from its cluster center. Assume that f_c is d_c hops away (in H') from the cluster center. If the cluster center is d_c hops away from f_c , then the opened facility f'_c is at most $d_c + 1$ hops away from f_c .

Following the definition of the dependency graph H' , there exists an edge between two facilities f and f' if and only if

$$\text{dist}(f, f') \leq O(1) \cdot \max\{r_f, r_{f'}\}.$$

Consequently, along any path in H' , facility radii can change (increase or decrease) by at most a constant factor per hop. Therefore, we obtain:

$$\text{dist}(f_c, f'_c) \leq \sum_{i=1}^{d_c} O(r_{f_i}) \leq r_{f_c} \cdot O(1)^{d_c} = 2^{O(d_c)} \cdot r_{f_c}.$$

Since client c approximately contributes to facility f_c , we have $\text{dist}(c, f_c) = O(r_{f_c}) = O(\sqrt{\alpha_c})$ by [Lemma 4.3](#). Combining these results, we derive:

$$\text{dist}(c, f'_c) \leq \text{dist}(c, f_c) + \text{dist}(f_c, f'_c) \leq O(\sqrt{\alpha_c}) + 2^{O(d_c)} \cdot \sqrt{\alpha_c} = 2^{O(d_c)} \cdot \sqrt{\alpha_c}.$$

Squaring both sides yields

$$\text{cost}(c, f'_c) = \text{dist}^2(c, f'_c) \leq 2^{O(d_c)} \cdot \alpha_c.$$

For each client $c \in \mathcal{C}_N$, we thus have $\text{cost}(c, f'_c) = O(1) \cdot \alpha_c$, and therefore,

$$\sum_{c \in \mathcal{C}_N} \text{cost}(c, f'_c) = O(1) \cdot \sum_{c \in \mathcal{C}_N} \alpha_c.$$

For clients $c \in \mathcal{C}_F$, we have

$$\sum_{c \in \mathcal{C}_F} \text{cost}(c, f'_c) = 2^{o(\log \log n)} \cdot \sum_{c \in \mathcal{C}_F} \alpha_c = o(\log n) \cdot \sum_{c \in \mathcal{C}_F} \alpha_c.$$

Since [Algorithm 1](#) guarantees that $\sum_{c \in \mathcal{C}_F} \alpha_{c,0} \leq \frac{1}{\log n} \cdot \sum_{c \in \mathcal{C}} \alpha_{c,0}$, it follows from the definition of α_c that:

$$\sum_{c \in \mathcal{C}_F} \text{cost}(c, f'_c) = O(1) \cdot \sum_{c \in \mathcal{C}} \alpha_c.$$

Combining the bounds obtained for $\sum_{c \in \mathcal{C}_O} \text{cost}(c, f'_c)$, $\sum_{c \in \mathcal{C}_N} \text{cost}(c, f'_c)$, and $\sum_{c \in \mathcal{C}_F} \text{cost}(c, f'_c)$, we

conclude that the lemma holds for a sufficiently large constant $\Lambda \geq 1$. \square

We can now prove that the original high-level algorithm described in [Algorithm 1](#) (i.e., with Step VII as stated in the pseudocode) also computes a constant-approximate solution to the facility location problem that satisfies the LMP property.

Theorem 4.7. *Algorithm 1 opens a set of facilities \mathcal{F}' and connects each client $c \in \mathcal{C}$ to an open facility $f'_c \in \mathcal{F}'$ such that*

$$\sum_{c \in \mathcal{C}} \text{cost}(c, f'_c) \leq \Lambda \cdot (\text{OPT} - |\mathcal{F}'| \cdot \lambda),$$

for some constant $\Lambda \geq 1$, where OPT denotes the objective value of an optimal solution to the given facility location instance.

Proof. Let \mathcal{F}' denote the set of facilities opened by [Algorithm 1](#). Further, let \mathcal{F}'' denote the set of facilities opened by the variant of the algorithm that uses the alternative Step VII. Both algorithms compute exactly the same clustering, and each opens exactly one facility per cluster. Hence, it clearly holds that $|\mathcal{F}'| = |\mathcal{F}''|$. Additionally, in both algorithms, each client is connected to the facility opened in its own cluster.

For each cluster center $f \in \mathcal{S}_0$, let $\mathcal{C}^{(f)}$ be the set of clients assigned to the cluster of f . For each $f \in \mathcal{S}_0$, let $f'_f \in \mathcal{F}'$ be the facility opened by [Algorithm 1](#) in the cluster of f . Similarly, let $f''_f \in \mathcal{F}''$ be the facility opened by the variant of [Algorithm 1](#) (with the alternative Step VII) in the cluster of f . Moreover, let f^*_f be the facility in the cluster of f minimizing the total cost $\sum_{c \in \mathcal{C}^{(f)}} \text{cost}(c, f^*_f)$. Step VII of [Algorithm 1](#) guarantees that for some constant $\psi \geq 1$:

$$\sum_{c \in \mathcal{C}^{(f)}} \text{cost}(c, f'_f) \leq \psi \cdot \sum_{c \in \mathcal{C}^{(f)}} \text{cost}(c, f^*_f).$$

By the definition of f^*_f , it also clearly holds that:

$$\sum_{c \in \mathcal{C}^{(f)}} \text{cost}(c, f^*_f) \leq \sum_{c \in \mathcal{C}^{(f)}} \text{cost}(c, f''_f).$$

Since every client belongs to exactly one cluster, combining these inequalities gives us:

$$\sum_{c \in \mathcal{C}} \text{cost}(c, f'_c) \leq \psi \cdot \sum_{c \in \mathcal{C}} \text{cost}(c, f''_c).$$

Using $|\mathcal{F}'| = |\mathcal{F}''|$ and applying the result from [Lemma 4.6](#), we obtain:

$$\sum_{c \in \mathcal{C}} \text{cost}(c, f'_c) \leq \psi \cdot \Lambda'' \cdot (\text{OPT} - |\mathcal{F}'| \cdot \lambda),$$

where Λ'' is the approximation ratio provided by [Lemma 4.6](#) for the algorithm variant using the alternative Step VII. Thus, the claim of the theorem follows directly from [Lemma 4.6](#). \square

4.3 High-Level k -Means to Facility Location Reduction Algorithm

We next show how to efficiently compute a constant approximation for the k -means problem by utilizing the existence of a constant factor LMP approximation for the facility location problem. Our approach builds on the randomized k -median algorithm presented by [\[JV01\]](#). We first describe

the algorithm and then analyze its approximation guarantees in [Section 4.4](#). The algorithm consists of the following three steps.

- (I) Given a k -means instance with point set P , $|P| = n$ and an opening cost $\lambda \geq 1$, one can define a facility location instance by setting $\mathcal{F} = \mathcal{C} = P$. In the first step, the algorithm finds two opening costs λ_1 and λ_2 and two corresponding feasible primal and dual facility location solutions. For each solution corresponding to opening cost λ_i (for $i \in \{1, 2\}$), the set of opened facilities is \mathcal{F}_i , each client $c \in \mathcal{C}$ is assigned to a facility $f_c^{(i)} \in \mathcal{F}_i$, the dual client value of c is $\alpha_c^{(i)}$, and the dual values $\beta_{cf}^{(i)}$ are equal to $\beta_{cf}^{(i)} = [\alpha_c^{(i)} - \text{cost}(c, f)]^+$. Both solutions are constant approximations with the LMP property, i.e., for $i \in \{1, 2\}$, we have

$$\sum_{c \in \mathcal{C}} \text{cost}(c, f_c^{(i)}) \leq \Lambda \cdot \left(\sum_{c \in \mathcal{C}} \alpha_c^{(i)} - \lambda_i \cdot |\mathcal{F}_i| \right) \quad (24)$$

for some constant $\Lambda \geq 1$. We further require $k_1 := |\mathcal{F}_1| \leq k$, $k_2 := |\mathcal{F}_2| \geq k$, and $\lambda_2 \leq \lambda_1 \leq 2\lambda_2$. If $k_1 = k$ or $k_2 = k$, we output the corresponding facility location solution as the final k -means solution and stop.

- (II) Determine a set \mathcal{F}'_2 of size $|\mathcal{F}'_2| = k_1$ as follows. For every $f \in \mathcal{F}_1$, add a facility $f'_2(f) \in \mathcal{F}_2$ to \mathcal{F}'_2 such that

$$\text{dist}(f, f'_2(f)) \leq \delta \cdot \min_{f' \in \mathcal{F}_2} \text{dist}(f, f') \quad (25)$$

for some constant $\delta \geq 1$. If the resulting set \mathcal{F}'_2 is of size $< k_1$, add arbitrary additional $k_1 - |\mathcal{F}'_2|$ facilities from $\mathcal{F}_2 \setminus \mathcal{F}'_2$ to \mathcal{F}'_2 .

- (III) Define $a := \frac{k_2 - k}{k_2 - k_1}$ and $b := \frac{k - k_1}{k_2 - k_1}$. The algorithm then determines a set Z of k centers for the k -means solution as follows:

- (A) With probability a , it adds the k_1 points in \mathcal{F}_1 to Z , otherwise (with probability $1 - a = b$) it adds the k_1 points in \mathcal{F}'_2 to Z .
- (B) Additionally, it selects $k - k_1$ uniformly random facilities from $\mathcal{F}_2 \setminus \mathcal{F}'_2$ and adds them to Z .

For each client c , we determine a facility $\phi(c) \in Z$ to which c is connected to as follows:

- If $f_c^{(2)} \in \mathcal{F}'_2$ and the set \mathcal{F}_1 is sampled in Step (A) above, we set $\phi(c) := f_c^{(1)}$.
- If $f_c^{(2)} \in \mathcal{F}'_2$ and the set \mathcal{F}'_2 is sampled in Step (A) above, we set $\phi(c) := f_c^{(2)}$.
- If $f_c^{(2)} \in \mathcal{F}_2 \setminus \mathcal{F}'_2$, we set $\phi(c) := f_c^{(2)}$ if $f_c^{(2)}$ is sampled as one of the additional $k - k_1$ facilities in Step (B) above. Otherwise we set $\phi(c) := f_1^{(1)}$ if \mathcal{F}_1 is sampled in Step (A) above. Otherwise, we set $\phi(c) := f'_2(f_1^{(1)})$.

4.4 High-Level k -Means to Facility Location Reduction Analysis

We now analyze the k -means algorithm described in [Section 4.3](#). We first show that the facility location that was computed for opening cost λ_1 in Step (I) of the algorithm is also a constant-factor LMP approximation for opening cost λ_2 .

Lemma 4.8. *Consider the facility location solution for opening cost λ_1 as defined in Step (I) above. Define updated dual values as follows:*

$$\bar{\alpha}_c^{(1)} := \frac{\lambda_2}{\lambda_1} \cdot \alpha_c^{(1)} \quad \text{and} \quad \bar{\beta}_c^{(1)} := [\bar{\alpha}_c^{(1)} - \text{cost}(c, f)]^+.$$

This dual solution is feasible for LP (4) with opening cost λ_2 and we have

$$\sum_{c \in \mathcal{C}} \text{cost}(c, f_c^{(1)}) \leq \frac{\lambda_1}{\lambda_2} \cdot \Lambda \cdot \left(\sum_{c \in \mathcal{C}} \bar{\alpha}_c^{(1)} - \lambda_2 \cdot k_1 \right)$$

Proof. To prove the feasibility w.r.t. (4), we have to show that for all $f \in \mathcal{F}$, $\sum_{c \in \mathcal{C}} \bar{\beta}_{cf}^{(1)} \leq \lambda_2$ and that for all $c \in \mathcal{C}$ and all $f \in \mathcal{F}$, $\bar{\alpha}_c^{(1)} - \bar{\beta}_{cf}^{(1)} \leq \text{cost}(c, f)$. The second part follows directly from the definition of $\bar{\beta}_{cf}^{(1)}$. For the first part, for all $c \in \mathcal{C}$ and $f \in \mathcal{F}$, we have

$$\bar{\beta}_{cf}^{(1)} = [\bar{\alpha}_c^{(1)} - \text{cost}(c, f)]^+ = \left[\frac{\lambda_2}{\lambda_1} \cdot \alpha_c^{(1)} - \text{cost}(c, f) \right]^+ \leq \left[\frac{\lambda_2}{\lambda_1} \cdot \alpha_c^{(1)} - \frac{\lambda_2}{\lambda_1} \cdot \text{cost}(c, f) \right]^+ = \frac{\lambda_2}{\lambda_1} \cdot \beta_{cf}^{(1)}.$$

The inequality follows because $\lambda_2 \leq \lambda_1$. Feasibility of the dual solution now follows because we already know that $\sum_{c \in \mathcal{C}} \beta_{cf}^{(1)} \leq \lambda_1$

Let us therefore now focus on showing that primal solution is a $\frac{\lambda_1}{\lambda_2} \cdot \Lambda$ -approximate LMP solution for the facility location problem with opening cost λ_2 . From (24), we have

$$\begin{aligned} \sum_{c \in \mathcal{C}} \text{cost}(c, f_c^{(1)}) &\leq \Lambda \cdot \left(\sum_{c \in \mathcal{C}} \alpha_c^{(1)} - \lambda_1 \cdot k_1 \right) \\ &= \frac{\lambda_1}{\lambda_2} \cdot \Lambda \cdot \left(\frac{\lambda_2}{\lambda_1} \cdot \sum_{c \in \mathcal{C}} \alpha_c^{(1)} - \frac{\lambda_2}{\lambda_1} \cdot \lambda_1 \cdot k_1 \right) \\ &= \frac{\lambda_1}{\lambda_2} \cdot \Lambda \cdot \left(\sum_{c \in \mathcal{C}} \bar{\alpha}_c^{(1)} - \lambda_2 \cdot k_1 \right), \end{aligned}$$

which concludes the proof. \square

We now switch our attention to the k -means problem. We begin by formulating the primal integer linear program (ILP) and its corresponding dual for the k -means problem. For convenience, we define k -means as a variant of the facility location problem in which facilities and clients are given, there is a cost for opening facilities, and we are required to open at most k facilities. This formulation is equivalent to the standard k -means problem when \mathcal{F} and \mathcal{C} are identical.

Primal k -Means Integer LP:

$$\begin{aligned}
\min \quad & \sum_{f \in \mathcal{F}} \sum_{c \in \mathcal{C}} \text{cost}(c, f) \cdot x_{cf} \\
\text{s.t.} \quad & \sum_{f \in \mathcal{F}} x_{cf} \geq 1, \quad \forall c \in \mathcal{C}, \\
& x_{cf} \leq y_f, \quad \forall f \in \mathcal{F}, \forall c \in \mathcal{C}, \\
& \sum_{f \in \mathcal{F}} y_f \leq k, \\
& x_{cf} \in \{0, 1\}, \quad \forall f \in \mathcal{F}, \forall c \in \mathcal{C}, \\
& y_f \in \{0, 1\}, \quad \forall f \in \mathcal{F}.
\end{aligned} \tag{26}$$

Dual k -Means LP:

$$\begin{aligned}
\max \quad & \sum_{c \in \mathcal{C}} \alpha_c - \lambda \cdot k \\
\text{s.t.} \quad & \alpha_c - \beta_{cf} \leq \text{cost}(c, f), \quad \forall f \in \mathcal{F}, \forall c \in \mathcal{C}, \\
& \sum_{c \in \mathcal{C}} \beta_{cf} \leq \lambda, \quad \forall f \in \mathcal{F}, \\
& \alpha_c \geq 0, \quad \forall c \in \mathcal{C}, \\
& \beta_{cf} \geq 0, \quad \forall f \in \mathcal{F}, \forall c \in \mathcal{C}.
\end{aligned} \tag{27}$$

Note that, unlike in the dual LP (4) for facility location, in (27), the variable λ is part of the solution.

