PTAS for Minimum Cost Multi-covering with Disks [*]

Ziyun Huang[†] Qilong Feng[‡] Jianxin Wang[§] Jinhui Xu[¶]

Abstract

In this paper, we study the following Minimum Cost Multi-Covering (MCMC) problem: Given a set of n *client points* C and a set of m *server points* S in a fixed dimensional $\mathbb{R}^d$ space, determine a set of disks centered at these server points so that each client point c is covered by at least $k(c)$ disks and the total cost of these disks is minimized, where $k(\cdot)$ is a function that maps every client point to some non-negative integer no more than m and the cost of each disk is measured by the α-th power of its radius for some constant $\alpha > 0$. MCMC is a fundamental optimization problem with applications in many areas such as wireless/sensor networking. Despite extensive research on this problem in the past two decades, only constant approximations were known for general k. It has been an open problem for a long time to determine whether a PTAS is possible. In this paper, we give an affirmative answer to this question by presenting the first PTAS for it. Our approach is based on a number of novel techniques, such as *Balanced Recursive Realization* and *Bubble Charging*, and new insights to the problem which are somewhat counter-intuitive. Particularly, we show that instead of optimizing each disk as a whole, it is possible to further approximate each disk with a set of sub-boxes and optimize them at the sub-disk level. This allows us to first compute an approximate disk cover with minimum cost through dynamic programming, and then obtain the desired disk cover through a balanced recursive realization procedure. Our techniques have the potential to be used to other geometric (covering) problems.

1 Introduction

In this paper, we study the following Minimum Cost Multi-covering (MCMC) problem. Given a set of n *client points* $C = \{c_1, c_2, \ldots, c_n\}$, a set of m *server points* $S = \{s_1, s_2, \ldots, s_m\}$ in a fixed dimensional $\mathbb{R}^d$ space, and a mapping $k(\cdot) : C \to \{1, 2, \cdots, m\}$, determine a radius assignment $\gamma : S \to \mathbb{R}$ which assigns a non-negative real value $\gamma(s)$ to every server point $s_j \in S$ as the radius of the disk centered at it so that each of the client point $c_i \in C$ is covered by at least $k(c_i)$ disks, and the total cost $\sum_{s_j \in S} \gamma(s_j)^\alpha$ of all the disks is minimized, where $\alpha > 0$ can be any positive constant. The MCMC problem has two basic requirements: the one that requires each client point c_i to be covered by at least $k(c_i)$ disks is called the *coverage* requirement, and the other that requires the total cost of all the disks to be minimized is called *cost minimization* requirement. A set $\mathcal{D}$ of disks that satisfies the coverage requirement is called a *disk k-cover* for the (S, C, k) instance of MCMC. Thus, the objective of MCMC is to find a radius assignment γ so that its induced disk set satisfies the coverage and cost minimization requirements simultaneously.

MCMC is a fundamental geometric optimization problem, and has been extensively studied in the past two decades [1, 2, 3, 4, 5, 6, 7, 8]. Originally, the problem was introduced for modeling the energy consumption problem in the communication of wireless/sensor networks, and later on found applications in several other areas. Broadly speaking, MCMC belongs to the family of geometric covering problems. In some sense, it can be viewed as a special variant of the well known geometric set cover problem [9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25] which has the following general formulation: Given a universe C of points (or objects) and a family $\mathbb{G}$ of ranges (or geometric objects like disks and polygons) in some Euclidean space with each range associated with a non-negative real number as its cost, determine a set $G \subseteq \mathbb{G}$ of ranges such that every element in C is covered by at least one range in G and the total cost of G is minimized. Clearly, MCMC can be formulated as a special geometric set cover problem, where $\mathcal{G}$ is the set of all possible disks centered at some server points and having at least one client point on their boundaries. The difference is that each client point $c_i \in C$ is required to be covered by not one, but $k(c_i)$ disks centered at different server points.

Geometric set cover problem is in general quite challenging to be solved optimally, even for some simple versions. For example, the unit disk covering problem,

[*]The research of the last author was supported in part by NSF through grants CCF-1716400 and IIS-1910492.

[†]Department of Computer Science and Software Engineering, Penn State Erie, the Behrend College. zxh201@psu.edu.

[‡]School of Computer Science and Engineering, Central South University. csufeng@csu.edu.cn.

[§]School of Computer Science and Engineering, Central South University. jxwang@csu.edu.cn.

[¶]Department of Computer Science and Engineering, State University of New York at Buffalo. jinhui@buffalo.edu.

where $\mathbb{G}$ is a set of unit disks on the plane and C is a set of 2D points, was shown by Feder and Greene to be NP-hard [20]. Thus, finding polynomial time approximation algorithms is the main objective for such problems. In [21], a polynomial time $(1+\epsilon)$-approximation algorithm has been achieved by Hochbaum and Maass for the above unit disk covering problem. Chekuri, Clarkson and Har-Peled [9] studied the general geometric covering problem for any shapes under fixed VC dimension, and achieved an $O(\log OPT)$-approximation.

Due to its connection to the geometric set cover problem, it is thus not surprising that MCMC is NP-Hard. In fact, it was shown by Alt *et al.* and Bilò *et al.* in [4, 1] that even for the case of $k(c_i) = 1$ for all client points c_i and $d = 2$, MCMC is still NP-Hard for any $\alpha > 1$. Our goal for MCMC is thus to design a polynomial time algorithm with the smallest approximation ratio.

1.1 Related Works A number of results exist for the MCMC problem. Below we discuss the ones that are most relevant to ours. Early efforts on this problem have focused on the version of MCMC where $k(c) = K$ for some fixed number K and any client $c \in C$. Charikar and Panigrahy [3] and Freund and Rawitz [26] considered the case of $K = 1$, and presented a linear programming based and a primal-dual based method, respectively, to achieve constant factor approximations. Lev-Tov and Peleg [2] studied the case of $\alpha = 1$ and designed a PTAS (Polynomial Time Approximation Scheme) for any constant dimensional space. Their approach is based on a dynamic programming method and can achieve optimality for $d = 1$ and a $(1+\epsilon)$-approximation for $d > 1$. Later, Bilò *et. al.* [1] presented a PTAS for any $\alpha \geq 1$ in constant dimensional space. Both of the above PTASes make use of a plane subdivision and shifted quad-tree technique from [27] for solving the minimum vertex cover problem on disk graphs. For the case of $K > 1$, Abu-Affash *et al.* [5] gave an $O(K)$-approximation for $\alpha = 2$ using a geometric approach. Later, Bar-Yehuda and Rawitz [28] achieved an $O(3^\alpha K + \epsilon)$-approximation for any $\alpha \geq 1$ and $\epsilon > 0$, using a local-ratio technique.