By using the two facility location solutions with opening costs λ_1 and λ_2 that are computed in Step (I) of the algorithm of Section 4.3 and by using the adapted solution constructed in Lemma 4.8, we now construct two feasible solutions to (26) and (27) with input values k_1 and k_2 . We will refer to the two solutions as solution 1 and solution 2 in the following

For the primal LP (26) with k_i centers, where $i \in \{1, 2\}$, the variables of solution i are set as follows. For every $c \in \mathcal{C}$ and $f \in \mathcal{F}$, define $x_{cf}^{(i)} := 1$ if $f = f_c^{(i)}$, and $x_{cf}^{(i)} := 0$ otherwise. For each $f \in \mathcal{F}$, we set $y_f^{(i)} := 1$ if $f \in \mathcal{F}_2$, and $y_f^{(i)} := 0$ otherwise.

For the dual LP (27), for solution 1, we use the modified dual solution that is constructed in Lemma 4.8. Hence, the variable λ is set to λ_2 for solution 1 and 2. Further, for every $c \in \mathcal{C}$, variable α_c is set to $\bar{\alpha}_c^{(1)}$ for solution 1 and to $\alpha_c^{(1)}$ for solution 2. Further, for all $c \in \mathcal{C}$ and $f \in \mathcal{F}$, we use the variables $\bar{\beta}_{cf}^{(1)}$ for solution 1 and $\beta_{cf}^{(2)}$ for solution 2. Feasibility in both cases follows directly from the fact that the facility location solutions are feasible for (3) and (4).

Moreover, Inequality (24) directly implies that the objective value of the primal k_2 -means solution 2 is at most Λ times the objective value of the corresponding dual solution. Similarly, Lemma 4.8 implies that the primal k_1 -means solution 1 is at most $\frac{\lambda_1}{\lambda_2} \cdot \Lambda \leq 2\Lambda$ times the objective value of the corresponding dual solution. Therefore, if $k_1 = k$ or $k_2 = k$, the respective facility location solution directly yields a 2Λ -approximate solution for the k -means problem.

The next lemma shows that if neither k_1 nor k_2 equals k , it is possible to interpolate between the two solutions to obtain a fractional $\frac{\lambda_1}{\lambda_2} \cdot \Lambda$ -approximate solution for the k -means problem, that is, a feasible solution to the fractional relaxation of (26).

We define two real numbers $a, b \geq 0$ as in Step (III) of the algorithm of Section 4.3, i.e.,

$a := \frac{k_2 - k}{k_2 - k_1}$ and $b := \frac{k - k_1}{k_2 - k_1}$. Note that $a + b = 1$ and $a \cdot k_1 + b \cdot k_2 = k$. We define fractional solutions with variables x_{cf} and y_f for (26), and with variables α_c , β_{cf} , and λ for (27) as follows.

$$\begin{aligned} \forall c \in \mathcal{C}, \forall f \in \mathcal{F} : \quad x_{cf} &= a \cdot x_{cf}^{(1)} + b \cdot x_{cf}^{(2)}, \quad y_f = a \cdot y_f^{(1)} + b \cdot y_f^{(2)} \\ \forall c \in \mathcal{C}, \forall f \in \mathcal{F} : \quad \alpha_c &= a \cdot \bar{\alpha}_c^{(1)} + b \cdot \alpha_c^{(2)}, \quad \beta_{cf} = [\alpha_c - \text{cost}(c, f)]^+ \end{aligned}$$

The value of λ is set to λ_2 . The following lemma shows that this convex combination of the two solutions provides a $\frac{\lambda_1}{\lambda_2} \cdot \Lambda$ -approximate solution for the fractional LP relaxation of (26).

Lemma 4.9. *The above fractional solutions are feasible for the fractional LP relaxation of (26) and for the LP (27). Moreover, we have*

$$\sum_{c \in \mathcal{C}, f \in \mathcal{F}} \text{cost}(c, f) \cdot x_{cf} = \sum_{c \in \mathcal{C}} \left(a \cdot \text{cost}(c, f_c^{(1)}) + b \cdot \text{cost}(c, f_c^{(2)}) \right) \leq \frac{\lambda_1}{\lambda_2} \cdot \Lambda \cdot \left(\sum_{c \in \mathcal{C}} \alpha_c - \lambda_1 \cdot k \right). \quad (28)$$

Note that this implies that the primal solution is optimal up to a factor at most $\frac{\lambda_1}{\lambda_2} \cdot \Lambda$.

Proof. We first prove the feasibility of the solution of (26). The inequalities of the form $\sum_{f \in \mathcal{F}} x_{cf} \geq 1$ and $x_{cf} \leq y_f$ directly follow because the variables x_{cf} and y_f result from a convex combination of two solutions that satisfy these inequalities. The inequality $\sum_{f \in \mathcal{F}} y_f \leq k$ follows because we have $\sum_{f \in \mathcal{F}} y_f^{(i)} \leq k_i$ for $i \in \{1, 2\}$ and because $y_f = a \cdot y_f^{(1)} + b \cdot y_f^{(2)}$ and $k = a \cdot k_1 + b \cdot k_2$.

Let us also prove the feasibility of the solution of (27). The inequalities $\alpha_c - \beta_{cf} \leq \text{cost}(c, f)$ follow directly from the choice of β_{cf} . The inequality $\sum_{c \in \mathcal{C}} \beta_{cf} \leq \lambda = \lambda_2$ follows because

$$\begin{aligned} \sum_{c \in \mathcal{C}} \beta_{cf} &= \sum_{c \in \mathcal{C}} [\alpha_c - \text{cost}(c, f)]^+ \\ &= \sum_{c \in \mathcal{C}} [a \cdot \bar{\alpha}_c^{(1)} + b \cdot \alpha_c^{(2)} - \text{cost}(c, f)]^+ \\ &\stackrel{a+b=1}{=} \sum_{c \in \mathcal{C}} [a \cdot \bar{\alpha}_c^{(1)} - a \cdot \text{cost}(c, f) + b \cdot \alpha_c^{(2)} - b \cdot \text{cost}(c, f)]^+ \\ &\leq \sum_{c \in \mathcal{C}} \left([a \cdot \bar{\alpha}_c^{(1)} - a \cdot \text{cost}(c, f)]^+ + [b \cdot \alpha_c^{(2)} - b \cdot \text{cost}(c, f)]^+ \right) \\ &= \sum_{c \in \mathcal{C}} \left(a \cdot [\bar{\alpha}_c^{(1)} - \text{cost}(c, f)]^+ + b \cdot [\alpha_c^{(2)} - \text{cost}(c, f)]^+ \right) \\ &\leq \lambda_2 \end{aligned}$$

The first inequality holds because for all $A, B \in \mathbb{R}$, we have $[A + B]^+ \leq [A]^+ + [B]^+$. The last inequality follows from $a + b = 1$ and from $\sum_{c \in \mathcal{C}} \bar{\beta}_{cf}^{(1)} \leq \lambda_2$ and $\sum_{c \in \mathcal{C}} \beta_{cf}^{(2)} \leq \lambda_2$.

It remains to prove (28). This however follows because $a + b = 1$, $k = a \cdot k_1 + b \cdot k_2$ and the inequality of (28) therefore just is a convex combination of (24) (for $i = 2$) and of the inequality proven in Lemma 4.8. This concludes the proof. \square

Theorem 4.10. *The k -means algorithm described in Section 4.3 returns a $(2 + 4\delta^2) \cdot \Lambda$ -approximate solution in expectation for any given k -means instance.*

Proof. Consider the connection cost of some client $c \in \mathcal{C}$. In the algorithm of Section 4.3, c is connected either to $f_c^{(1)}$, to $f_c^{(2)}$, or to $f_2'(f_c^{(1)})$. Let us distinguish two cases.

Case 1: $f_c^{(2)} \in \mathcal{F}'_2$: In this case, c is connected to $f_c^{(1)}$ (i.e., $\phi(c) = f_c^{(1)}$) if and only if the set \mathcal{F}_1 is sampled in Step (A) of Step (III) of the algorithm and it is connected to $f_c^{(2)}$ otherwise. The set \mathcal{F}_1 is sampled with probability exactly a . Client c is therefore connected to $f_c^{(1)}$ with probability a and to $f_c^{(2)}$ with probability $b = 1 - a$. The expected connection cost of c is therefore

$$\mathbb{E}[\text{cost}(c, \phi(c))] = a \cdot \text{cost}(c, f_c^{(1)}) + b \cdot \text{cost}(c, f_c^{(2)}).$$

Case 2: $f_c^{(2)} \in \mathcal{F}_2 \setminus \mathcal{F}'_2$: In this case, we have $\phi(c) = f_c^{(2)}$ if and only if $f_c^{(2)}$ is sampled as one of the $k - k_1$ random additional centers in $\mathcal{F}_2 \setminus \mathcal{F}'_2$ in Step (B). The probability for this is $(k - k_1)/|\mathcal{F}_2 \setminus \mathcal{F}'_2| = (k - k_1)/(k_2 - k_1) = b$. We have $\phi(c) = f_c^{(1)}$ if set \mathcal{F}_1 is sampled in Step (A) and $f_c^{(2)}$ is not sampled in Step (B). This happens with probability $a(1 - b) = a^2$. In the remaining cases, i.e., with probability $1 - a^2 - b = ab$, c is connected to $f'_2(f_c^{(1)})$. In this case, the connection cost of c can be bounded as follows. By the definition of $f'_2(f_c^{(1)})$, i.e., by Eq. (25), we have

$$\text{dist}(f_c^{(1)}, f'_2(f_c^{(1)})) \leq \delta \cdot \text{dist}(f_c^{(1)}, f_c^{(2)}),$$

where $\delta \geq 1$ is some constant that is given by Step (II) of the algorithm of Section 4.3. We now have

$$\begin{aligned} \text{dist}(c, f'_2(f_c^{(1)})) &\leq \text{dist}(c, f_c^{(1)}) + \text{dist}(f_c^{(1)}, f'_2(f_c^{(1)})) \\ &\leq \text{cost}(c, f_c^{(1)}) + \delta \cdot \text{cost}(f_c^{(1)}, f_c^{(2)}) \\ &\leq \text{cost}(c, f_c^{(1)}) + \delta \cdot (\text{cost}(f_c^{(1)}, c) + \text{cost}(c, f_c^{(2)})) \\ &\leq (\delta + 1) \cdot (\text{dist}(c, f_c^{(1)}) + \text{dist}(c, f_c^{(2)})). \end{aligned}$$

By squaring on both sides, we obtain

$$\text{cost}(c, f'_2(f_c^{(1)})) \leq 2(\delta + 1)^2 \cdot (\text{cost}(c, f_c^{(1)}) + \text{cost}(c, f_c^{(2)})).$$

The expected connection cost in Case 2 can therefore be upper bounded as

$$\begin{aligned} \mathbb{E}[\text{cost}(c, \phi(c))] &= a^2 \cdot \text{cost}(c, f_c^{(1)}) + b \cdot \text{cost}(c, f_c^{(2)}) + ab \cdot \text{cost}(c, f'_2(f_c^{(1)})) \\ &\leq a^2 \cdot \text{cost}(c, f_c^{(1)}) + b \cdot \text{cost}(c, f_c^{(2)}) + 2(\delta + 1)^2 \cdot (a \cdot \text{cost}(c, f_c^{(1)}) + b \cdot \text{cost}(c, f_c^{(2)})) \\ &= [a + 2(\delta + 1)^2 \cdot b] \cdot a \cdot \text{cost}(c, f_c^{(1)}) + [1 + 2(\delta + 1)^2 \cdot a] \cdot b \cdot \text{cost}(c, f_c^{(2)}) \\ &\leq [1 + 2(\delta + 1)^2] \cdot (a \cdot \text{cost}(c, f_c^{(1)}) + b \cdot \text{cost}(c, f_c^{(2)})). \end{aligned}$$

The last inequality follows from $a, b \leq 1$. The upper bound on the expected connection cost in Case 2 clearly also holds in Case 1. Together with Lemma 4.9, this implies that the expected connection cost is within a factor at most $(1 + 2(\delta + 1)^2) \cdot \frac{\lambda_1}{\lambda_2} \cdot \Lambda \leq (2 + 4(\delta + 1)^2) \cdot \Lambda$ of the objective value of some feasible dual solution to the k -means LP. This concludes the proof. \square

5 Implementation in the MPC Model

In this section, we discuss how to implement our high level facility location algorithm of Algorithm 1 and the randomized rounding algorithm to obtain an approximate k -means solution in a fully

scalable way in the MPC model. As the technically demanding part is the implementation of the facility location algorithm, we first concentrate on the implementation of [Algorithm 1](#) and only discuss the k -means algorithm at the very end.

As discussed in [Section 1](#), by using locality-sensitive hashing techniques, one can approximate the distances in \mathbb{R}^d by a sparse graph with $n^{1+\varepsilon}$ edges for some constant $\varepsilon > 0$. In the following, in [Section 5.1](#), we first provide the details of this approximation. In [Section 5.2](#), we then describe the necessary MPC graph algorithms to implement our facility location and k -means algorithm in the MPC model. Finally, in [Section 5.3](#), we then show how to put the pieces together and thus how to use the graph approximation of \mathbb{R}^d described in [Section 5.1](#) together with the algorithm of [Section 5.2](#) to implement [Algorithm 1](#).

5.1 Approximating the Geometric Space by a Graph

In the following, we consider a set of points $P \subset \mathbb{R}^d$ of size $|P| = n$ equipped with the ℓ_2 -metric. For $x, y \in P$, we use $\text{dist}(x, y)$ to denote the ℓ_2 -distance between x and y . We assume that for any $x, y \in P$, if $x \neq y$, then $\text{dist}(x, y) \geq 1$ and $\text{dist}(x, y) \leq \text{poly}(n)$. Our goal is to obtain a constant approximation of the metric space (P, dist) by (weighted) shortest path metric of a relatively sparse graph. This can be achieved by using locality-sensitive hashing.

Definition 5.1 (Locality-Sensitive Hashing (LSH)). *Consider a family \mathcal{H} of hash functions mapping \mathbb{R}^d into some universe \mathcal{U} . For $D, \Gamma > 1$ and $p_1, p_2 \in [0, 1]$, \mathcal{H} is $(D, \Gamma \cdot D, p_1, p_2)$ -sensitive, if $\forall x, y \in \mathbb{R}^d$, it satisfies:*

- a) If $\text{dist}(x, y) \leq D$, $\Pr_{h \in \mathcal{H}}[h(x) = h(y)] \geq p_1$.
- b) If $\text{dist}(x, y) \geq \Gamma \cdot D$, $\Pr_{h \in \mathcal{H}}[h(x) = h(y)] \leq p_2$.

We next describe locality-sensitive hashing that can be used to compute a graph approximation of (P, ℓ_2) . We first give an algorithm that only takes care of point pairs $(x, y) \in P^2$ for which $\text{dist}(x, y) \approx D$.

LSH-Based Spanner Algorithm for a given distance (a.k.a. scale) D

- a) **Input:** n points $P \subset \mathbb{R}^d$, $\Gamma > 1$, $D > 0$.
- b) Choose a $(D, \Gamma \cdot D, p_1, p_2)$ -sensitive family \mathcal{H} and draw independent h_1, \dots, h_t from H for $t = 5 \ln(n)/p_1$.
- c) Create a graph $G_D = (P, E_D)$ where each point $x \in P$ denotes a vertex in the graph.
- d) For each hash function h_i , $i \in [t]$ and each $u \in \mathcal{U}$ for which there is some $x \in P$ for which $h_i(x) = u$, let $x_u \in P$ be an arbitrary point for which $h_i(x_u) = u$ and add an edge $\{x_u, y\}$ to E_D between x and every other $y \in P$ for which $h_i(y) = h_i(x_u) = u$.
- e) Return G_D

We have the following lemma.

Lemma 5.2. *Consider a set $P \subset \mathbb{R}^d$ of n points and $\Gamma > 1, D > 0$. Let $G_D = (P, E_D)$ be the output of the above algorithm. Then the number of edges is $|E_D| \leq \frac{5 \ln n}{p_1}$ and with probability at least $1 - \max\{1/n^3, 5p_2p_1^{-1}n^2 \ln n\}$, the following holds: (I) $\forall x, y \in P$ with $\text{dist}(x, y) \leq D$, either there is an edge between x and y in G or x and y have a common neighbor in G . (II) If there is an edge between $x, y \in P$ in G , $\text{dist}(x, y) < \Gamma \cdot D$.*

Proof. The bound on $|E_D|$ follows because the number of edges added per hash function h_i is at most $n - 1$ and there are $5 \ln(n)/p_1$ hash functions h_i .

For any $x, y \in P$, there is a path of length ≤ 2 in the graph G_D if there exists an $i \in \{1, \dots, t\}$ for which $h_i(x) = h_i(y)$. By the assumption that \mathcal{H} is a $(D, \Gamma \cdot D, p_1, p_2)$ -sensitive family of hash functions, for every $x, y \in P$ with $\text{dist}(x, y) \leq D$ and every i , we have $\Pr(h_i(x) = h_i(y)) \geq p_1$. The probability that x and y are not connected by a path of length ≤ 2 is therefore at most

$$1 - (1 - p_1)^t \geq 1 - e^{-p_1 \cdot t} = 1 - e^{-p_1 \cdot \frac{5 \ln n}{p_1}} = \frac{1}{n^5}.$$

By a union bound over the at most $n^2/2$ pairs of nodes x, y with $\text{dist}(x, y) \leq D$, the probability that each such pair is connected by a path of length at most 2 and thus (I) holds is therefore at least $1 - \frac{1}{2n^3}$.

In order for $x, y \in P$ to be connected by an edge, we must have $h_i(x) = h_i(y)$ for some $i \in \{1, \dots, t\}$. If $\text{dist}(x, y) \geq \Gamma \cdot D$, the probability that this happens for a specific i is at most p_2 . Taking a union bound over all $t = 5 \ln(n)/p_1$ hash functions h_i and all at most $n^2/2$ pairs of nodes x, y , we can upper bound the probability that (II) does not hold as $p_2 \cdot \frac{5 \ln n}{p_1} \cdot \frac{n^2}{2}$. \square

Lemma 5.3. [AI06, CMZ22] For any “scale” $D > 0$, dimension $d > 0$, $c > 0$ and $s \in [1, \log n]$, there is a $(D, \Gamma \cdot D, 1/n^{s/\Gamma^2 + o(1)}, 1/n^s)$ -sensitive family \mathcal{H} of hashing functions for \mathbb{R}^d . In addition, for any set of n points $P \subset \mathbb{R}^d$ and any $h \in \mathcal{H}$, there is a fully scalable MPC algorithm with $O(1)$ rounds and $n^{1+o(1)}d$ total space that computes $h(x)$ for every $x \in P$. The space to store each $h(x)$ is $O(\log^3 n)$.

We now have everything that we need to describe the graph approximation of $(\mathbb{R}^d, \text{dist})$ that we use.

Lemma 5.4. Let P be a set of $|P| = n$ points in \mathbb{R}^d and let $\varepsilon > 0$ be an arbitrary constant. There is a weighted graph $G = (P, E)$ with node set P and $|E| \leq n^{1+\varepsilon}$ edges and positive edge weights $w(e) \geq 1$ such that for some constant Γ , G has the following properties

- (I) For any two nodes $x, y \in P$, either G contains an edge $e = \{x, y\}$ of length $w(e) \leq \text{dist}(x, y)/2$ or there exists a point $z \in P$ such that $e_1 = \{x, z\} \in E$, $e_2 = \{z, y\} \in E$, and $w(e_1) + w(e_2) \leq \text{dist}(x, y)$.
- (II) For any two nodes $x, y \in P$, $\text{dist}(x, y) \leq \Gamma \cdot d_G(x, y)$, where $d_G(x, y)$ denotes the weighted shortest path distance in G .

Further, there is a randomized MPC algorithm to compute G in $O(1)$ rounds and with a global memory of size $O(n^{1+\varepsilon})$.

Proof. We normalize the distances such that $\text{dist}(x, y) \in [1, O(n^6)]$ for all $x, y \in P$ (see Section 2). Let $D_i = 2^i$ for $i \in [0, O(\log n)]$ and define G_i to be the graph constructed using Lemma 5.2 with parameter $D = D_i$.

In G_i , for every pair $x, y \in P$ with $\text{dist}(x, y) \leq D_i$, either $\{x, y\} \in E_i$ or there exists $z \in P$ such that $\{x, z\}, \{z, y\} \in E_i$. For all edges $\{x, y\} \in E_i$, assign weight $w(x, y) = D_i/4$. We take the union of all G_i over all i to form G and for each edge, retain the minimum weight among those assigned in different G_i .

Suppose $\text{dist}(x, y) \in (D_{i-1}, D_i]$. Then $w(x, y) = D_i/4 < \text{dist}(x, y)$, and for pairs connected through an intermediate point z , we have $w(x, z) + w(z, y) \leq D_i/2 < \text{dist}(x, y)$. This establishes Property (I).

For Property (II), note that any path of edges in G consists of edges of length at most D_i , so the total length over any shortest path is at most $\Gamma \cdot \text{dist}(x, y)$ for some constant Γ .

To analyze the complexity, observe that for each scale $D_i = 2^i$, we construct a graph G_i using [Lemma 5.2](#). Each such construction requires a family of $(D_i, \Gamma \cdot D_i, p_1, p_2)$ -sensitive hash functions, which, by [Lemma 5.3](#), can be computed for all $x \in P$ in $O(1)$ MPC rounds using $n^{1+o(1)}d$ total space. Since we have $O(\log n)$ different scales D_i , we perform $O(\log n)$ such graph constructions in parallel, so the total space for all hash functions is $\tilde{O}(n^{1+o(1)})$. Each G_i contains at most $p_1^{-1} \cdot 5 \log n$ edges, and taking the union across all i yields at most $p_1^{-1} \cdot 5 \log^2 n$ total edges.

By choosing parameters so that $p_1 = 1/n^{s/\Gamma^2+o(1)}$, we ensure that the total number of edges in G is at most $n^{s/\Gamma^2+o(1)} \log^2 n$. Choosing $\Gamma = O(1/\sqrt{\epsilon})$ implies:

$$|E| \leq \log^2 n \cdot n^{s \cdot \epsilon + o(1)}.$$

To ensure $|E| \leq n^{1+\epsilon}$, we select

$$s = \frac{1+\epsilon}{\epsilon} - o(1).$$

Since $\log^2 n = n^{o(1)}$, it follows that the number of edges is bounded by $n^{1+\epsilon}$ as required.

Finally, since each edge weight is derived from a constant number of scales and updating the minimum weights can be done in parallel, the final graph G can be computed in $O(1)$ MPC rounds using total space $\tilde{O}(n^{1+\epsilon})$. \square

5.2 MPC Graph Algorithms

Since we can approximate the underlying Euclidean metric by a graph, some of our algorithms become standard MPC graph algorithms. We therefore next discuss some MPC algorithms that directly work on a given undirected graph $G = (V, E)$. When designing those MPC algorithms, we assume that each machine has a local memory of n^δ words, where $n = |V|$ is the number of nodes and $\delta > 0$ is an arbitrary constant (and where each word typically consists of $\Theta(\log n)$ bits). Note that in the context of graph algorithms, this setting is known as the sublinear-memory MPC regime. If not specified explicitly, we will in the following assume that an MPC algorithm is allowed to use a total memory of $O(m + n)$ bits, where $m = |E|$ is the number of edges of G . That is, we allow a total space that is by a poly($\log n$) factor larger than what is needed to store G .

We assume that G is given as a list of nodes and edges. To be able to distinguish different nodes, each node has a unique $O(\log n)$ -bit name, which we will refer to as the ID of the node. Since sorting can be done in constant time and with linear global memory in the MPC model [\[GSZ11\]](#), we can assume that the edges of G are sorted by one of their nodes. If for each edge $e = \{u, v\} \in E$, we store a copy of e for u and a copy of e for v , we can assume that the edges of each node are stored consecutively. If the degree of a node is at most n^δ , all its edges can be stored on a single machine. Otherwise, the edges of a node will be stored on consecutive machines. In this way, operations such as computing a sum or a minimum overall neighbors for each node are straightforward to implement in $O(1)$ time. For our algorithm, we will however also need to implement such basic operations and run some algorithms on the graph G^t for some $t = O(1)$, where G^t is the graph on nodes V with an edge between any two nodes that are in hop distance $\leq t$ in G . Note that already for $t = 2$, G^t can be a much denser graph than G and we can therefore not store G^t explicitly without increasing the global memory too substantially. We therefore need to implement algorithms on G^t as algorithms that operate on G . The next two lemmas provide some basic operations on G^t that can be implemented efficiently by using standard techniques. In the following, we use $N_t(v) := \{u \in V : d_G(u, v) \leq t\}$ to denote the t -hop neighborhood of v in G (for convenience,

including the node v itself).

For convenience, we introduce the following notation to denote the ℓ smallest elements of a set $A \subseteq \mathbb{X}$ of some totally ordered domain \mathbb{X} .

$$\text{min-}\ell(A) := A' \subseteq A \text{ such that } |A'| = \min\{\ell, |A|\} \wedge \forall (a, b) \in A' \times (A \setminus A') : a < b. \quad (29)$$

Lemma 5.5. *Let $G = (V, E)$ be an n -node graph and let $\ell \geq 1$ be an integer parameter such that $\ell \leq n^{\delta/2}$ (where n^δ is the memory of a single machine). Assume that as input, each node $u \in V$ has a set $A_u \subseteq \mathbb{X}$ of size $|A_u| \leq \ell$, where \mathbb{X} is a totally ordered set of size $|\mathbb{X}|$ at most $\text{poly}(n)$. Let $t \geq 1$ be an integer. As output, every node u must output the set $B_u = \text{min-}\ell(\bigcup_{v \in N^t(u)} A_v)$. If one can use $O((m+n) \cdot \ell)$ bits of global memory, the sets B_u can be computed in $O(t)$ rounds in the MPC model.*

Proof. First, note that it is sufficient to show that the problem can be solved for $t = 1$ in $O(1)$ rounds. To see this, note that for every $u \in V$ and every $t > 1$, we have

$$\text{min-}\ell\left(\bigcup_{v \in N^1(u)} \text{min-}\ell\left(\bigcup_{w \in N^{t-1}(v)} A_w\right)\right) = \text{min-}\ell\left(\bigcup_{v \in N^t(u)} A_v\right)$$

We can therefore break the task of finding the ℓ smallest values in the t -hop neighborhood into t times consecutively finding the ℓ smallest values in the 1-hop neighborhood. For the remainder of the proof, we therefore restrict ourselves to the case $t = 1$.

Because we have $O((m+n) \cdot \ell)$ bits of global memory, we can explicitly store every edge $\{u, v\}$ together with the input sets A_u and A_v of its two nodes. As usual, we store each edge twice, once for each of its nodes and we sort the edges by the nodes so that for each node, all its edges are stored consecutively. We can do this so that a) if all edges $\{u, v\}$ of a node u (including the sets A_u and A_v) fit on one machine, they are all stored on one machine and b) otherwise, we have a set of machines that only store the edges of node u . For the nodes u for which all edges fit on one machine, we can compute $\text{min-}\ell(\bigcup_{v \in N^1(u)} A_v)$ without communication. Consider a node u for which the edges $\{u, v\}$ do not fit on one machine and let \mathcal{M}_u be the set of machines that store u 's edges $\{u, v\}$ together with the sets A_u and A_v . We connect the machines into a $n^{\delta/2}$ -ary aggregation tree (of height at most $O(1/\delta) = O(1)$). We can aggregate the smallest v values among the neighbors of v on this aggregation tree in $O(1)$ time. To see that we have enough bandwidth, note that each machine on this tree needs to receive at most ℓ values from each of its $n^{\delta/2}$ children in the tree. Because $\ell \leq n^{\delta/2}$, the total number of values a machine has to receive in a single round is n^δ . This concludes the proof. \square

Lemma 5.6. *Let $G = (V, E)$ be a graph, let $\varepsilon \geq 1/\text{poly log } n$, let $t \geq 0$ be a non-negative integer, and assume that every node $u \in V$ is given an input value $x_u \geq 0$. For each node $u \in V$, we define $X_u^{(t)} := \sum_{v \in N^t(u)} x_v$. There is an $O(t)$ -round sublinear-memory MPC algorithm to compute a value $\tilde{X}_u^{(t)}$ such that $X_u^{(t)} \leq \tilde{X}_u^{(t)} \leq (1 + \varepsilon)X_u^{(t)}$. The algorithm is randomized and succeeds with high probability.*

Proof. To approximate the sum $X_u^{(t)}$ of the values x_v , we can use the technique described in [MS08], where it is shown that the computation of a $(1 + \varepsilon)$ -approximation of the sum of n values can be reduced to $O(\log(n)/\varepsilon^2)$ (independent) instances of computing the minimum of n values. We can apply the algorithm of Lemma 5.5 for $\ell = 1$ to determine the minimum value in the t -hop neighborhood $O(\log(n)/\varepsilon^2)$ times in parallel to compute $\tilde{X}_v^{(t)}$. \square

As a key step of our facility location algorithm, we have to find a set of center nodes in the dependency graph of paid (or approximately paid) facilities so that two nodes in the set are sufficiently separated and ideally, all nodes are within a constant distance of those center nodes. Such sets are known as ruling sets [AGLP89]. Formally, an (a, b) -ruling set of a graph $G = (V, E)$ is a subset $S \subseteq V$ of the nodes so that for all $u, v \in S$, $d_G(u, v) \geq a$ and for all $u \notin S$, there exists $v \in S$ such that $d_G(u, v) \leq b$. There is quite extensive literature on distributed and also MPC algorithms for computing ruling sets. However, unfortunately, for the sublinear-memory regime, it is not known if an (a, b) -ruling set for any $a \geq 2$ and $b = O(1)$ can be computed in time $\text{poly}(\log \log n)$ in the distributed or in the sublinear memory MPC setting. In fact, it is not even known if such an algorithm exists for $b = O(\log \log n)$ if $a > 2$. As a first step, we show how to compute an (a, b) -ruling set for $a > 1$ and $b = O(\log \log \log n)$ by using a bit of additional global memory and an algorithm of [KPP20].