For the general MCMC problem (where k could be any mapping), Bhowmick, Varadarajan and Xue [6] proposed the first constant factor approximation (with approximate ratio $\rho = 2d(27\sqrt{d})^\alpha$). (This is also the first polynomial time constant factor approximation for the case of $k(c) = K$ for any $1 \leq K \leq m$.) To solve the general MCMC problem, they first introduced the concept of *outer cover*, which is a radius assignment for the set of server points so that every client point is covered by a large enough disk. Then, they adopted a geometric approach to obtain a disk k-cover from a disk $(k-1)$-cover by utilizing an outer cover produced by a primal-dual scheme.

1.2 Our Contributions and Main Ideas Despite the aforementioned tremendous progresses, it remains open to determine whether it is possible to achieve a PTAS for the general MCMC problem. To answer this question, we first analyze all previous approaches for MCMC.

A common feature of existing approaches for the MCMC problem is that they all treat each disk as a non-splittable entity and minimize the cost at the disk level (note that even though primal-dual based methods may increase their variables fractionally, they still view each disk as a whole and represent it by a single variable). A consequence of this is that it can easily cause too much rounding error for linear programming (or primal-dual) based approaches, and introduce an exponential number of sub-problems in dynamic programming based approaches, making them difficult to achieve a $(1+\epsilon)$-approximation for the general MCMC problem.

To overcome this challenge, our idea is to divide the task of finding a disk k-cover into two steps. In the first step, we use a shifted quad-tree like technique (called Γ-tree) to partition the space into a set of hierarchically organized boxes, and then develop a dynamic programming procedure on these boxes. To avoid the issue of having a possibly exponential number of sub-problems associated with each box in the dynamic programming, we view each disk as the union of a set of sub-boxes. This set of sub-boxes (called A-Disk) jointly approximates the shape of the disk, and is centered at a sub-box that contains at least one server point. Similarly, all client points are replaced by the sub-boxes containing them. By approximating the disks with A-Disks and replacing the input client/server points with sub-boxes, we can dramatically reduce the number of sub-problems associated with each box in the dynamic programming from exponential to polynomial. Thus, a dynamic programming procedure can be used to optimize the problem at sub-disk level and find in polynomial time a set of A-Disks (called AD-Cover) that satisfies the coverage requirement of the MCMC problem. It can be shown that the cost of the AD-Cover is quite close to the minimum cost of a disk k-cover.

The second step of our approach is to transform the obtained AD-Cover to a disk k-cover through a *balanced recursive realization* procedure. The main task of realization is to find a distinct server point for each A-Disk as the center so that the set of resulting disks preserve the coverage of the AD-Cover and do not incur too much additional cost. Since the A-Disks are determined in a bottom-up manner along the leaf-to-root paths of the Γ-tree in the dynamic programming procedure, A-Disks in sibling boxes of the Γ-tree are computed independently. A major issue is that they could compete for server points during the realization procedure. It is not always possible to resolve their conflicts locally (*i.e.,* within the same box), and needs to be resolved in a much larger region (*i.e.*, an upper level box). A consequence is that some conflicting disks have to expand their radii considerably. This imposes the risk of dramatically increasing the cost of the realized disks. Our idea is to start with a special AD-Cover called *Balanced AD-Cover* that reserves server points for the realizations of A-Disks in a "balanced" way across all levels of the Γ-tree. This Balanced AD-Cover can be obtained in the first step through dynamic programming, and ensures that most of the A-Disks can be realized locally. For the remaining conflicts, we lift them to an upper level box and resolve them there through a re-coordination procedure. By using a scheme called *bubble charging*, we are able to show that the total increased cost (due to disk expansions) can be bounded by an ϵ factor of the optimal cost. This leads us to the **first PTAS** of the general MCMC problem, which is summarized in the following main theorem. For any fixed $\alpha > 0$ and dimension d:

Theorem 1.1. *Given an MCMC instance (S, C, k), it is possible to find a $(1 + \epsilon)$-approximate minimum cost disk k-cover in $O(n^{O(1)} m^{O(1/\epsilon)^{O(d/\alpha)}})$ time.*

Our approach is somewhat counter-intuitive, especially the idea of sub-disk level optimization, since decomposing each disk into a set of sub-boxes often increases the complexity of the problem, thus making it more difficult to solve. However, as shown in our approach, it also gives us the room to optimize the disks more precisely. We believe that such an approach, along with the balanced recursive realization and bubble charging techniques, has potential to be used to solve other geometric (covering) problems.

2 Preliminaries

Let (S, C, k) be an instance of the MCMC problem with $S = \{s_1, s_2, \cdots, s_m\}$ and $C = \{c_1, c_2, \cdots, c_n\}$, and $Cost_{OPT}$ be the cost of an optimal disk k-cover $\mathcal{D}_{OPT}$ of (S, C, k). (S, C, k) is called a bounded instance if there exists a real number R_{bound} such that $(Cost_{OPT})^{1/\alpha} \le R_{bound} \le (\rho Cost_{OPT})^{1/\alpha}$ for some constant $\rho \ge 1$ and the diameter of the server point set S in $\mathbb{R}^d$ is no larger than $3mR_{bound}$. R_{bound} is called the radius bound of (S, C, k), and ρ is called the bound factor.

It can be shown that any instance of MCMC can be decomposed into a set of bounded instances of MCMC that can be solved independently, where ρ is the constant approximate ratio for MCMC achieved in [6] (The proof would be omitted due to space limitation and is left to the full version of this paper). Therefore, we will always assume that (S, C, k) is a bounded instance with radius bound R_{bound} and bound factor ρ, and our objective is to obtain a PTAS for (S, C, k) through dynamic programming.

We make use of a tree data structure called Γ-*Tree* for our dynamic programming algorithm, which is based on a shifted gird technique in [27]. (Here Γ is an integer factor whose value depends on ϵ and d and will be determined later.)

Definition 1. *A* Γ-Tree *is a Γ^d-ary tree, where each node represents a box (*i.e.*, an axis-aligned hypercube) in $\mathbb{R}^d$, and the children of every internal node B are obtained by decomposing B into Γ^d box equally using a Γ^d grid.*

Let T be a Γ-Tree with root B_r and height h. Each node (or box) B of T is associated with a level, which is its distance to B_r in T. The edge length of a box at level i is denoted by $L(i)$, which has the value of $L(i) = L(0)/(\Gamma)^i$, where $L(0)$ is the edge length of the root box. For any closed disk D in $\mathbb{R}^d$, if it is completely contained inside the box of B_r, and its diameter $L(i)/\Gamma(\Gamma - 1) \le Diam(D) < L(i)/(\Gamma - 1)$, then we say that D **fits** T and is **at level** i. We say that a disk set $\mathcal{D}$ fits T if every $D \in \mathcal{D}$ fits T. A level-i disk D is **good** for T if and only if D fits T and its interior does not intersect the boundary of any level i box of T. Otherwise, D is **bad** for T.