Lemma 5.7. *Let $G = (V, E)$ be an n -node graph with $m = |E|$ edges and let $U \subseteq V$. For every $t \geq 2$, there exists a sublinear-memory MPC algorithm that computes an $(2, O(\log \log \log n))$ -ruling set $S \subseteq U$ of $G^t[U]$ in $O(t \cdot \log \log \log n + \log \log n \cdot \log \log \log n)$ rounds. The algorithm requires $O((m + n) \cdot n^\varepsilon)$ bits of global memory, where $\varepsilon > 0$ is a constant that can be chosen arbitrarily small.*

Proof. We first describe the high-level idea of the algorithm. Note that a set $S \subseteq U$ is a $(2, O(\log \log \log n))$ -ruling set of $G^t[U]$ if and only if S is an independent set of G^t and for every node $u \in U \setminus S$, there exists a node $v \in S$ at distance $O(\log \log \log n)$ in G^t . The algorithm consists of $O(1/\varepsilon)$ phases. At the beginning $S = \emptyset$ and the set of nodes that still needs to be covered is $U_0 = U$. After phase p , we have already selected a set $S_p \subseteq U$ to be in S and the set of uncovered nodes in U is U_p (i.e., U_p are the nodes $u \in U$ for which there is no node $v \in S_p$ that is within distance $O(\log \log \log n)$ in G^t). The goal of each phase is to reduce the maximum degree of the remaining subgraph of G^t by a factor $O(n^\varepsilon)$. More precisely, we will compute $S_p \supseteq S_{p-1}$ such that $G^t[U_p]$ has maximum degree at most $n^{1-\varepsilon p}$. The number of phases is therefore at most $O(1/\varepsilon) = O(1)$. In the following, we describe how a single phase p is implemented. At the same time, we also show by induction that for every $p \geq 0$, the maximum degree of $G^t[U_p] \leq n^{1-\varepsilon p}$. This condition is clearly true for $p = 0$.

We assume that at the beginning of phase $p \geq 1$, we know the set U_{p-1} of uncovered nodes. By using the induction hypothesis, we further assume that the maximum degree of $G^t[U_{p-1}]$ is at most $n^{1-(p-1)\varepsilon}$. In phase p , we first independently mark every node in U_{p-1} with probability $c \cdot \log(n)/n^{1-\varepsilon p}$ for a sufficiently large constant $c > 0$. Let M_p be the set of those marked nodes in phase p . Note that in the graph $G^t[U_{p-1}]$, every node of degree at least $n^{1-\varepsilon p}$ with high probability has at least one neighbor in M_p . Assume that we can compute a $(2, b)$ -ruling set S_p^+ of $G^t[M_p]$ for some $b = O(\log \log \log n)$. By setting $S_p = S_{p-1} \cup S_p^+$ and including in U_p all nodes from U_{p-1} that are at a G^t -distance greater than $b + 1 = O(\log \log \log n)$ from S_p^+ , the maximum degree of $G^t[U_p]$ is reduced to at most $n^{1-\varepsilon p}$, as required. Once, S_p^+ is known, we can use Lemma 5.5 with $\ell = 1$ to compute U_p in $O(t \log \log \log n)$ rounds. It remains to show that we can compute a $(2, O(\log \log \log n))$ -ruling set of $G^t[M_p]$.

We first show that the graph $G^t[M_p]$ can be explicitly constructed. Because the maximum degree of $G^t[U_{p-1}]$ is at most $n^{1-\varepsilon(p-1)}$ and every node in U_{p-1} is added to M_p with probability $c \cdot \log(n)/n^{1-\varepsilon p}$, w.h.p., for each node in U_{p-1} there are at most $c' \cdot n^\varepsilon \log n$ marked nodes in the t -hop neighborhood in G , where $c' > 0$ is a sufficiently large constant. We choose ε such that $c' \cdot n^\varepsilon \log n \leq n^{\delta/2}$. Then, by using Lemma 5.5 with $\ell = O(n^\varepsilon \log n)$, every node in U_{p-1} can obtain the IDs of all the marked nodes in its t -hop neighborhood in G in time $O(t)$. We can therefore explicitly construct the graph $G^t[M_p]$. We can now use the algorithm of Kothapalli, Pai, and Pemmaraju [KPP20] to

compute a $(2, O(\log \log \log n))$ -ruling set of $G^t[M_p]$ in $O(\log \log n \cdot \log \log \log n)$ rounds. The global memory needed by this algorithm is $O(m' + n^{1+\varepsilon})$, where m' is the number of edges of $G^t[M_p]$. \square

We are not aware of a sublinear-memory $\text{poly}(\log \log(n))$ -time MPC algorithm to compute a ruling set of G^t that is better than what [Lemma 5.7](#). Unfortunately, this does not suffice to obtain a constant approximation in our facility location algorithm (and thus in our k -means algorithm). For this, we would need a $(2, O(1))$ -ruling set of G^t for some sufficiently large constant t . Since such an algorithm is not known, we have to instead settle for an algorithm that computes centers that only cover most of the nodes within constant distance.

To this end, we can employ a variant of Luby's classic parallel MIS algorithm [[ABI86](#), [Lub86](#)]. In Theorem 2 of [[BGKO23](#)], it is shown that if we do one step of Luby's algorithm and if, after adding the nodes of the step to the independent set, instead of only the direct neighbors, we remove the 2-hop neighborhoods of those nodes, then a constant fraction of the graph gets removed with constant probability.

Adapted Luby Step Consider the following process on an undirected graph $H = (V_H, E_H)$. Assume that every node $v \in V_H$ has an estimate $\hat{d}(v)$ of its degree in H such that $\deg_H(v) \leq \hat{d}(v) \leq 2\deg_H(v)$. Each node $v \in V_H$ is added to the set S_H independently with probability $c \log n / \hat{d}(v)$ for a sufficiently large constant $c > 0$.

Lemma 5.8. *When applying the above step on a graph H , with high probability, a constant fraction of the nodes in V_H are within distance at most 2 of S_H . Furthermore, for every node u , there are at most $O(\log n)$ neighbors $v \in S_H$ with $\hat{d}(v) \geq \hat{d}(u)$.*

Proof. We first prove that, with high probability, a constant fraction of the nodes in V_H are within distance at most 2 of S_H . To this end, we classify nodes as either *good* or *bad*. A node $v \in V_H$ is considered *good* if the total marking probability over its inclusive 2-hop neighborhood, defined as $N_2^+(v) := \{u \in V_H : d_H(u, v) \leq 2\}$, satisfies $\sum_{u \in N_2^+(v)} p_u \geq 1/2$. Otherwise, v is deemed *bad*.

Let $B \subseteq V_H$ denote the set of bad nodes. For each $v \in B$, we have by definition that $\sum_{u \in N_2^+(v)} p_u < 1/2$. To bound $|B|$, we distribute the *badness* of each $v \in B$ to its immediate neighbors $u \in N_H(v)$ by assigning a charge of $1/\hat{d}(v)$ to each such neighbor. Since $\sum_{u \in N_H(v)} 1/\hat{d}(v) = 1$, each bad node contributes a total charge of 1.

Consider now a node $u \in V_H$ that may receive charge from multiple bad neighbors. The total charge received by u is bounded by the marking probabilities in its neighborhood, which by assumption is less than $1/2$. Hence, no node receives more than $1/2$ units of total charge.

Since the total charge distributed by the bad nodes is equal to $|B|$, and no node can receive more than $1/2$ units of charge, it follows that $|B| \leq |V_H|/2$. Consequently, at least half of the nodes are good, which means that a constant fraction of nodes are within distance at most 2 of S_H , as required.

We now turn to the second part of the lemma and show that, with high probability, for any node $u \in V_H$, the number of neighbors $v \in S_H$ with $\hat{d}(v) \geq \hat{d}(u)$ is $O(\log n)$. For each neighbor $v \in N_H(u)$, the probability that $v \in S_H$ is $p_v = \frac{c \log n}{\hat{d}(v)}$. Since we are only considering neighbors with $\hat{d}(v) \geq \hat{d}(u)$, we have $p_v \leq \frac{c \log n}{\hat{d}(u)}$.

The expected number of such neighbors is then at most $\deg_H(u) \cdot \frac{c \log n}{\hat{d}(u)} \leq c \log n$, using the fact that $\hat{d}(u) \geq \deg_H(u)$. Applying a Chernoff bound, we conclude that with high probability, the number of such neighbors does not exceed $O(\log n)$. \square

The following lemma provides our main ruling set algorithm that we use to compute the center facilities in [Algorithm 1](#).

Lemma 5.9. *Let $G = (V, E)$ be a n -node graph with $m = |E|$ edges and node weights $w(v)$ such that $w(v) \geq 1$ and $w(v) \leq w_{\max} = \text{poly}(n)$. Further, assume that we are given a parameter $\varepsilon \geq 1/\text{poly}(\log n)$ and an integer $t \geq 1$. There exists an $O(t \cdot \log \log n)$ -round sublinear-space MPC algorithm that computes a set of nodes S with the following properties.*

- (I) *The nodes in S form an independent set of G^t*
- (II) *Let $U := \{u \in V : \exists v \in S \text{ with } d_G(u, v) \leq 5t\}$, $W_V := \sum_{v \in V} w(v)$, and $W_U := \sum_{v \in U} w(v)$. W.h.p., we have $W_U \geq (1 - \varepsilon) \cdot W_V$.*
- (III) *For all $u \in V$, there exists a $v \in S$ for which $d_G(u, v) \leq O(t \log \log \log n)$.*

The algorithm requires global memory $O((m + n) \cdot n^\varepsilon)$ for a constant $\varepsilon > 0$ that can be chosen arbitrarily small.

Proof. As a first step, for every $v \in V$, we define $w'(v) := 2^{\lfloor \log_2 w(v) \rfloor}$, i.e., we round each weight $w(v)$ down to the next integer power of 2. In this way, we only have $O(\log n)$ different weights $w'(v) \in \{2^0, 2^1, \dots, 2^h\}$, where $h = \lfloor \log_2 w_{\max} \rfloor = O(\log n)$. For each of the possible rounded node weights 2^ℓ , let V_ℓ be the set of nodes with $w'(v) = 2^\ell$.

We now focus on one value of k . We define $H_\ell := G^{2t}[V_\ell]$ to be the subgraph of G^{2t} induced by the nodes V_ℓ of rounded weight $w'(v) = 2^\ell$. We first show that one *Adapted Luby Step* as described above can be implemented on H_ℓ in $O(t)$ rounds in the MPC model. By Lemma 5.6 each node $v \in V_\ell$ can compute an estimate $\hat{d}(v)$ of its degree $\deg_{H_\ell}(v)$ in H_ℓ such that $\deg_{H_\ell}(v) \leq \hat{d}(v) \leq 2 \deg_{H_\ell}(v)$. Each node $v \in V_\ell$ can then be independently added to the set S_{H_ℓ} with probability $1/\hat{d}(v)$. Finally, one can use Lemma 5.5 to determine the set of nodes $U_\ell \subseteq V_\ell$ such that for $u \in U_\ell$, there is a node $v \in S_{H_\ell}$ within distance $4t$ in G . Note that U_ℓ includes all nodes $u \in V_\ell$ that have a node $v \in S_{H_\ell}$ within distance 2 in H_ℓ . By Lemma 5.8, we therefore know that w.h.p., U_ℓ contains a constant fraction of the nodes in V_ℓ . If we think of the nodes in U_ℓ as being removed from the graph, we can repeat this process on the remaining nodes. If we repeat this $c \cdot \log(1/\varepsilon)$ times for a sufficiently large constant c , the number of remaining nodes is at most $\varepsilon/2 \cdot |V_\ell|$. The final set S_{H_ℓ} is defined as the union of the sets that we construct in those $O(\log 1/\varepsilon)$ iterations.

We next move our attention to the graph G^t . For a node $u \in V$, let $N_\ell^t(u) := S_{H_\ell} \cap N^t(u)$ be the set of S_{H_ℓ} -neighbors of u in G^t . Note that the nodes in $N_\ell^t(u)$ are forming a clique in the graph $H_\ell = G^{2t}[V_\ell]$. By Lemma 5.8, for every node $v \in S_{H_\ell}$, w.h.p., there are at most $O(\log n)$ nodes $w \in S_{H_\ell}$ such that v and w are neighbors in H_ℓ and $\hat{d}(w) \geq \hat{d}(v)$. The edges of the clique induced by the nodes $N_\ell^t(u)$ in H_ℓ can therefore be oriented such that each node $v \in N_\ell^t(u)$ has at most $O(\log n)$ outneighbors $w \in N_\ell^t(u)$. This directly implies that $|N_\ell^t(u)| = O(\log n)$. For every node $u \in V$, the number of S_{H_ℓ} -neighbors in G^t is therefore bounded by $O(\log n)$, w.h.p.

We next define $S_H := \bigcup_{\ell=0}^h S_{H_\ell}$. Note that since the sets S_{H_ℓ} can be computed in parallel (by increasing the global space by an $O(\log n)$ -factor), the set S_H can be computed in $O(\log \log n)$ rounds. We compute a set S' that satisfies conditions (I) and (II) of the lemma by computing a maximal independent set (MIS) on the induced subgraph $G^t[S_H]$ of G^t . Note that when doing this, every node $u \in V$ that is within distance at most $4t$ from a node $v \in S_H$ in G is within distance at most $5t$ of a node in S' . Because in every set V_ℓ , the number of nodes that is not within distance $4t$ of a node in S_{H_ℓ} (and thus within distance $5t$ of a node in S') is at most $\varepsilon/2 \cdot |V_\ell|$, overall, the set of nodes that is not within distance $5t$ of a node in S' has rounded weight at most an $\varepsilon/2$ -fraction of the total rounded weight of all the nodes in V . For the original weights, we therefore have $W_U \geq (1 - \varepsilon) \cdot W_V$ as required.

We need to show that given S_H , the set S' can be computed in $O(t \cdot (\log \log n \cdot \log \log \log n))$ rounds. We observed above that for every $u \in V$, the number of S_{H_ℓ} -neighbors in G^t is at most

$O(\log n)$. Therefore, for every $u \in V$, the number of S_H -neighbors in G^t is at most $O(\ell \log n) = O(\log^2 n)$. The graph $G^t[S_H]$ therefore has maximum degree $O(\log^2 n)$. By using [Lemma 5.5](#), the edges of the graph $G^t[S_H]$ can be computed explicitly in $O(t)$ rounds. We can then compute S' by using the MIS algorithm of [\[GU19\]](#). For graphs of maximum degree Δ , the algorithm of [\[GU19\]](#) has a round complexity of $O(\sqrt{\log \Delta} \cdot \log \log \Delta + \sqrt{\log \log n})$. We can therefore compute the set S' from the set S_H in time $O(t + \sqrt{\log \log n} \cdot \log \log \log n)$.

The set S' only satisfies conditions (I) and (II) of the lemma statement. To also satisfy condition (III), we have to add some nodes to S' to make sure that every node that is not covered by S' is within distance $O(t \log \log \log n)$ of the final set S . Let $U \subseteq V$ be the set of nodes that are not within distance $O(t \log \log \log n)$ of S' . We can apply [Lemma 5.7](#) to compute a set $S^+ \subseteq U$ that guarantees this and we can then set $S := S' \cup S^+$. The time for computing S^+ is $O(t \log \log \log n + \log \log n \cdot \log \log \log n)$. The overall time for computing the set S is therefore also $O(t \log \log \log n + \log \log n \cdot \log \log \log n)$ and we need $O((m + n) \cdot n^\varepsilon)$ memory for some (arbitrarily small) constant $\varepsilon > 0$ in order to compute the set S^+ . \square

5.3 Implementation of Our Algorithm in the MPC Model

Before discussing the implementation of our facility location and k -means algorithm in the MPC model, we give a lemma that provides a standard building block that we use in several of the steps.