For any d-dimensional disk D with radius r, we define its cost $Cost(D)$ as $Cost(D) = r^\alpha$. Thus, the cost $Cost(\mathcal{D})$ of a set $\mathcal{D}$ of disks is $Cost(\mathcal{D}) = \sum_{D \in \mathcal{D}} Cost(D)$. By using a shifted quad-tree based technique in [27], for any given constant λ, it is possible to construct a set $\mathcal{T}(S, C, k)$ of Γ^d Γ-trees (called candidate trees) for (S, C, k), such that there exists a tree $T \in \mathcal{T}(S, C, k)$ and a disk k-cover $\mathcal{D}$ of (S, C, k) that fits T and satisfy the following:

1. $Cost(\mathcal{D}_{good}) \ge (1 - 1/(\Gamma - 1))^d Cost(\mathcal{D})$, where $\mathcal{D}_{good}$ is the set of disks in $\mathcal{D}$ that are good for T.
2. $Cost(\mathcal{D}) \le (1 + \lambda) Cost_{OPT}$ where $Cost_{OPT}$ is the minimum cost of a disk k-cover of (S, C, k).

Each tree in $\mathcal{T}(S, C, k)$ has height $O(\log(m\rho/\lambda))$ and $(m\rho/\lambda)^{O(d \log \Gamma)}$ nodes in total. Construction would take $O(\Gamma^d (m\rho/\lambda)^{O(d \log \Gamma)})$ time. Details about the construction would be left to the full version of this paper.

To approximate each disk by smaller boxes, we further partition each box B at level i of a Γ-tree T into Δ^d small boxes of the same size by a Δ^d grid, where $\Delta > 1$ is a to-be-determined integer constant (depends on ϵ and d) satisfying the conditions of $\Delta \geq 2\sqrt{d}\Gamma^2$ and $\Delta/2(\Gamma(\Gamma-1)) > 1$ is an integer. These boxes are called **sub-boxes** of T at level i. A level-i sub-box U is called an *inner sub-box* of B at the same level, if U is contained inside B. The set of all inner sub-boxes of B is denoted by $\mathcal{I}(B)$. The $\frac{\Delta}{2(\Gamma-1)}$ layers of level-i sub-boxes around B (see Figure 1) are called the *outer sub-boxes* of B and denoted by $\mathcal{O}(B)$. All boxes in $\mathcal{S}(B) = \mathcal{I}(B) \cup \mathcal{O}(B)$ are called the sub-boxes of B.

From the above discussion, it is easy to see that for any sub-box U at level i, there exists exactly one level-i box that has U as its inner sub-box. There are at most $2^d - 1$ level-i boxes that have U as one of their outer sub-boxes. A sub-box U at level i is the union of Γ^d sub-boxes at level $i+1$, except for the case where U is a sub-box of a leaf node of T. Since the number of layers (*i.e.,* $\Delta/2(\Gamma-1)$) of outer sub-boxes is divisible by Γ, for any sub-box U at level i and any node B at level $i+1$, either none of the sub-boxes of B is covered by U, or U is a union of Γ^d sub-boxes of B. If U is a sub-box of level i and U' is a sub-box of level $i' \geq i$, then U' is either completely inside or completely outside of U. If U' is contained inside U, U is an ancestor sub-box of U' and U' is a descendant sub-box of U. Particularly, if $i' = i+1$, U is an parent sub-box of U' and U' is a child sub-box of U, denoted by $U' \sqsubset U$.

2.1 Overview of Our Approach As mentioned earlier, our approach consists of two steps, 1) finding a Balanced AD-Cover and 2) realizing the AD-Cover to obtain a disk k-cover. Below is the outline of our strategy for achieving a PTAS of the general MCMC problem.

(1) We first show that there exists a disk k-cover $\mathcal{D}$ whose corresponding A-Disk multi-set (generated through a procedure called **discretization**) on one of the Γ-trees T in $\mathcal{T}(S, C, k)$ has a cost close to $Cost_{OPT}$. (Section 3)

(2) We then introduce the concept of Balanced AD-Cover for A-Disk multi-set $\mathcal{A}$ (Section 4) and show that such defined Balanced AD-Covers indeed exist; actually the discretization of any disk k-cover is a Balanced AD-Cover (Section 6).

(3) We demonstrate that any Balanced AD-Cover can be realized to a disk k-cover with similar cost through a Balanced Recursive Realization algorithm. (Section 5)

(4) We show that given any Γ-tree T in $\mathcal{T}(S, C, k)$, a minimum cost Balanced AD-Cover $\mathcal{A}$ of T can be found in polynomial time by a dynamic programming procedure. (Section 7)

(5) Repeatedly apply the algorithm in step (4) to all Γ-trees in $\mathcal{T}(S, C, k)$, and identify the A-Disk multi-set $\mathcal{A}_{OPT}$ with the minimum cost. Then, the realization of $\mathcal{A}_{OPT}$ (using the algorithm in step (3)) will have a cost close to $Cost_{OPT}$. This leads to a $(1+\epsilon)$-approximate minimum cost disk k-cover for (S, C, k).

The most challenging part of our approach lies in Section 5, which heavily relies on the concept of Balanced AD-Cover in Section 4.

3 Approximate Disks and Discretization

For a level-l box B of T and its two sub-boxes $U_0 \in \mathcal{S}(B)$ and $U_1 \in \mathcal{I}(B)$, let o_0 and o_1 be the center of U_0 and U_1, respectively. Assume that $\|o_0o_1\| \geq (\frac{\Delta}{2\Gamma(\Gamma-1)} - \sqrt{d})L$, where L is the edge length of U_0 (or U_1). We define the approximate disk determined by U_0 and U_1 as follows, where D is the disk centered at o_0 and with radius $\|o_0o_1\|$.

Definition 2. *Let $A \subset \mathcal{I}(B)$ be the set of inner sub-boxes intersecting the interior of D. A is called an* approximate disk *(or* A-Disk*) of B determined by U_0 and U_1, and U_0 is called the* center box *of A.*

From the definition, it is easy to see that if U_0 and U_1 are far enough, their determined A-Disk is a good approximation of the disk centered at some server point in U_0 and with radius $\|o_0o_1\|$. Note that an A-Disk may only approximate a portion of the disk, instead of the entire one. Below we discuss how to transform an A-Disk into a disk.