Lemma 5.10. *Assume that we have n elements, where each element has some label and some value. There is an $O(1)$ -round fully scalable MPC algorithm to group the elements by label and aggregate (e.g., compute the sum/min/max of the values) over each group. Each machine that initially stores an element of some given label in the end learns the aggregate value for that label.*

Proof. By using the algorithm of [\[GSZ11\]](#), we can first sort the elements according to their label. Using one global aggregation tree, it is then straightforward to aggregate over the elements of each group. If each machine has a local memory of n^σ bits for some constant σ , one can build an aggregation tree of fan-out $n^{\Theta\sigma}$ and therefore of $O(1)$ height. The result of the aggregation can be distributed to the machines that initially hold the elements by using the same tree (communicating in the reverse order). \square

We first focus on the implementation of [Algorithm 1](#), i.e., of the facility location algorithm. Let us therefore assume that we are given a set of facilities $\mathcal{F} \subset \mathbb{R}^d$, a set of clients $\mathcal{C} \subset \mathbb{R}^d$, and a fixed opening cost $\lambda > 0$ per facility. As a first step, we take the set of points $\mathcal{F} \cup \mathcal{C} \subset \mathbb{R}^d$ and apply [Lemma 5.4](#) to obtain a graph $G = (\mathcal{F} \cup \mathcal{C}, E)$ with edge weights $w(e) > 0$ such that G satisfies the conditions of [Lemma 5.4](#). In the following, we will use Γ as the constant that is guaranteed by [Lemma 5.4](#).

In order to implement [Algorithm 1](#) in the MPC model, we have to approximate the size of balls around points in \mathbb{R}^d . We first introduce some notation. As already defined in [Section 2](#), we use $B_X(x, r) := \{y \in X : \text{dist}(x, y) \leq r\}$ to denote the ball of radius r around $x \in \mathbb{R}^d$, restricted to the points in X . We further define corresponding neighborhoods in the graph G that we use to approximate (P, dist) . For $x, y \in P$, we use $d_G^{(2)}(x, y)$ to denote the length of the shortest path connecting x and y and consisting of at most 2 hops. Recall that by the definition of G (cf. [Lemma 5.4](#)), for any two point $x, y \in X$, we have $d_G^{(2)}(x, y) \leq \text{dist}(x, y)$. We define

$$B_X^+(x, r) := \left\{ y \in X : d_G^{(2)}(x, y) \leq r \right\}.$$

Note that by Lemma 5.4, for all x , r , and X , we immediately get

$$B_X^+ \left(x, \frac{r}{\Gamma} \right) \subseteq B_X(x, r) \subseteq B_X^+(x, r). \quad (30)$$

We now have everything that we need to describe the implementations of the individual steps of Algorithm 1.

Lemma 5.11. *Step I of Algorithm 1 can be implemented in $O(1)$ MPC rounds with $C_R = 9\Gamma$.*

Proof. For convenience, we define a function $\phi(r)$ as

$$\phi(r) := \sum_{c \in \mathcal{C}} [r^2 - \text{cost}(c, f)]^+ = \sum_{c \in B_{\mathcal{C}}(f, r)} (r^2 - \text{cost}(c, f)).$$

Recall that for each facility $f \in \mathcal{F}$, r_f is defined as $r_f := \min_{r \geq 0} \left\{ \sum_{c \in \mathcal{C}} [r^2 - \text{cost}(c, f)]^+ \geq \lambda \right\}$ and thus r_f is defined as the smallest r for which $\phi(r) \geq \lambda$. Our goal is to compute an estimate \hat{r}_f such that $r_f/C_R \leq \hat{r}_f \leq r_f$ (where $C_R = 9\Gamma$).

First note that for every $r \geq 0$, we have

$$\phi(r) = \sum_{c \in B_{\mathcal{C}}(f, r)} (r^2 - \text{cost}(c, f)) \leq |B_{\mathcal{C}}(f, r)| \cdot r^2 \leq |B_{\mathcal{C}}^+(f, r)| \cdot r^2. \quad (31)$$

The last inequality follows from (30). We define r'_f to be the supremum over the values r for which $|B_{\mathcal{C}}^+(f, r)| \cdot r^2 \leq \lambda$. Note that by (31) this implies that every $\xi > 0$, we have $\phi(r'_f - \xi) < \lambda$ and thus $r'_f \leq r_f$.

For any $r \geq 0$, we can further lower bound $\phi(r)$ as follows

$$\phi(r) \geq \sum_{c \in B_{\mathcal{C}}(f, r/2)} (r^2 - \text{cost}(c, f)) \geq |B_{\mathcal{C}}(f, r/2)| \cdot \left(\frac{r}{2} \right)^2 \geq |B_{\mathcal{C}}^+(f, r/(2\Gamma))| \cdot \left(\frac{r}{2\Gamma} \right)^2. \quad (32)$$

The second inequality follows because for all $c \in B_{\mathcal{C}}(f, r/2)$, we have $\text{cost}(c, f) \leq (r/2)^2$ and because $r^2 - (r/2)^2 > (r/2)^2$. The last inequality follows from (30) and from the fact that $\Gamma \geq 1$. The definition of r'_f together with the fact that $\phi(2r_f/3) < \lambda$ and (32) imply that $r'_f \geq r_f/(3\Gamma)$. We therefore know that $r_f/(3\Gamma) \leq r'_f \leq r_f$. The radius r'_f thus satisfies the requirements for the estimate \hat{r}_f (with $C_R = 3\Gamma$). We can however not efficiently compute r'_f exactly for every facility f .

We can however compute a value \hat{r}_f for which $r'_f/3 \leq \hat{r}_f \leq r'_f$ as follows. For every integer $\ell \geq 0$ and each facility i , we define $b_{\ell}^+(f) := |B_{\mathcal{C}}^+(f, 2^{\ell})|$ to be the size of the number of clients c within 2-hop distance at most 2^{ℓ} in G . For a fixed ℓ and a constant $\varepsilon > 0$, we can use Lemma 5.6 to compute a value $\tilde{b}_{\ell}^+(f)$ with $b_{\ell}^+(f) \leq \tilde{b}_{\ell}^+(f) \leq 3/2 \cdot b_{\ell}^+(f)$ for every facility f in $O(1)$ rounds in the sublinear-memory MPC model. And since we assume that the maximum distance between any two points is polynomial in n , by increasing the global memory by a factor $O(\log n)$, we can compute $\tilde{b}_{\ell}^+(f)$ for all f and all integers ℓ in parallel in $O(1)$ sublinear-space MPC rounds. We define

$$\hat{\ell}_f := \max_{\ell \in \mathbb{N}_0} \left\{ \tilde{b}_{\ell}^+(f) \cdot 4^{\ell} \leq \lambda \right\} \quad \text{and} \quad \hat{r}_f := 2^{\hat{\ell}_f}.$$

Note that $\hat{r}_f \leq r'_f$ because for any $\ell > \log_2(r'_f)$, we have

$$\tilde{b}_{\ell}^+(f) \cdot 4^{\ell} \geq b_{\ell}^+(f) \cdot 4^{\ell} = |B_{\mathcal{C}}(f, 2^{\ell})| \cdot 4^{\ell} > \lambda.$$

The last inequality follows because $2^\ell > r'_f$ and we defined r'_f as the supremum over all r for which $|B_{\mathcal{C}}(f, r)| \cdot r^2 \leq \lambda$. To see that $\hat{r}_f \geq r'_f/3$, consider the largest ℓ for which $\tilde{b}_\ell^+(f) \cdot 4^\ell \leq \lambda$. Since $\tilde{b}_\ell^+(f) \geq b_\ell^+(f)$, we know that

$$b_\ell^+(f) \cdot 4^\ell \leq \lambda.$$

By the definition of r'_f , this implies $r'_f \leq 2^\ell$.

On the other hand, from the definition of $\tilde{b}_\ell^+(f)$, we also have:

$$\tilde{b}_{\ell+1}^+(f) \cdot 4^{\ell+1} > \lambda.$$

Using the fact that $\tilde{b}_{\ell+1}^+(f) \leq \frac{3}{2}b_{\ell+1}^+(f)$, we get:

$$\frac{3}{2}b_{\ell+1}^+(f) \cdot 4^{\ell+1} > \lambda \quad \Rightarrow \quad b_{\ell+1}^+(f) \cdot 4^{\ell+1} > \frac{2}{3}\lambda.$$

By the definition of r'_f , this implies that $r'_f \geq 2^{\ell+1}/3$. Since $\hat{r}_f = 2^\ell$, we conclude:

$$\hat{r}_f \geq \frac{r'_f}{3}.$$

Thus, we obtain:

$$\frac{r'_f}{3} \leq \hat{r}_f \leq r'_f.$$

Combining this with our previous bound $r_f/(3\Gamma) \leq r'_f \leq r_f$, we conclude:

$$\frac{r_f}{9\Gamma} \leq \hat{r}_f \leq r_f.$$

Hence, \hat{r}_f satisfies the required approximation guarantee with $C_R = 9\Gamma$.

Finally, since $\tilde{b}_\ell^+(f)$ values can be computed in $O(1)$ rounds and the selection of $\hat{\ell}_i$ requires only a maximum search over $O(\log n)$ values, the entire procedure runs in $O(1)$ MPC rounds. \square

Now, we move towards the second step of [Algorithm 1](#) and prove that it can be implemented in $O(1)$ MPC rounds. Specifically, we show that the computation of the initial dual values $\alpha_{c,0}$ for each client j can be performed efficiently in parallel.

Lemma 5.12. *Step II of [Algorithm 1](#) can be implemented in $O(1)$ MPC rounds with constants $C_D^+ = 8\Gamma^4$ and $C_D^- = 2\Gamma^2$.*

Proof. Recall that for each client $c \in \mathcal{C}$, the value α_c^* is defined as $\alpha_c^* = \min_{f \in \mathcal{F}} \max\{r_f^2, \text{cost}(c, f)\}$. This represents the smallest dual value for which client c can contribute to the opening of a facility. Our goal is to compute an estimate $\alpha_{c,0}$ such that

$$\frac{\min_{f \in \mathcal{F}} \max\{r_f^2, \text{cost}(c, f)\}}{C_D^+} = \frac{\alpha_c^*}{C_D^+} \leq \alpha_{c,0} \leq \frac{\alpha_c^*}{C_D^-} = \frac{\min_{f \in \mathcal{F}} \max\{r_f^2, \text{cost}(c, f)\}}{C_D^-},$$

where the constants C_D^+ and C_D^- ensure the necessary approximation guarantees.

For convenience, for a ball of radius 2^ℓ around a given client $c \in \mathcal{C}$, we define the function $\rho(\ell)$

as

$$\rho(\ell) = \min_{f \in B_{\mathcal{F}}(c, 2^\ell)} \{r_f\}.$$

Here, $\ell \in [\log n]$ because, as discussed in [Section 2](#), the distance between any two input points x and y satisfies $1 \leq \text{dist}(x, y) \leq \text{poly}(n)$, implying that the number of relevant distance scales is at most $O(\log n)$.

For each client $c \in \mathcal{C}$, we define a value $\bar{\alpha}_c$ as

$$\bar{\alpha}_c = \min_{\ell} \max \left\{ \rho(\ell)^2, 4^\ell \right\}.$$

We now show that this definition ensures the bound $\alpha_c^* \leq \bar{\alpha}_c \leq 4\alpha_c^*$. To that end, observe that for each facility $f \in \mathcal{F}$ at distance $d(c, f)$ from client c , the facility first appears in the ball $B_{\mathcal{F}}(c, 2^\ell)$ with $\ell = \ell_f := \lceil \log_2 d(c, f) \rceil$. Since all balls with larger radius also contain f , and the minimization in the definition of $\bar{\alpha}_c$ is taken over all such ℓ , the contribution of facility f is accounted for at index ℓ_f .

Thus, for every facility f , we have:

$$\max\{r_f^2, \text{cost}(c, f)\} \leq \max\{r_f^2, 4^{\ell_f}\},$$

and therefore:

$$\alpha_c^* = \min_{f \in \mathcal{F}} \max\{r_f^2, \text{cost}(c, f)\} \leq \min_{\ell} \max\{\rho(\ell)^2, 4^\ell\} = \bar{\alpha}_c.$$

To obtain the upper bound, consider any facility f^* that achieves the minimum in the definition of α_c^* , i.e.,

$$\alpha_c^* = \max\{r_{f^*}^2, \text{cost}(c, f^*)\}.$$

Let $\ell^* = \lceil \log_2 d(c, f^*) \rceil$. Then $f^* \in B_{\mathcal{F}}(c, 2^{\ell^*})$ and hence:

$$\rho(\ell^*) \leq r_{f^*}, \quad \text{so} \quad \bar{\alpha}_c \leq \max\{r_{f^*}^2, 4^{\ell^*}\}.$$

Using $\text{cost}(c, f^*) = d(c, f^*)^2 \leq 4^{\ell^*}$, it follows that:

$$\begin{aligned} \bar{\alpha}_c &\leq \max\{r_{f^*}^2, 4^{\ell^*}\} \leq 4 \cdot \max\{r_{f^*}^2, d(c, f^*)^2\} \\ &= 4\alpha_c^*. \end{aligned}$$

Combining both bounds yields:

$$\alpha_c^* \leq \bar{\alpha}_c \leq 4\alpha_c^*. \tag{33}$$

We further define the function $\rho^+(\ell)$ as follows:

$$\rho^+(\ell) = \min_{f \in B_{\mathcal{F}}^+(c, 2^\ell)} \{r_f\}.$$

Moreover, we define the value α_c^+ as

$$\alpha_c^+ = \min_{\ell} \max \left\{ \rho^+(\ell)^2, 4^\ell \right\}.$$

We now show that this definition ensures the bound $\alpha_c^*/\Gamma^2 \leq \alpha_c^+ \leq 4\alpha_c^*$. Considering the definitions of α_c^+ and $\bar{\alpha}_c$, and noting that both are defined as a minimum over ℓ of the maximum between the squared radius and 4^ℓ , we apply the containment property from (30). Since $B_{\mathcal{F}}(c, 2^\ell) \subseteq B_{\mathcal{F}}^+(c, 2^\ell)$, and the minimization across all ℓ naturally discards large-radius facilities when possible, it follows that:

$$\alpha_c^+ \leq \bar{\alpha}_c.$$

Using the previously established bound $\bar{\alpha}_c \leq 4\alpha_c^*$ from (33), we obtain:

$$\alpha_c^+ \leq 4\alpha_c^*.$$

To establish a lower bound on α_c^+ , we expand its definition as follows:

$$\begin{aligned} \alpha_c^+ &= \min_{\ell} \max \left\{ \rho^+(\ell)^2, 4^\ell \right\} \\ &= \min_{\ell} \max \left\{ \min_{f \in B_{\mathcal{F}}^+(c, 2^\ell)} r_f^2, 4^\ell \right\} \\ &= \min_f \max \left\{ r_f^2, 4^{\ell_f^+} \right\}, \end{aligned}$$

where $\ell_f^+ := \min\{\ell \mid f \in B_{\mathcal{F}}^+(c, 2^\ell)\}$.

From the definition of B^+ and the underlying graph structure, we have the guarantee:

$$\frac{\text{dist}(c, f)}{\Gamma} \leq 2^{\ell_f^+}.$$

Since $\text{cost}(c, f) = \text{dist}(c, f)^2$, we conclude:

$$\frac{\text{cost}(c, f)}{\Gamma^2} \leq 4^{\ell_f^+}.$$

Now, combining the above with the definition of α_c^* , we obtain:

$$\frac{\alpha_c^*}{\Gamma^2} \leq \alpha_c^+.$$

Combining both bounds yields:

$$\alpha_c^*/\Gamma^2 \leq \alpha_c^+ \leq 4\alpha_c^*. \tag{34}$$

Finally, we set

$$\alpha_{c,0} := \frac{\alpha_c^+}{8\Gamma^2},$$

which yields the desired bounds:

$$\frac{\alpha_c^*}{8\Gamma^4} \leq \alpha_{c,0} \leq \frac{\alpha_c^*}{2\Gamma^2}.$$

This concludes the proof. □

Now, we move towards the third step of [Algorithm 1](#) and prove that it can be implemented in $O(1)$ MPC rounds. Specifically, we show that finding the set of problematic clients can be performed efficiently in parallel.

Lemma 5.13. *Step III of [Algorithm 1](#) can be implemented in $O(1)$ MPC rounds with constants $\gamma_1 = 4\Gamma^4$, $\gamma_2 = 9\Gamma^4$, and $Q = 8\Gamma^4$.*

Proof. Recall that a client $c \in \mathcal{C}$ is marked as problematic if there exists another client $c' \in B_{\mathcal{C}}(c, \sqrt{\alpha_c^*})$ such that

$$\alpha_{c',0} \leq \frac{\alpha_{c,0}}{Q}.$$

However, since the algorithm only operates on approximations $\alpha_{c,0}$ of the true values α_c^* , where

$$\frac{\alpha_c^*}{8\Gamma^2} \leq \alpha_{c,0} \leq \frac{\alpha_c^*}{2},$$

which means we conservatively search within $B_{\mathcal{C}}(c, \sqrt{\alpha_{c,0}}) \in B_{\mathcal{C}}(c, \sqrt{\alpha_c^*/2})$.

Moreover, the algorithm uses the approximate neighborhood $B_{\mathcal{C}}^+$ (computed from the power graph) instead of the exact ball $B_{\mathcal{C}}$. From [\(30\)](#), we know that

$$B_{\mathcal{C}}(j, \sqrt{\alpha_c^*/2}) \subseteq B_{\mathcal{C}}^+(j, \sqrt{\alpha_c^*/2}),$$

which ensures that any client c' that could make c problematic is still included in the search region.

By [Lemma 5.4](#), the power graph guarantees that for all clients x, y ,

$$\text{dist}(x, y) \leq \Gamma \cdot d_G^{(2)}(x, y),$$

so any client c' in $B_{\mathcal{C}}^+(c, \sqrt{\alpha_c^*/2})$ lies within distance at most $\Gamma \cdot \sqrt{\alpha_c^*/2}$ of c , which is within distance at most $\gamma_1 \cdot \sqrt{\alpha_j^*}$.

To show that the connection cost from a problematic client c to a facility f' remains bounded after freezing α_c , we assume that c previously contributed to f' and is now frozen due to the presence of a nearby client c' such that $\alpha_{c',0} \leq \alpha_{c,0}/Q$.

Let c' be the witness for c being problematic, and suppose that in the final solution, c connects to facility f' (which c' contributes to). We aim to bound $\text{cost}(c, f')$ in terms of $\alpha_{c,0}$.

Since c' lies within distance at most $\sqrt{\alpha_c^*/2}$ of c , and we only have access to approximate distances via [Lemma 5.4](#), the actual distance is bounded by:

$$\text{dist}(c, c') \leq \Gamma \cdot d_G^{(2)}(c, c') \leq \Gamma \cdot \sqrt{\alpha_c^*/2}.$$

Moreover, since c' contributes to facility f' , we have:

$$\text{dist}(c', f') \leq \sqrt{\alpha_{c',1}} \leq \sqrt{C_A \cdot \alpha_{c',0}} \leq \sqrt{C_A \cdot \frac{\alpha_{c,0}}{Q}}.$$

Using the triangle inequality, we can now bound $\text{dist}(c, f')$:

$$\begin{aligned} \text{dist}(c, f') &\leq \text{dist}(c, c') + \text{dist}(c', f') \\ &\leq \Gamma \cdot \sqrt{\frac{\alpha_c^*}{2}} + \sqrt{C_A \cdot \frac{\alpha_{c,0}}{Q}}. \end{aligned}$$

Using the bound $\alpha_c^* \leq 8\Gamma^2 \cdot \alpha_{c,0}$ from Step II, we obtain:

$$\begin{aligned} \text{dist}(c, f') &\leq \Gamma \cdot \sqrt{4\Gamma^2 \cdot \alpha_{c,0}} + \sqrt{\frac{C_A}{Q}} \cdot \sqrt{\alpha_{c,0}} \\ &= 2\Gamma^2 \cdot \sqrt{\alpha_{c,0}} + \sqrt{\frac{C_A}{Q}} \cdot \sqrt{\alpha_{c,0}} \\ &= \left(2\Gamma^2 + \sqrt{\frac{C_A}{Q}} \right) \cdot \sqrt{\alpha_{c,0}}. \end{aligned}$$

Squaring both sides, we get:

$$\text{cost}(c, f') = \text{dist}^2(c, f') \leq \left(2\Gamma^2 + \sqrt{\frac{C_A}{Q}} \right)^2 \cdot \alpha_{c,0}.$$

Thus, the connection cost of a problematic client c remains bounded by a constant factor of $\alpha_{c,0}$, where:

$$\gamma_2 := \left(2\Gamma^2 + \sqrt{\frac{C_A}{Q}} \right)^2.$$

Given the requirement that $Q \geq C_D^+/C_D^- = 4\Gamma^2$, we choose $C_A = \Gamma^4 \cdot Q$, which implies that $C_A/Q \leq \Gamma^4$. As a result, we obtain $\gamma_2 = 9\Gamma^4$.

Finally, note that B_c^+ corresponds to the two-hop neighborhood in the graph G and can be computed in $O(1)$ MPC rounds. Therefore, the entire implementation of Step III requires only constant rounds. \square

Lemma 5.14. *Step IV of Algorithm 1 can be implemented in $O(1)$ MPC rounds, and it computes a set $S \subseteq \mathcal{F}$ of facilities satisfying the following properties:*

- (a) *S includes all fully paid facilities in \mathcal{F} .*
- (b) *Every facility $i \in S$ is κ -approximately paid, where $\kappa = \Gamma^2$.*

Proof. Recall that a facility i is called *paid* if the total contribution from clients in its neighborhood satisfies the condition:

$$\sum_{c \in B_c(f, r_f)} (r_f^2 - \text{cost}(c, f)) = \lambda,$$

where λ is the opening cost of facility f .

To determine whether a facility is paid, we need to compute this sum by considering all clients within the ball $B_C(f, r_f)$. Specifically, we rewrite the summation as:

$$\sum_{c \in B_C(f, r_f)} (r_f^2 - \text{cost}(c, f)) = |B_C(f, r_f)| \cdot r_f^2 - \sum_{c \in B_C(f, r_f)} \text{cost}(c, f).$$

Thus, we only need to compute $|B_C(f, r_f)|$, the number of clients in the ball, and the sum of distances $\text{cost}(c, f)$ for all $c \in B_C(f, r_f)$.

However, we do not have direct access to $B_C(f, r_f)$. Instead, following (30), we approximate it using $B_C^+(f, r_f)$, which can be accessed in $O(1)$ rounds. The issue with using $B_C^+(f, r_f)$ is that it might include additional clients whose true Euclidean distances from f fall within the range $[r_f, \Gamma \cdot r_f]$. These additional clients contribute negatively to the sum since $r_f^2 - \text{cost}(c, f)$ becomes negative for them, potentially misclassifying a truly paid facility as unpaid.

To mitigate this issue, when summing over all clients in $B_C^+(f, r_f)$, we only consider those clients whose distances satisfy $\text{cost}(c, f) \leq r_f^2$, ensuring that only valid contributions are included.

Additionally, as in previous steps, we do not work directly with r_f , but rather with its approximation \hat{r}_f , which satisfies:

$$\frac{r_f}{9\Gamma} \leq \hat{r}_f \leq r_f.$$

This means that we must evaluate the condition using \hat{r}_f instead of r_f . Since we only have a constant-factor approximation of r_f , we must adjust the threshold κ to be Γ^2 . This ensures that facilities that are actually paid remain paid in our approximation, though some additional facilities that are not fully paid may also be included.

Given that accessing $B_C^+(f, r_f)$, computing its size, and summing contributions all take $O(1)$ MPC rounds, the overall computation remains efficient. \square

Lemma 5.15. *Step V of Algorithm 1 can be implemented in $O(1)$ MPC rounds. More specifically, there is an $O(1)$ -round MPC algorithm that computes an unweighted graph H_{base} with at most $n^{1+\varepsilon}$ edges (for $\varepsilon > 0$ an arbitrarily small constant) such that the graph H_{base}^2 satisfies the requirements for graph H' in Algorithm 1 with the constants $C_{H,1}$ and $C_{H,2}$ given by*

$$C_{H,1} = 4 \cdot C_R^4 \cdot \zeta^2, \quad C_{H,2} = 12 \cdot \sqrt{\kappa\rho} \cdot C_R^3 \cdot \zeta^2.$$

where ρ and ζ are the constants from Lemmas 4.3 and 4.4.

Proof. Recall that the dependency graph $H = (\mathcal{S}, E_H)$ on the set \mathcal{S} is defined as follows:

$$E_H := \left\{ \{f, f'\} \in \binom{\mathcal{S}}{2} \mid \exists c \in \mathcal{C} \text{ such that } \kappa\alpha_{c,1} > \max\{\text{cost}(c, f), \text{cost}(c, f')\} \right\}.$$

We cannot directly construct the graph H . We can however construct an alternative graph $H' = (\mathcal{S}, E_{H'})$, ensuring that $E_H \subseteq E_{H'}$. To understand the conditions that $E_{H'}$ must satisfy, let us first understand some basic properties of H . Consider some edge $\{f, f'\}$ of H . W.l.o.g., assume that $r_f \leq r_{f'}$. First, by Lemma 4.3 we know that $\alpha_{c,1} \leq \rho \cdot r_f^2$, where ρ is the constant specified in the statement of Lemma 4.3. Because we also know that $\alpha_{c,1} \geq \kappa \max\{\text{cost}(c, f), \text{cost}(c, f')\}$, the distance between f and f' can therefore be bounded as

$$\text{dist}(f, f') \leq \text{dist}(f, c) + \text{dist}(c, f') \leq 2 \cdot \sqrt{\kappa \cdot \alpha_{c,1}} \leq 2 \cdot \sqrt{\kappa \cdot \rho} \cdot r_f. \quad (35)$$

From Lemma 4.4, we further know that $r_{f'} \leq \zeta \cdot r_f$ for the constant ζ that is defined in the statement of Lemma 4.4. The algorithm does not know the exact value of r_f for a facility f . In Step I, it however computes an estimate \hat{r}_f such that $r_f/C_R \leq \hat{r}_f \leq r_f$. For f and f' , we therefore know that

$$\max\{\hat{r}_f, \hat{r}_{f'}\} \leq C_R \cdot \zeta \cdot \min\{\hat{r}_f, \hat{r}_{f'}\}. \quad (36)$$

If we construct the graph H' such that any two facilities $f, f' \in \mathcal{S}$ for which (35) and (36) hold are connected by an edge in H' , then we definitely guarantee that $E_H \subseteq E_{H'}$ (i.e., H is a subgraph of H'). We want to achieve this while not adding any edges $\{f, f'\}$ to H' for which $\text{dist}(f, f') = \omega(\min\{r_f, r_{f'}\})$ or for which r_f and $r_{f'}$ are not within a constant factor. Moreover since H and thus also H' might be a dense graph, we cannot explicitly compute it. We will construct a sufficiently sparse graph H_{base} such that H_{base}^2 satisfies the conditions of H' .

As a first step, we define $O(\log n)$ radius classes and assign the facilities in \mathcal{S} to those classes. Each facility $f \in \mathcal{S}$ is assigned to $O(1)$ of the classes. For $i = 0, 1, \dots, O(\log n)$, we define

$$\mathcal{S}_i := \{f \in \mathcal{S} : 2^i \leq \hat{r}_f \leq 2^{i+1} \cdot C_R \cdot \zeta\}.$$

Note that by (36), for any two facilities f, f' that are connected by an edge in H , there is at least one radius class \mathcal{S}_i such that f and f' are both in \mathcal{S}_i . For each set \mathcal{S}_i , we first compute a weighted graph G_i by applying Lemma 5.4. The lemma guarantees that G_i can be computed in $O(1)$ time in the MPC model, that G_i has at most $|\mathcal{S}_i|^{1+\varepsilon} \leq n^{1+\varepsilon}$ edges, and that any two facilities $f, f' \in \mathcal{S}_i$ are connected by a path of hop-length at most 2 and total weight at most $\text{dist}(f, f')$ in G_i . In addition, for any f, f' , the weighted distance in G_i is at least $\text{dist}(f, f')/\Gamma$. Based on G_i , we now define an unweighted graph $H_i = (\mathcal{S}_i, E_{H_i})$, where the edge set E_{H_i} is defined as

$$E_{H_i} := \left\{ \{f, f'\} \in \binom{\mathcal{S}_i}{2} : G_i \text{ contains an edge } \{f, f'\} \text{ of weight } \leq 2^{i+2} \cdot \sqrt{\kappa\rho} \cdot C_R^2 \cdot \zeta \right\}.$$

We next verify that if $f, f' \in \mathcal{S}_i$ and $\{f, f'\} \in E_H$, then H_i contains a path of length at most 2 between f and f' . We thus have to show that $f, f' \in \mathcal{S}_i$ and (35) holds, then H_i contains a path of length at most 2 between f and f' . From (35), we have

$$\text{dist}(f, f') \leq 2\sqrt{\kappa\rho} \cdot r_f \leq 2\sqrt{\kappa\rho} C_R \cdot \hat{r}_f \leq 2\sqrt{\kappa\rho} \cdot C_R \cdot 2^{i+1} \cdot C_R \cdot \zeta.$$

The last inequality follows from $f \in \mathcal{S}_i$. We therefore know that G_i contains a path of hop length 2 and total weight at most $2^{i+2} \cdot \sqrt{\kappa\rho} \cdot C_R^2 \cdot \zeta$. Consequently, H_i contains a path of length at most 2 between f and f' .

We now define our base graph $H_{base} = (\mathcal{S}, E_{base})$ as $E_{base} := E_{H_0} \cup E_{H_1} \cup \dots$. From the above observation and the fact that for any edge $\{f, f'\}$ of H , there is some i for which $f, f' \in \mathcal{S}_i$, H_{base} definitely contains a path of length at most 2 between f and f' . The graph $H' := H_{base}^2$ therefore is a supergraph of H as desired.