Definition 3. *Let A, U_0, U_1, o_0, o_1 be defined as above, and $s \in S$ be a server point in U_0. A* realization *of A at s is a disk centered at s and with radius $\|o_0o_1\| + 2\sqrt{d}L$, where L is the edge length of U_0.*

Figure 2 shows a possible realizations of an A-Disk. Intuitively, if the grid is dense enough (*i.e.,* Δ is large enough), an A-Disk could be geometrically close to its realization (see Figure 2 (a)). We define the cost of an A-Disk A to be $Cost(A) = (\|o_0o_1\| + 2\sqrt{d}L)^\alpha$, which is the α-th power of its realization disk's radius. We denote by D_A a realization of an A-Disk A.

Lemma 3.1. *A is completely contained inside D_A.*

The following definition shows how to transform a closed disk D into an A-Disk, or a number of A-Disks.

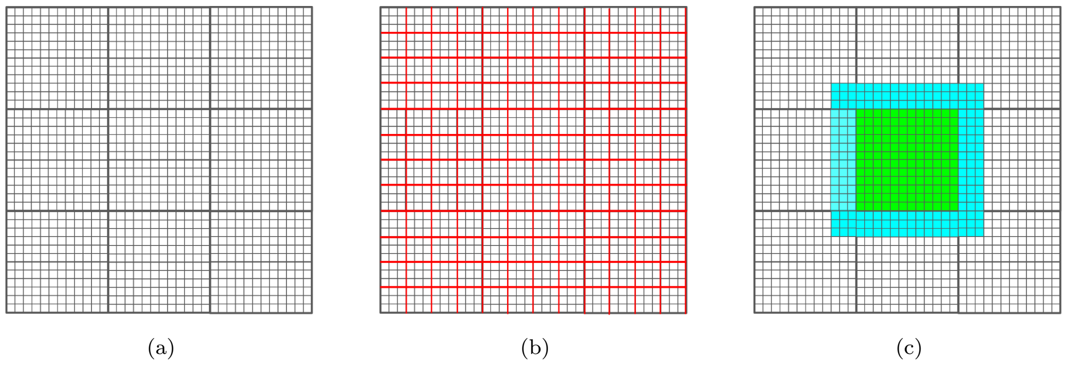

Figure 1: Illustration of a node (box) of a Γ-tree and its children, together with their sub-boxes, where $\Gamma = 3, \Delta = 12$ and $d = 2$. Note that for better illustration, values of Γ, Δ in the examples might not satisfy all of the requirements (*e.g.* $\Delta \geq 2\sqrt{d}\Gamma^2$). (a) A box B is divided (shown by the bold line) to 9 children boxes. Each child is divided to 12×12 inner sub-boxes. (b) Red lines divide B to 12×12 inner sub-boxes. Sub-boxes at child level are also shown. Note that each sub-box of B is a union of 3×3 child sub-boxes. (c) Inner (Colored green) and outer (Colored blue) sub-boxes of the middle child of B. Outer sub-boxes of a node are the $\Delta/2(\Gamma - 1)$ (which is 3 in this example) layers of sub-boxes at same level around it.

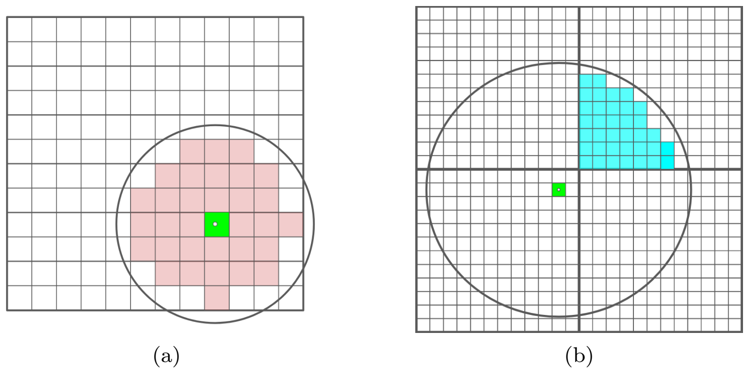

Figure 2: Illustration of 2 A-Disks and their realizations. The centers of both A-Disks are colored in green. The center of a realization is a server point in the center box of the A-Disk. In (b), the A-disk is on the upper right box B, but the center is an outer sub-box of B. In this case the center is not considered geometrically a part of the A-Disk.

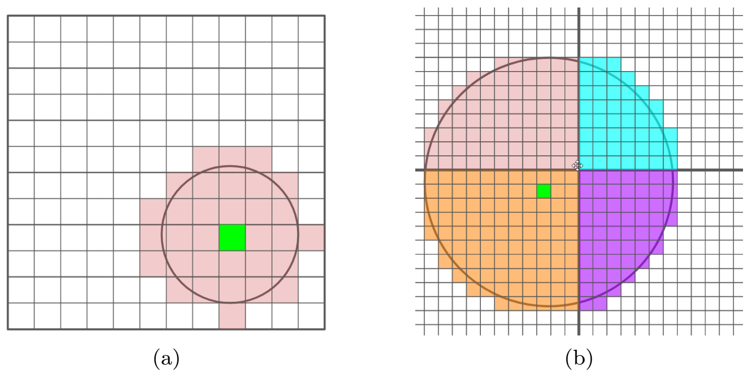

Figure 3: Illustration of 2 disks and their discretizations. In (a), the disk is entirely contained in a node, and thus has exactly one discretization. In (b), the disk intersects 4 nodes, and thus has 4 discretizations (marked in 4 different colors). All discretizaions have the same center, marked in green.

Definition 4. *Let D be a disk that fits T. A* ***discretization*** *of D on T is an A-Disk that is created through the following process.*

1. *Choose a box B of T at the same level l as D such that B and D have non-empty intersection.*
2. *Let U_0 be the level-l sub-box containing the center of D and U_1 be the farthest inner sub-box of B to U_0 (where the distance between U_0 and U_1 is defined as the distance of their corresponding centers) that has non-empty intersection with D.*
3. *The A-Disk A determined by U_0 and U_1 with U_0 as the center is then a* ***discretization*** *of D.*

We first show in the following lemma that the definition is consistent with the notion of A-Disk by showing (1) U_0 is a sub-box of B and (2) the distance between U_0 and U_1 is at least $(\frac{\Delta}{2\Gamma(\Gamma-1)} - \sqrt{d})L$.

Lemma 3.2. *The discretizaion procedure in Definition 4 generates an A-Disk of B.*

An example of discretization is shown in Figure 3. Note that a disk D that fits T has at least one and at most 2^d discretizations on T. (Recall the size requirement for a disk at level l. A disk at level l could intersect at most 2^d level l boxes.) The following fact about discretization is important: Let D be a disk that fits T. D has exactly one discretization on T if and only if D is good for T.

It is easy to see that D is fully contained inside the union of all its discretizations.

Lemma 3.3. *Let $\mathcal{A}_D$ be the set of all discretizations of a disk D on T. The union of sub-boxes in $\mathcal{A}_D$ covers the whole region of D.*

The following lemma shows that the cost of a discretization of a disk D is close to that of D, with appropriately chosen parameters Δ, Γ and λ.

Lemma 3.4. *Let A be any discretization of a disk D on T. Then, $Cost(D) \leq Cost(A) \leq (1 + 2\sqrt{d}\Gamma(\Gamma - 1)/\Delta)^\alpha Cost(D)$.*

3.1 Discretization of a Disk Set Given a disk set $\mathcal{D}$ such that all the disks in $\mathcal{D}$ fit T, it is possible to produce a set of A-Disks from $\mathcal{D}$ through the discretization process.

Definition 5. *The discretization of $\mathcal{D}$ on T is the set of A-Disks that are discretizations of every disk in $\mathcal{D}$ on T.*

Note that each disk in $\mathcal{D}$ can contribute up to 2^d A-Disks in the discretization of $\mathcal{D}$. As a result, the discretization of a disk set could have a much larger cost than the disk set, even though the cost of a single discretization is quite close to that of the corresponding disk (with carefully chosen parameters). However, if T is good for $\mathcal{D}$ (i.e. the total cost of bad disks in $\mathcal{D}$ is only a small fraction ($\leq (1 - (\Gamma - 1)^{-1})^d$) of the total cost of $\mathcal{D}$), we know that its discretization could have a similar cost.

Lemma 3.5. *Let T be a Γ-tree good for a disk set $\mathcal{D}$, $\mathcal{A}$ be the discretization of $\mathcal{D}$ on T. Then, $Cost(\mathcal{D}) \leq Cost(\mathcal{A}) \leq (1 + 2\sqrt{d}\Gamma(\Gamma - 1)/\Delta)^\alpha((1 - (\Gamma - 1)^{-1})^d + 2^d(1 - (1 - (\Gamma - 1)^{-1})^d))Cost(\mathcal{D})$.*

It is not hard to see that if Γ and Δ are large enough, $Cost(\mathcal{D})$ and $Cost(\mathcal{A})$ could be arbitrarily close. By Lemma 3.5 and properties of $\mathcal{T}(S, C, k)$, we have the following key fact, where $\lambda > 0$ is a constant used in the creation of $\mathcal{T}(S, C, k)$.

Lemma 3.6. *There exists at least one Γ-tree T from $\mathcal{T}(S, C, k)$ and one disk set $\mathcal{D}$ such that the following is true. (1) $\mathcal{D}$ is a disk k-cover for (S, C, k). (2) $\mathcal{D}$ fits T. (3) The discretization $\mathcal{A}$ of $\mathcal{D}$ on T has a cost no more than*

$$Cost(\mathcal{A}) \leq (1 + \lambda)(1 + 2\sqrt{d}\Gamma(\Gamma - 1)/\Delta)^\alpha ((1 - (\Gamma - 1)^{-1})^d + 2^d(1 - (1 - (\Gamma - 1)^{-1})^d))Cost_{OPT}, \tag{3.1}$$

where $Cost_{OPT}$ is the minimum cost of a disk k-cover for (S, C, k).

4 Balanced AD-Cover

In this section, we introduce a set of key properties to ensure that A-Disk multi-sets can be mutually transformable with disk k-covers. We start with the requirements on an A-Disk multi-set $\mathcal{A}$.

(1) **Full coverage:** Every client point $c \in C$ should be covered by at least $k(c)$ A-Disks in $\mathcal{A}$.

(2) **Realizability:** It is possible to realize every A-Disk in $\mathcal{A}$ with a disk so that the resulting disk set $\mathcal{D}$ preserves the coverage of $\mathcal{A}$. That is, for every client point $c \in C$, if it is covered by x A-Disks in $\mathcal{A}$, it should be covered by at least x disks in $\mathcal{D}$ that are centered at different server points in S. [1]

[1]Note that we will show later that the process of obtaining $\mathcal{D}$ from $\mathcal{A}$ is actually much more complicated than simply realizing each A-Disk independently to a disk.

To design properties for $\mathcal{A}$ to meet the above requirements, we first observe that A-Disks are defined on the Γ-tree T, which is naturally a hierarchical structure. This suggests that the properties should be formulated in a recursive manner so that a dynamic programming approach can be used to find an A-Disk multi-set that is realizable to a disk k-cover with similar cost. This means that we need to define $\mathcal{A}$ recursively and determine its recurrence relations. For this purpose, we let B_r be the root of T, and $B_1, B_2, \ldots, B_{\Gamma^d}$ be its children. For $i = 1, 2, \ldots, \Gamma^d$, denote by T_i the subtree of T rooted at B_i. Clearly, any A-Disk multi-set $\mathcal{A}$ of T can be partitioned into $\Gamma^d + 1$ subsets (with possibly some empty sets), $\mathcal{A} = \mathcal{A}_r \cup \mathcal{A}_1^* \cup \mathcal{A}_2^* \ldots \cup \mathcal{A}_{\Gamma_d}^*$, where $\mathcal{A}_r = \{A \in \mathcal{A} \mid A \text{ is an A-Disk of } B_r\}$, $\mathcal{A}_i = \{A \in \mathcal{A} \mid A \text{ is an A-Disk of } B_i\}$, and $\mathcal{A}_i^* = \{A \in \mathcal{A} \mid A \text{ is an A-Disk of a node of } T_i\}$. Thus, the process of searching for an A-Disk multi-set $\mathcal{A}$ from T with the desired properties consists of 2 tasks: (1) Determine $\mathcal{A}_r$ by selecting A-Disks of B_r, and (2) Determine $\mathcal{A}_i^*$ from T_i. Next, we discuss how to design the recurrence relations to meet the full coverage and realizability requirements

1. Full Coverage. Note that in the coverage requirement of a disk k-cover, every client point c needs to be covered by at least $k(c)$ disks with different centers. Since any A-Disk A is fully covered by its realization, the Full Coverage requirement for A-Disk multi-set $\mathcal{A}$ naturally ensures the coverage requirement in any realization $\mathcal{D}$ of $\mathcal{A}$.

To enforce the full coverage requirement for $\mathcal{A}$, we restate it in a recursive manner. Consider any client point $c \in C$ that needs to be covered by $k(c)$ A-Disks. Let B_i be the child of B_r in T that contains c in its interior. If an A-disk $A \in \mathcal{A}$ covers c, then either $A \in \mathcal{A}_r$ or $A \in \mathcal{A}_i^*$. Let $k_{loc}(c)$ and $k_i(c)$ denote the number of A-disks in $\mathcal{A}_r$ and $\mathcal{A}_i^*$ that cover c, respectively. Then, the full coverage requirement can be represented by inequality $k_{loc}(c) + k_i(c) \geq k(c)$ for all c in C. In other words, for any client point c in B_i, it needs to be covered by at least $k(c) - k_{loc}(c)$ A-Disks in $\mathcal{A}_i^*$.