It remains to show that for any edge $\{f, f'\}$ in H' , we have

$$\max\{r_f, r_{f'}\} \leq C_{H,1} \cdot \min\{r_f, r_{f'}\} \quad \text{and} \quad \text{dist}(f, f') \leq C_{H,2} \cdot \min\{r_f, r_{f'}\}.$$

Note that for any edge $\{f, f'\}$ in H_{base} , the values of \hat{r}_f and $\hat{r}_{f'}$ are within a factor at most $2^{i+1} \cdot C_R \cdot \zeta$. The values of r_f and $r_{f'}$ are then within a factor of at most $2 \cdot C_R^2 \cdot \zeta$. Consequently, for any edge $\{f, f'\}$ of H' , we r_f and $r_{f'}$ are within a factor

$$C_{H,1} = 4 \cdot C_R^4 \cdot \zeta^2.$$

Let us now also upper bound the distance between any two facilities f, f' that are connected by an edge in H' (and thus by a path of length at most 2 in H_{base}). We first consider two facilities f, f' that are connected by an edge in H_{base} . Assume that this edge was added as part of the graph H_i , i.e., both f and f' are in \mathcal{S}_i . We know that f and f' are connected by an edge of weight at most $2^{i+2} \cdot \sqrt{\kappa\rho} \cdot C_R^2 \cdot \zeta$ in G_i . From the properties of G_i (cf. [Lemma 5.4](#)), we know that

$$\begin{aligned} \text{dist}(f, f') &\leq \Gamma \cdot d_{G_i}(f, f') \\ &\leq \Gamma \cdot 2^{i+2} \cdot \sqrt{\kappa\rho} \cdot C_R^2 \cdot \zeta \\ &\leq 4 \cdot \sqrt{\kappa\rho} \cdot C_R^2 \cdot \zeta \cdot \min\{\hat{r}_f, \hat{r}_{f'}\} \\ &\leq 4 \cdot \sqrt{\kappa\rho} \cdot C_R^2 \cdot \zeta \cdot \min\{r_f, r_{f'}\}. \end{aligned}$$

Now consider a path f, f', f'' of length 2 in H_{base} . Assume that $\hat{r}_f \leq \hat{r}_{f''}$. The above calculation implies that

$$\begin{aligned} \text{dist}(f, f'') &\leq 4 \cdot \sqrt{\kappa\rho} \cdot C_R^2 \cdot \zeta \cdot \min\{\hat{r}_f, \hat{r}_{f'}\} + 4 \cdot \sqrt{\kappa\rho} \cdot C_R^2 \cdot \zeta \cdot \min\{\hat{r}_{f'}, \hat{r}_{f''}\} \\ &\leq 4 \cdot \sqrt{\kappa\rho} \cdot C_R^2 \cdot \zeta \cdot \hat{r}_f + 4 \cdot \sqrt{\kappa\rho} \cdot C_R^2 \cdot \zeta \cdot \hat{r}_{f'} \\ &\leq 4 \cdot \sqrt{\kappa\rho} \cdot C_R^2 \cdot \zeta \cdot (1 + 2 \cdot C_R \cdot \zeta) \cdot \hat{r}_f \\ &\leq 12 \cdot \sqrt{\kappa\rho} \cdot C_R^3 \cdot \zeta^2 \cdot \min\{r_f, r_{f''}\}. \end{aligned}$$

The last inequality in particular uses that for all f , $\hat{r}_f \leq r_f$ and thus $\hat{r}_f = \min\{\hat{r}_f, \hat{r}_{f''}\} \leq \min\{r_f, r_{f''}\}$. \square

Lemma 5.16. *Step VI of [Algorithm 1](#) can be implemented in $\mathcal{O}(\log \log n \cdot \log \log \log n)$ MPC rounds.*

Proof. We start by finding one a facility $f_c \in \mathcal{S}$ for which $\max\{r_f^2, \text{cost}(c, f)\} = O(\alpha_{c,0})$. By [Lemma 4.2](#), such a facility must exist. In order to find it, every client c searches for the \mathcal{S} -node with minimum \hat{r}_f -value within an approximate $O(\alpha_{c,0})$ ball of sufficiently large radius. This can be done in $O(1)$ rounds by using the graph G that is provided by [Lemma 5.4](#) and by using [Lemma 5.5](#) to probe the 2-hop neighborhood in G restricted to edges of weight $O(\alpha_{c,0})$. In this way, in $O(1)$ time, every client can find its facility f_c .

We will compute the set of cluster centers $\mathcal{S}_0 \subseteq \mathcal{S}$ by using the approximate ruling set algorithm of [Lemma 5.9](#). However in order to guarantee that a large fraction of the total dual client values is within constant distance of a cluster center, we first need to compute weights for the facilities $f \in \mathcal{S}$. The weight $w(f)$ of a facility $f \in \mathcal{S}$ is defined as $w(f) := \sum_{c \in \mathcal{C}: f_c = f} \alpha_c$, i.e., $w(f)$ is the sum of the dual values of the clients that chose f as their close-by approximately paid facility. We can exactly compute those weights for all $f \in \mathcal{S}$ in $O(1)$ MPC rounds by using [Lemma 5.10](#).

By using [Lemma 5.15](#), we can assume that we have an explicitly constructed graph H_{base} that is relatively sparse ($\leq n^{1+\delta}$ edges for an arbitrarily small constant $\delta > 0$) and such that H_{base}^2 satisfies the criteria for graph H' in [Algorithm 1](#). The cluster centers \mathcal{S}_0 have to form a set of facilities that are at pairwise distance at least 4 in H and such that all except a $1/\log n$ -fraction of the node (facility) weights are within distance $O(1)$ in H' . We can enforce pairwise distance at least 4 in H by requiring pairwise distance at least 4 in the supergraph H' and thus pairwise distance at least 7 in H_{base} . By using the algorithm of [Lemma 5.9](#) with parameters $t = 6$ and $\varepsilon = 1/\log n$, we obtain a set that directly satisfies all the properties we need. We can therefore compute \mathcal{S}_0 in $O(\log \log n \cdot \log \log \log n)$ rounds in the MPC model.

In the last part of Step VI, we have to build the clusters by assigning each facility $f \in \mathcal{S}$ to the cluster of the closest facility $f_0 \in \mathcal{S}_0$. The cluster center that is at minimum distance from f in H_{base} is also at minimum distance from f in H' . We can therefore assign all facilities to

their cluster centers by running parallel BFS searches on H_{base} from all cluster centers for at most $O(\log \log \log n)$ steps (we always only forward the first search that reaches a node). This can clearly be done in $O(\log \log \log n)$ MPC rounds. \square

Lemma 5.17. *Step VII of Algorithm 1 can be implemented in $O(1)$ MPC rounds.*

Proof. The goal of Step VII is to open one facility in each cluster that approximately minimizes the total distance to the points in that cluster. Let each point be labeled according to its assigned cluster. Our objective is to compute, for each cluster, a single facility to open that best represents the cluster in terms of proximity to its members.

We proceed in two phases:

Phase 1: Computing geometric centers of clusters. We apply Lemma 5.10 to group points by cluster label and compute the arithmetic mean of the coordinate vectors of the points in each cluster in $O(1)$ MPC rounds. The result is a mean vector μ_f for each cluster C_f .

Phase 2: Selecting the representative facility. For each cluster, we now need to select a facility from that cluster which is closest (in squared Euclidean distance) to the corresponding mean vector μ_f . We again use Lemma 5.10 to group all facilities by cluster label and compute the facility minimizing the distance to μ_f in each group in $O(1)$ MPC rounds.

Therefore, the full procedure for Step VII completes in $O(1)$ MPC rounds. \square

5.4 Implementation of the k -Means Reduction in the MPC Model

In this section, we describe how to implement the 3-step k -means algorithm from Section 4.3 in the fully scalable MPC model.

Lemma 5.18. *Step (I) of the k -means algorithm described in Section 4.3 can be implemented in $O(\log \log n \cdot \log \log \log n)$ MPC rounds.*

Proof. In this step, the goal is to compute two opening costs λ_1 and λ_2 such that the facility location algorithm instantiated with λ_1 opens at most k facilities (i.e., $k_1 := |\mathcal{F}_1| \leq k$), and with λ_2 opens at least k facilities (i.e., $k_2 := |\mathcal{F}_2| \geq k$), where $\lambda_2 \leq \lambda_1 \leq 2\lambda_2$.

Following (1), we know that for sufficiently large values of $\lambda = \text{poly}(n)$, the facility location algorithm opens only a single facility, whereas for $\lambda = 1$, it opens all facilities. Therefore, there must exist some value of λ in the interval $[1, \lambda_{\max}]$ at which the number of opened facilities transitions from more than k to fewer than k .

We discretize the interval $[1, \lambda_{\max}]$ into $O(\log n)$ geometrically spaced values of λ . Since the number of opened facilities is monotonically non-increasing in λ , we are guaranteed to find two consecutive values λ_i and λ_{i+1} such that one opens fewer than k facilities and the other opens more than k . These two values therefore satisfy the condition that $\lambda_2 \leq \lambda_1 \leq 2\lambda_2$. If either value corresponds to a solution that opens exactly k facilities, we return that facility location solution as the final k -means solution and terminate.

Otherwise, we return the facility location instances corresponding to λ_1 and λ_2 as the output of Step (I). Since all facility location instances for the $O(\log n)$ candidate values of λ are executed in parallel, the total number of MPC rounds is determined by the runtime of a single facility location computation. This results in a total round complexity of $O(\log \log n \cdot \log \log \log n)$. The global memory usage increases only by a factor of $O(\log n)$ due to the parallel execution over $O(\log n)$ instances. \square

Now, we move towards the second step of the k -means algorithm explained in [Section 4.3](#) and prove that it can be implemented in $O(1)$ MPC rounds. Specifically, we show that computing the set \mathcal{F}'_2 can be performed efficiently in parallel.

Lemma 5.19. *Step (II) of the k -means algorithm described in [Section 4.3](#)—i.e., selecting a set \mathcal{F}'_2 of size k_1 containing a δ -approximate nearest neighbor from \mathcal{F}_2 for each facility in \mathcal{F}_1 —can be implemented in $O(1)$ MPC rounds.*

Proof. The objective is to compute a set $\mathcal{F}'_2 \subseteq \mathcal{F}_2$ of size k_1 such that for every facility $f \in \mathcal{F}_1$, there exists a facility $f'_2(f) \in \mathcal{F}_2$ included in \mathcal{F}'_2 that serves as a δ -approximate nearest neighbor of f . If the total number of selected facilities is less than k_1 , we complete the set \mathcal{F}'_2 by arbitrarily adding facilities from $\mathcal{F}_2 \setminus \mathcal{F}'_2$ until the desired size is reached.

To compute the approximate nearest neighbors efficiently in the MPC model, we use the graph G constructed as described in [Lemma 5.4](#). This graph is a sparse spanner, constructed via locality-sensitive hashing (LSH), where an edge (f, f') exists if the distance $\text{dist}(f, f')$ is known and sufficiently small.

We then apply [Lemma 5.5](#) with parameter $\ell = O(1)$ to each $f \in \mathcal{F}_1$ to compute a facility $f'_2(f) \in \mathcal{F}_2$ such that

$$\text{dist}(f, f'_2(f)) \leq \delta \cdot \min_{f' \in \mathcal{F}_2} \text{dist}(f, f'),$$

where δ is the approximation factor guaranteed by the graph construction. According to [Lemma 5.5](#), this step can be executed in $O(1)$ MPC rounds using $\tilde{O}(n)$ global memory.

Next, using a convergecast over the facilities in \mathcal{F}_2 , we compute the cardinality of the set \mathcal{F}'_2 . If $|\mathcal{F}'_2| < k_1$, we need to add $k_1 - |\mathcal{F}'_2|$ additional facilities from $\mathcal{F}_2 \setminus \mathcal{F}'_2$ to ensure that the final set \mathcal{F}'_2 has exactly k_1 facilities. Since these additional facilities can be selected arbitrarily, we use a greedy selection strategy that can be implemented efficiently in the MPC model.

To do so, we first sort the facilities in \mathcal{F}_2 according to a fixed total order (e.g., their IDs or spatial coordinates) in $O(1)$ MPC rounds using standard parallel sorting algorithms. The sorted facilities are then evenly assigned to the machines. Each machine has a local memory of $O(n^\sigma)$ for some constant $\sigma \in (0, 1)$, and there are at most n facilities in \mathcal{F}_2 , which guarantees that the total number of machines is $O(n^{1-\sigma})$. This induces a logical tree over the machines with fan-out n^σ and depth $O(1/\sigma) = O(1)$.

In the first phase (bottom-up), we perform a convergecast operation along this tree to compute, for each internal node, the number of unselected facilities stored in its subtree. Since the tree has constant height, this aggregation step requires only $O(1)$ MPC rounds.

In the second phase (top-down), we traverse the tree in a greedy fashion to select the required number of additional facilities. The root node begins with the total demand of $k_1 - |\mathcal{F}'_2|$ and distributes this demand among its children proportionally to the number of unselected facilities in each subtree (or arbitrarily, if selection is unconstrained). This process is recursively continued by each internal node down to the leaves. When a leaf is reached, the machine locally selects the requested number of facilities from its memory.

Because the tree has constant height and each level of communication can be executed in $O(1)$ MPC rounds, the entire padding step also completes in $O(1)$ rounds. At the end of this process, the set \mathcal{F}'_2 contains exactly k_1 facilities, as required. \square

We now consider Step (III) of the k -means algorithm described in [Section 4.3](#). In this step, the final set of k centers is constructed by sampling from the facility sets \mathcal{F}_1 , \mathcal{F}'_2 , and $\mathcal{F}_2 \setminus \mathcal{F}'_2$, according to a convex combination determined by $a := \frac{k_2 - k}{k_2 - k_1}$ and $b := \frac{k - k_1}{k_2 - k_1} = 1 - a$. We show that this step can be implemented efficiently in the MPC model.

Lemma 5.20. *Step (III) of the k -means algorithm described in Section 4.3 can be implemented in $\mathcal{O}(1)$ MPC rounds.*

Proof. The goal of this step is to sample a final set Z of k facilities that defines the output clustering. This is done in two stages:

- (A) With probability $a := \frac{k_2 - k}{k_2 - k_1}$, the set \mathcal{F}_1 is selected (i.e., all facilities in \mathcal{F}_1 are added to Z). Otherwise, with probability $b := 1 - a$, the set \mathcal{F}'_2 is selected and added to Z .
- (B) Additionally, $k - k_1$ facilities are sampled uniformly at random from $\mathcal{F}_2 \setminus \mathcal{F}'_2$ and added to Z .

Both of these sampling steps can be implemented in constant MPC rounds as follows:

Step (A): Sampling either \mathcal{F}_1 or \mathcal{F}'_2 . Since the sampling decision is based on a single global coin toss (deciding whether to choose \mathcal{F}_1 or \mathcal{F}'_2), we can broadcast the outcome of this coin toss to all machines in $\mathcal{O}(1)$ rounds. The selected set is then included in Z by every machine that holds elements of it.

Step (B): Uniform Sampling of $k - k_1$ Facilities from $\mathcal{F}_2 \setminus \mathcal{F}'_2$. Unlike Step (II), where additional facilities were added arbitrarily, here we must sample exactly $k - k_1$ facilities uniformly at random from $\mathcal{F}_2 \setminus \mathcal{F}'_2$.