In the latter statement, the client points are limited to those in B_i, A-Disks are limited to those in $\mathcal{A}_i^*$ (*i.e.*, A-disks of nodes in subtree T_i), and the number of times that c needs to be covered is reduced by $k_{loc}(c)$ (*i.e.,* the number of times that c is covered by A-Disks of the upper level node B_r). Generalizing the above requirement to an arbitrary node B of T, we have the following concept of K-AD-Cover. Briefly speaking, a K-AD-Cover of a node B of T is a multi-set of A-Disks that requires every client c in B to be covered by $k(c)$ minus a relieved cover number specified by a mapping K. The relieved cover number reflects the coverage of c by A-Disks of ancestor nodes of B.

Definition 6. *Let B be a box of T and $K(\cdot)$ be a mapping that maps every inner sub-box U of B to a non-negative integer in $\{0, 1, 2, \ldots, m\}$. Let $\mathcal{A}^*$ be a set of A-Disks of B or its descendants. $\mathcal{A}^*$ is called a **K-AD-Cover** of B for instance (S, C, k), if for any client $c \in C$ that lies in some sub-box U of B, it is covered by at least $k(c) - K(U)$ A-Disks in $\mathcal{A}^*$.*

Note that $\mathcal{A}$ satisfies the full coverage requirement if and only if it is a K_0-AD-Cover of B_r, where $K_0(\cdot) := 0$.

2. Realizability. Recall that the realization of an A-Disk involves choosing a server point s from its center box and creating a disk using s as the center. Consider transforming $\mathcal{A}$ to a disk k-cover by realizing each of its A-Disks. Note that in a disk k-cover, disks should have distinct centers, thus a server point cannot be chosen multiple times as the center during the transformation. Since A-Disks could compete for server points (as their centers) in their realizations, a certain mechanism is hence needed to ensure that conflicts can be properly resolved and the realizability requirement can be satisfied.

We informally describe the process of realizing A-Disks in $\mathcal{A}$ as a recursive procedure, which consists of 2 main steps: (1) Realize each of the A-Disks multi-sets $\mathcal{A}_i^*$ for $i = 1, 2, \ldots, \Gamma^d$; (2) Realize each of the A-Disks in $\mathcal{A}_r$; in this step, avoid using the same server point as center multiple times, and avoid using server points that had already been used in the realization of $\mathcal{A}_i^*$. Note that step (2) can be completed only if there is a sufficient number of unused server points after the realization of all $\mathcal{A}_i^*$. More specifically, consider any sub-box U of B_r. Let Φ_U be the number of A-Disks in $\mathcal{A}_r$ that have U as the center box, and $w(U)$ be the number of server points that lie in U. Then, step (2) is possible only when the number of server points in U that are used to realize all $\mathcal{A}_i^*$ does not exceed $w(U) - \Phi_U$ for every U of B_r. In other words, the realization of all $\mathcal{A}_i^*$ should reserve at least Φ_U points (for every U of B_r) for the realization of upper level A-Disks in $\mathcal{A}_r$.

The above discussion leads us to introduce a parameter Φ to control the number of reserved server points for the realization of upper level A-Disks. Let B be any internal node of T and $\mathcal{A}^*$ be a multi-set of A-Disks of B or its descendants. Let $\Phi(\cdot)$ be a mapping that maps every sub-box of B to a non-negative integer in $\{0, 1, 2, \ldots, m\}$. The mapping $\Phi(\cdot)$ can be interpreted as the reservation requirement for each sub-box: when realizing $\mathcal{A}^*$, for any sub-box U of B, at least $\Phi(U)$ server points in U should be reserved for the realization of upper level A-Disks.

With Φ, we can now introduce the concept of Φ-Balanced. Intuitively, $\mathcal{A}^*$ is Φ-Balanced if it is possible to realize all its A-Disks without causing any conflict, and the server points are reserved properly as specified by Φ. Let $\mathcal{A}(B, U)$ be the set of A-Disks of B in $\mathcal{A}^*$ that are centered at U. For $i = 1, 2, \ldots, \Gamma^d$, let B'_i be a child of B and $\mathcal{A}'^*_i = \{A \in \mathcal{A} \mid A \text{ is an A-Disk of } B'_i \text{ or its descendant}\}$. Then, in order to realize A-Disks in $\mathcal{A}(B, U)$, we can only use server points in U that are reserved while realizing $\mathcal{A}'^*_i$ to avoid conflict with lower level A-Disks. Thus, when realizing $\mathcal{A}'^*_i$, we need to ensure that the total number of reserved server points from all child sub-boxes of U should be $\Phi(U) + |\mathcal{A}(B, U)|$, where $|\mathcal{A}(B, U)|$ is the number of server points for realizing A-Disks in $\mathcal{A}(B, U)$, and $\Phi(U)$ is the number of server points reserved for realizing upper level A-Disks.

Definition 7. *$\mathcal{A}^*$ is called Φ-**Balanced** if the following holds.*

- *$|\mathcal{A}(B, U)| + \Phi(U) \leq w(U)$ for any sub-box U of B.*
- *If B is an internal node of T, there exists a mapping η that maps every child sub-box U' of any sub-box of B to a non-negative integer $0 \leq \eta(U') \leq m$ so that the following conditions are satisfied.*
    1. *$\sum_{U' \sqsubset U} \eta(U') = \Phi(U) + |\mathcal{A}(B, U)|$ for any sub-box U of B.*
    2. *$\mathcal{A}'^*_i$ is Φ_i-Balanced, where Φ_i is the mapping that maps every sub-box U' of B'_i to $\Phi_i(U') = \eta(U')$.*

In the above definition, $\eta(U')$ is a mapping indicating the number of server points in any sub-box U' reserved for realizing upper level A-Disks. It provides a scheme to implement $\Phi(U)$ (and $|\mathcal{A}(B, U)|$) which is defined on all sub-boxes U of a node B of T. Since a sub-box U' can be either an inner or an outer sub-box of (up to) 2^d nodes of T, in some sense it serves as a bridge and a coordinator in the otherwise independent implementations of all those $\Phi_i(\cdot)$ associated with those 2^d nodes.

Note that the above definition only specifies the number of server points that need to be reserved in each sub-box. However, as we will see in Section 5, realization is computed independently among all those $\mathcal{A}^*_i$'s with overlapping sub-boxes among their corresponding sub-trees T_i's. Such independent computation may lead to different subsets of server points being reserved in some sub-boxes. Thus, a mechanism called *re-coordination* is needed to enforce consistency, which is also one of the main challenges in realization.

An A-Disk multi-set $\mathcal{A}$ is called a (K, Φ)**-AD-Cover** if it is a K-AD-Cover and also Φ-Balanced. $\mathcal{A}$ is called a **Balanced AD-Cover** for (S, C, k) if $\mathcal{A}$ is a (K_0, Φ_0)-AD-Cover, where K_0 is a mapping that maps every inner sub-box of the root B_r of T to 0, and Φ_0 is a mapping that maps every sub-box of B_r to 0.

In Section 6, we will show that such defined Balanced AD-Cover indeed exists and can be obtained from the discretization of a disk k-cover. In Section 7, we will demonstrate that a minimum cost Balanced AD-Cover can be computed through dynamic programming. In Section 5, we present an algorithm to transform a Balanced AD-Cover to a disk k-cover with similar cost.

5 Realization of Balanced AD-Cover

In this section, we show that any Balanced AD-Cover $\mathcal{A}$ of a Γ-tree T can be transformed into a disk k-cover for (S, C, k) with similar cost through a balanced recursive realization algorithm.

5.1 The Balanced Recursive Realization Algorithm For any node B of T, let $\mathcal{A}(B) = \{A \in \mathcal{A} \mid A \text{ is an A-Disk of } B\}$ and $\mathcal{A}^*(B) = \{A \in \mathcal{A} \mid A \text{ is an A-Disk of } B \text{ or its descendant}\}$. For any sub-box U of B, let $\Phi_{loc}(B, U)$ be the number of A-Disks of B centered at U, *i.e.*, $\Phi_{loc}(B, U) = |\mathcal{A}(B, U))|$. From the definition of Balanced AD-Cover, we know that it is possible to assign to each node B of T at level l a mapping Φ_B (to map every sub-box U of B to an integer $0 \leq \Phi_B(U) \leq m$) and a mapping η_B (to map every sub-box U' at level $l + 1$ that is a child sub-box of a sub-box U of B to a non-negative integer) so that

- $\Phi_{B_r}(U) = 0$ for any sub-box U of the root B_r of T.
- $\mathcal{A}^*(B)$ is Φ_B-Balanced for every node B of T.
- $\sum_{U' \sqsubset U} \eta_B(U') = \Phi_{loc}(B, U) + \Phi_B(U)$ for every internal node B of T and any sub-box U of B.
- $\Phi_{B'}(U') = \eta_B(U')$ for every internal node B of T, any child B' of B, and any sub-box U' of B'.

To realize a Balanced AD-Cover $\mathcal{A}$, we assume that two mappings, Φ_B and η_B, are also given along with $\mathcal{A}$, for every node B of T. In Section 7, we will show that such information can indeed be computed through dynamic programming.

The following balanced recursive **Realization** algorithm (Algorithm 1) can then transform $\mathcal{A}$ into a disk k-cover in a bottom-up manner. The algorithm takes as input a Balanced AD-Cover $\mathcal{A}$ and a node B of T, and outputs a disk set $\mathcal{D}(B)$ that is the transformation of $\mathcal{A}^*(B)$, together with a set $S_B(U)$ of $\Phi_B(U)$ server points in U for every sub-box U of B. $S_B(U)$ is the set of server points that have not been used in the realization of $\mathcal{A}^*(B)$, and thus can be used for the realization of upper level A-Disks in T. $\mathcal{D}(B_r)$ from Realization$(\mathcal{A}, B_r)$ would then be the disk set realized from $\mathcal{A}$. Since the algorithm runs in a bottom-up manner, it assumes that either B is a leaf node of T or Algorithm 1 has already been executed on all children of B.

If B is a leaf node of T, Algorithm 1 directly realizes every A-Disk in $\mathcal{A}(B)$ and outputs the resulting disks. Otherwise (*i.e.,* B is an internal node), the algorithm goes through a realization process for every A-Disk in $\mathcal{A}(B)$, and then outputs the generated disks, together with the realized disks $\mathcal{D}(B_i)$ (which might be altered by the Re-Coordination algorithm in Step 4; See Figure 4 for an illustration of the process of re-coordination) for each child B_i of B. If there are multiple disks in $\mathcal{D}(B)$ that share a common center, only the one with the largest radius will be kept. To realize A-Disks in $\mathcal{A}(B)$, our algorithm generates a set $S_{loc}(U)$ of server points in each sub-box U of B that are not used for the realization of any A-Disk in $\mathcal{A}^*(B_i)$. Realization of A-Disks with center U (*i.e.* $\mathcal{A}(B, U)$) can use only server points in $S_{loc}(U)$ to avoid conflict with lower level disks. Step 4 and the Re-Coordination algorithm are to enforce consistency on points not used for the realization of any $\mathcal{A}^*(B_i)$, and resolve their potential conflicts.

5.2 Correctness Analysis To analyze the above algorithm, we first show the correctness of the algorithm (*i.e.,* it generates a disk k-cover from a Balanced AD-Cover), then estimate the cost of the resulting disk k-cover, and finally analyze the running time of the algorithm. We start with the following claims.

(1) In step 2 of Algorithm 2, it is possible to complete the realization without using the same server point multiple times.

(2) In step 3 of Algorithm 2, it is possible to choose server points for $S_B(U)$ as described.

(3) In step 5 of Algorithm 1, for all sub-box U of B, it is indeed possible to realize all A-Disks in $\mathcal{A}(B)$ centered at U by using server points in $S_{loc}(U)$ but not using the same server point multiple times.

(4) The output of Algorithm 1 matches the output description at the beginning of the algorithm.

Below, we argue that the claims are indeed true.

Proof. **[Claims (1) and (2)]** Note that Algorithm 2 always runs on a leaf node B. Since $\mathcal{A}^*(B) = \mathcal{A}(B)$ is Φ_B-Balanced, for any sub-box U of B, we have $\Phi_{loc}(B, U) + \Phi_B(U) \leq w(U)$ (where $w(U)$ is the number of server points in U). The realization of the $\Phi_{loc}(B, U)$ A-Disks in $\mathcal{A}(B)$ with U as the center takes only $\Phi_{loc}(B, U)$ different server points from U. After the realization, at least $w(U) - \Phi_{loc}(B, U) \geq \Phi_B(U)$ distinct server points in U can be chosen to form $S_B(U)$. Thus, step 2 and 3 of Algorithm 2 can always find enough distinct server points to output. Hence, Claims (1) and (2) are true.

□

Proof. **[Claims (3) and (4)]** We first show that after step 4 of Algorithm 1, there are $\Phi_B(U) + \Phi_{loc}(B, U)$ server points in $S_{loc}(U)$, and none of them is the center of a disk in $\mathcal{D}_{Pr}(B')$ for any child B' of B. To demonstrate this, we consider the following 2 cases.