To do this efficiently in MPC, we distribute the elements of $\mathcal{F}_2 \setminus \mathcal{F}'_2$ across machines and organize these machines into a virtual tree of height $\mathcal{O}(1)$ and fan-out n^σ , where $\sigma \in (0, 1)$ is determined by the available local memory $\mathcal{O}(n^\sigma)$. The sampling process proceeds in two phases:

- **Bottom-Up (Convergecast Phase).** Each machine computes the number of unmarked facilities it stores locally. This count is aggregated up the tree using a convergecast to compute, for each internal node, the total number of facilities in its subtree. This requires only $\mathcal{O}(1)$ rounds because the tree has constant height.
- **Top-Down (Sampling Phase).** Once the subtree sizes are known, we begin the sampling process at the root. The root node samples how many of the $k - k_1$ points should be taken from each of its child subtrees, proportional to the size of each subtree. These target counts are passed recursively down the tree. At the leaves (i.e., the machines storing the facilities), each machine samples the required number of facilities uniformly at random from its local memory.

This two-phase sampling procedure ensures that the final $k - k_1$ facilities are drawn uniformly at random from $\mathcal{F}_2 \setminus \mathcal{F}'_2$ and completes in $\mathcal{O}(1)$ MPC rounds due to the bounded tree height and constant-round communication per level.

Final Client Assignments. Once the full set Z of k centers has been determined, each client c must be connected to a facility $\phi(c) \in Z$ according to the following rules:

- If $f_c^{(2)} \in \mathcal{F}'_2$ and \mathcal{F}_1 is chosen in Step (A), then $\phi(c) := f_c^{(1)}$.
- If $f_c^{(2)} \in \mathcal{F}'_2$ and \mathcal{F}'_2 is chosen in Step (A), then $\phi(c) := f_c^{(2)}$.
- If $f_c^{(2)} \in \mathcal{F}_2 \setminus \mathcal{F}'_2$:
 - If $f_c^{(2)}$ was among the $k - k_1$ sampled facilities in Step (B), then $\phi(c) := f_c^{(2)}$.
 - If not, and \mathcal{F}_1 was chosen in Step (A), then $\phi(c) := f_c^{(1)}$.
 - Otherwise, set $\phi(c) := f'_2(f_c^{(1)})$.

All of this logic can be implemented locally using previously computed mappings (from Steps I and II) and a broadcasted sampling outcome. Therefore, this final step also requires only $\mathcal{O}(1)$ MPC rounds. \square

Lemma 5.21. *By repeating the randomized steps of the k -means algorithm explained in [Section 4.3](#) independently $O(\log n)$ times and selecting the solution with the minimum total connection cost among these repetitions, we can ensure that with high probability, the total cost of the solution is within a constant factor K of the optimal cost.*

Proof. From [Theorem 4.10](#), we know that the expected total connection cost $\mathbb{E}[\text{COST}]$ of a single run of k -means algorithm satisfies:

$$\mathbb{E}[\text{COST}] \leq C \cdot \text{OPT},$$

where C is the approximation factor, and OPT is the optimal cost of the k -means problem.

Our goal is to strengthen this guarantee from expectation to high probability. To achieve this, we employ probabilistic analysis using Markov's inequality.

Bounding the Probability for a Single Run. Using Markov's inequality, the probability that the total cost COST exceeds $K \cdot \text{OPT}$ in a single run is:

$$\Pr[\text{COST} > K \cdot \text{OPT}] \leq \frac{\mathbb{E}[\text{COST}]}{K \cdot \text{OPT}} \leq \frac{C}{K}.$$

By choosing $K = 2C$, we obtain:

$$\Pr(\text{COST} > K \cdot \text{OPT}) \leq \frac{1}{2}.$$

Repeating the Algorithm. We independently repeat the randomized portion of [Section 4.3](#) $T = \lambda \log n$ times in parallel, where $\lambda \geq 1$ is a constant to be determined later.

Bounding the Probability Over Multiple Repetitions. Since the repetitions are independent, the probability that *all* repetitions result in a total cost exceeding $K \cdot \text{OPT}$ is:

$$\Pr\left(\bigcap_{t=1}^T \{\text{COST}_t > K \cdot \text{OPT}\}\right) = (\Pr(\text{COST}_t > K \cdot \text{OPT}))^T \leq \left(\frac{1}{2}\right)^T = 2^{-T}.$$

By setting $T = \lambda \log n$, we have:

$$\Pr(\text{No run has } \text{COST}_t \leq K \cdot \text{OPT}) \leq n^{-\lambda}.$$

Selecting An Approximately Best Solution. By selecting the solution with the minimum total cost among the T repetitions, we ensure that with high probability (at least $1 - 1/n^\lambda$), the total cost is at most $K \cdot \text{OPT}$. We cannot compute the exact cost of a given solution. By using our graph approximation of the Euclidean metric space (cf. [Lemmas 5.4](#) and [5.6](#)), we can however compute the cost of a given solution up to a constant factor. We can therefore find an approximately best solution among the T repetitions, which still leads to a constant approximation of the k -means problem w.h.p.

Conclusion. By choosing $\lambda \geq 2$, the failure probability becomes negligible for large n . Therefore, by repeating the algorithm $O(\log n)$ times and selecting the best solution, we obtain a solution

whose total cost is within a constant factor K of the optimal cost with high probability, completing the proof. \square

5.5 Proof of the Main Theorem

It remains to prove the main theorem that is stated in [Section 1](#). It follows by [Theorems 4.7](#) and [4.10](#) that the highlevel algorithms described in [Sections 4.1](#) and [4.3](#) achieve a constant approximation for the facility location problem and the k -means problem. While [Theorem 4.10](#) only proves that the k -means algorithm achieves a constant approximation in expectation, by running the algorithm $O(\log n)$ in parallel (cf. [Lemma 5.21](#)), the constant approximation for the k -means problem can also be achieved w.h.p. In [Lemmas 5.11](#) to [5.17](#), it is shown that facility location algorithm of [Section 4.1](#) can be implemented in the MPC model in a fully scalable way in $O(\log \log n \cdot \log \log \log n)$ rounds. Similarly, in [Lemmas 5.18](#) to [5.20](#), it is shown that the k -means algorithm of [Section 4.3](#) can be implemented in $O(1)$ MPC rounds. Altogether, this proves the claim of the main theorem. \square

References

- [ABI86] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567–583, 1986.
- [AGLP89] B. Awerbuch, A. V. Goldberg, M. Luby, and S. A. Plotkin. Network decomposition and locality in distributed computation. In *Proc. 30th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 364–369, 1989.
- [AI06] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Proc. 47th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 459–468, 2006.
- [AJM09] Nir Ailon, Ragesh Jaiswal, and Claire Monteleoni. Streaming k-means approximation. In Yoshua Bengio, Dale Schuurmans, John D. Lafferty, Christopher K. I. Williams, and Aron Culotta, editors, *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada*, pages 10–18. Curran Associates, Inc., 2009.
- [ANFSW20] Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k -means and euclidean k -median by primal-dual algorithms. *SIAM Journal on Computing*, 49(4):FOCS17–97–FOCS17–156, 2020.
- [BBCA⁺19] Luca Becchetti, Marc Bury, Vincent Cohen-Addad, Fabrizio Grandoni, and Chris Schwiegelshohn. Oblivious dimension reduction for k-means: beyond subspaces and the johnson-lindenstrauss lemma. In *Proc. 51st ACM Symp. on Theory of Computing (STOC)*, page 1039–1050, 2019.
- [BEL13] Maria Florina Balcan, Steven Ehrlich, and Yingyu Liang. Distributed k-means and k-median clustering on general topologies. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2013, December 5-10, 2013, Lake Tahoe, Nevada, United States*, pages 1995–2003, 2013.
- [BGKO23] Alkida Balliu, Mohsen Ghaffari, Fabian Kuhn, and Dennis Olivetti. Node and edge averaged complexities of local graph problems. *Distributed Computing*, 36(4):451–473, 2023.
- [BGT12] Guy E. Blelloch, Anupam Gupta, and Kanat Tangwongsan. Parallel probabilistic tree embeddings, k-median, and buy-at-bulk network design. In Guy E. Blelloch and Maurice Herlihy, editors, *24th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA ’12, Pittsburgh, PA, USA, June 25-27, 2012*, pages 205–213. ACM, 2012.
- [BKBL07] Ji-Won Byun, Ashish Kamra, Elisa Bertino, and Ninghui Li. Efficient k-anonymization using clustering techniques. In *International conference on database systems for advanced applications*, pages 188–200. Springer, 2007.
- [BLK18] Olivier Bachem, Mario Lucic, and Andreas Krause. Scalable k-means clustering via lightweight coresets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 1119–1127. ACM, 2018.

- [BT10] Guy E. Blelloch and Kanat Tangwongsan. Parallel approximation algorithms for facility-location problems. In Friedhelm Meyer auf der Heide and Cynthia A. Phillips, editors, *SPAA 2010: Proceedings of the 22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures, Thira, Santorini, Greece, June 13-15, 2010*, pages 315–324. ACM, 2010.
- [BW18] Aditya Bhaskara and Maheshakya Wijewardena. Distributed clustering via LSH based data partitioning. In *Proceedings of the 35th International Conference on Machine Learning*, pages 570–579, 2018.
- [CAGLS23] Vincent Cohen-Addad, Fabrizio Grandoni, Euiwoong Lee, and Chris Schwiegelshohn. Breaching the 2 LMP approximation barrier for facility location with applications to k -median. In *Proc. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 940–986, 2023.
- [CEM⁺22] Vincent Cohen-Addad, Alessandro Epasto, Vahab Mirrokni, Shyam Narayanan, and Peilin Zhong. Near-optimal private and scalable k -clustering. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.
- [CGGJ25] Artur Czumaj, Guichen Gao, Mohsen Ghaffari, and Shaofeng H.-C. Jiang. Fully scalable MPC algorithms for euclidean k -center. In *Proc. 52nd Int. Coll. on Automata, Languages, and Programming (ICALP)*, volume 334 of *LIPIcs*, pages 64:1–64:20, 2025.
- [CGJ⁺24] Artur Czumaj, Guichen Gao, Shaofeng H.-C. Jiang, Robert Krauthgamer, and Pavel Veselý. Fully-scalable MPC algorithms for clustering in high dimension. In *Proc. 51st Int. Coll. on Automata, Languages, and Programming (ICALP)*, pages 50:1–50:20, 2024.
- [CGKR21] Vincent Cohen-Addad, Benjamin Guedj, Varun Kanade, and Guy Rom. Online k -means clustering. In Arindam Banerjee and Kenji Fukumizu, editors, *The 24th International Conference on Artificial Intelligence and Statistics, AISTATS 2021, April 13-15, 2021, Virtual Event*, volume 130 of *Proceedings of Machine Learning Research*, pages 1126–1134. PMLR, 2021.
- [CJK⁺22] Artur Czumaj, Shaofeng H.-C. Jiang, Robert Krauthgamer, Pavel Veselý, and Mingwei Yang. Streaming facility location in high dimension via geometric hashing. In *63rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 450–461. IEEE, 2022.
- [CKPU23] Mélanie Cambus, Fabian Kuhn, Shreyas Pai, and Jara Uitto. Time and space optimal massively parallel algorithm for the 2-ruling set problem. In *37th Int. Symp. on Distributed Computing (DISC)*, pages 11:1–11:12, 2023.
- [CLN⁺21] Vincent Cohen-Addad, Silvio Lattanzi, Ashkan Norouzi-Fard, Christian Sohler, and Ola Svensson. Parallel and efficient hierarchical k -median clustering. In *Proc. Conf. on Advances in Neural Information Processing (NeurIPS)*, pages 20333–20345, 2021.

- [CMZ22] Vincent Cohen-Addad, Vahab S. Mirrokni, and Peilin Zhong. Massively parallel k-means clustering for perturbation resilient instances. In *Proc. Int. Conf. on Machine Learning (ICML)*, pages 4180–4201, 2022.
- [COP03] Moses Charikar, Liadan O’Callaghan, and Rina Panigrahy. Better streaming algorithms for clustering problems. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 30–39. ACM, 2003.
- [DIIM04] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, page 253–262, 2004.
- [EIM11] Alina Ene, Sungjin Im, and Benjamin Moseley. Fast clustering using mapreduce. In Chid Apté, Joydeep Ghosh, and Padhraic Smyth, editors, *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pages 681–689. ACM, 2011.
- [Gha16] M. Ghaffari. An improved distributed algorithm for maximal independent set. In *Proc. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 270–277, 2016.
- [GP24] Jeff Giliberti and Zahra Parsaeian. Massively parallel ruling set made deterministic. In Dan Alistarh, editor, *Proc. 38th Int. Symp. on Distributed Computing (DISC)*, volume 319 of *LIPIcs*, pages 29:1–29:21, 2024.
- [GSZ11] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the MapReduce framework. In *Proc. Int. Symp. on Algorithms and Computation (ISAAC)*, pages 374–383. Springer, 2011.
- [GU19] Mohsen Ghaffari and Jara Uitto. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In *Proc. 30th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1636–1653, 2019.
- [IKL⁺23] Sungjin Im, Ravi Kumar, Silvio Lattanzi, Benjamin Moseley, and Sergei Vassilvitskii. Massively parallel computation: Algorithms and applications. *Foundations and Trends in Optimization*, 5(4):340–417, 2023.
- [Ind04] Piotr Indyk. Nearest neighbors in high-dimensional spaces. In Jacob E. Goodman and Joseph O’Rourke, editors, *Handbook of Discrete and Computational Geometry, Second Edition*, pages 877–892. Chapman and Hall/CRC, 2004.
- [JV01] Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k -median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001.
- [KMW18] Fabian Kuhn, Yannic Maus, and Simon Weidner. Deterministic distributed ruling sets of line graphs. In *Proc. 25th Int. Coll. on Structural Information and Communication Complexity (SIROCCO)*, pages 193–208, 2018.
- [KP12] K. Kothapalli and S. V. Pemmaraju. Super-fast 3-ruling sets. In *FSTTCS*, volume 18 of *LIPIcs*, pages 136–147, 2012.

- [KPP20] Kishore Kothapalli, Shreyas Pai, and Sriram V. Pemmaraju. Sample-and-gather: Fast ruling set algorithms in the low-memory MPC model. In *Proc. 40th IARCS Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 182 of *LIPICs*, pages 28:1–28:18, 2020.
- [KSV10] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A Model of Computation for MapReduce. In *Proc. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 938–948, 2010.
- [Llo82] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.
- [Lub86] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15:1036–1053, 1986.
- [Max60] Joel Max. Quantizing for minimum distortion. *IRE Transactions on Information Theory*, 6(1):7–12, 1960.
- [Mey01] Adam Meyerson. Online facility location. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 426–431. IEEE Computer Society, 2001.
- [MS08] Damon Mosk-Aoyama and Devavrat Shah. Fast distributed algorithms for computing separable functions. *IEEE Trans. Inf. Theory*, 54(7):2997–3007, 2008.
- [PPP⁺17] S. Pai, G. Pandurangan, S. V. Pemmaraju, T. Riaz, and P. Robinson. Symmetry breaking in the congest model: Time- and message-efficient algorithms for ruling sets. In *Proc. Symp. on Distributed Computing (DISC)*, volume 91 of *LIPICs*, pages 38:1–38:16, 2017.
- [SYZ18] Zhao Song, Lin F. Yang, and Peilin Zhong. Sensitivity sampling over dynamic geometric data streams with applications to k-clustering. *arXiv preprint arXiv:1802.00459*, 2018.