Case 1: U does not cover any sub-box of a child of B. This means that U does not cover the sub-box of any descendant of B. It implies that none of the server points in U is a center of a disk in $\mathcal{D}_{Pr}(B')$ for any child B' of B. Note that $S_{loc}(U)$ is created in step 4.1. Since $w(U) \geq \Phi_B(U) + \Phi_{loc}(B, U)$, clearly it is possible to choose $\Phi_B(U) + \Phi_{loc}(B, U)$ server points from U to form $S_{loc}(U)$, and none of them is a center of a disk of $\mathcal{D}_{Pr}(B')$ for any child B' of B.

Case 2: U covers some sub-box of a child of B. This means that every child sub-boxes $U_1, U_2, \ldots, U_{\Gamma^d}$ of U is a sub-box of some children of B. Thus, for every child sub-box U_i, step 4.2 calls Re-Coordination. This means that all disks in any of $\mathcal{D}_{Pr}(B'_j)$ do not use any server point in $S_{B'_1}(U_i)$ as the center. Note that the input of the call to Re-Coordination is legitimate: for any $j = 1, 2, \ldots t$, $|S_{B'_j}(U_i)| = |S_{B'_1}(U_i)| = \eta_B(U_i) = \Phi_{B'_j}(U_i) = \Phi_{B'_1}(U_i)$. As a result, after step 4.2, $S_{loc}(U)$ has $\eta_B(U_i)$ server points from U_i that are not the center of any disk in $\mathcal{D}_{Pr}(B')$ for any child B' of B, and the total number of server points in $S_{loc}(U)$ is $\sum_{1 \leq i \leq \Gamma^d} \eta_B(U_i) = \Phi_B(U) + \Phi_{loc}(B, U)$.

Now in step 5, when putting disks into $\mathcal{D}_{loc}$, for every sub-box U, we need to use $\Phi_{loc}(B, U)$ server points to realize those A-Disks in $\mathcal{A}(B)$ that use U as the center. After that, U still has $\Phi_B(U) + \Phi_{loc}(B, U) - \Phi_{loc}(B, U) = \Phi_B(U)$ unused server points, which can be output as $S_B(U)$. Clearly none of these points is a center of a disk in $\mathcal{D}(B)$. This implies that claims (3) and (4) are both true.

Clearly our algorithm guarantees that no two disks in $\mathcal{D}(B)$ use the same server point as center. □

Next, we prove some key properties of the disk set generated by the Realization algorithm. We start with some facts and notations.

Algorithm 1 Realization($\mathcal{A}, B$)

```
Input: A-Disk multi-set $\mathcal{A}$ that is a Balanced AD-cover for $(S, C, k)$, and a node $B$ of $T$.
Output: A set $\mathcal{D}(B)$ of $\mathbb{R}^d$ disks. A set $S_B(U)$ of $\Phi_B(U)$ server points for every sub-box $U$ of $B$, with none of them used as a center for a disk in $\mathcal{D}(B)$. No multiple disks in $\mathcal{D}(B)$ share the same server point as center.
1: If $B$ is a leaf node of $T$, Call Realization-Leaf($\mathcal{A}, B$) and return the result.
2: Initialize an empty server point set $S_{loc}(U)$ for every sub-box $U$ of $B$.
3: Initialize an empty disk set $\mathcal{D}_{loc}$.
4: For any child $B'$ of $B$ in $T$, let $\mathcal{D}(B')$ and $S_{B'}(\cdot)$ denote the output of Realization($\mathcal{A}, B'$). For each child $B'$ of $B$, initialize a disk set $\mathcal{D}_{Pr}(B')$ which is a copy of $\mathcal{D}(B')$. For each sub-box $U$ of $B$, Do:
    4.1 If $U$ does not cover any sub-box of a child of $B$. Pick arbitrarily $\Phi_B(U) + \Phi_{loc}(B, U)$ server points in $U$ and put them into $S_{loc}(U)$.
    4.2 Else: Let $U_1, U_2, \ldots, U_{\Gamma^d}$ be the child sub-boxes of $U$. For each $U_i$, let $B'_1, B'_2, \ldots, B'_t$ (the order can be arbitrary) be the children of $B$ such that $U_i$ is one of their sub-boxes, Do:
        • For each $B'_j$, $j = 2, 3, \ldots, t$, call Re-Coordination($\mathcal{D}_{Pr}(B'_j), U_i, S_{B'_j}(U_i), S_{B'_1}(U_i)$).
        • Put all points of $S_{B'_1}(U_i)$ into $S_{loc}(U)$.
5: For every A-Disk $A$ in $\mathcal{A}(B)$, let $U$ be its center, realize $A$ using a point in $S_{loc}(U)$ as the center, and put the resulting disk into $\mathcal{D}_{loc}$. Avoid using the same server point multiple times in the realization.
6: For sub-box $U$ of $B$, let $S_B(U)$ be the set of server points in $S_{loc}(U)$ that are not used for the realization in the last step. Output $S_B(U)$.
7: Set $\mathcal{D}(B)$ as $\mathcal{D}_{loc} \cup (\bigcup_i \mathcal{D}_{Pr}(B_i))$.
8: If there are multiple disks in $\mathcal{D}(B)$ that use a same server point as their centers, only the disk with the largest radius is kept and the others are removed from $\mathcal{D}(B)$. Output $\mathcal{D}(B)$
```

Algorithm 2 Realization-Leaf($\mathcal{A}, B$)

```
Input: A-Disk multi-set $\mathcal{A}$ that is a Balanced AD-cover for $(S, C, k)$, a Leaf node $B$ of $T$.
Output: A set $\mathcal{D}(B)$ of $\mathbb{R}^d$ disks. A set $S_B(U)$ of $\Phi_B(U)$ server points for every sub-box $U$ of $B$, with none of them used as a center for a disk in $\mathcal{D}(B)$. No multiple disks in $\mathcal{D}(B)$ share the same server point as center.
1: Initialize an empty disk set $\mathcal{D}$.
2: For every A-Disk $A$ in $\mathcal{A}(B)$, create a disk $D$ as the realization of $A$ and put it into $\mathcal{D}$. During this process, avoid using the same server point multiple times for the realization.
3: For every sub-box $U$ of $B$, choose $\Phi_B(U)$ distinct server points in $U$ and avoid those already used during the realization process. Put these server points into $S_B(U)$.
4: Output $\mathcal{D}$ and $S_B(U)$.
```

Algorithm 3 Re-Coordination($\mathcal{D}_{Pr}(B'), U', S', S_1$)

```
Input: A disk set $\mathcal{D}_{Pr}(B')$ generated from step 4 of Algorithm 1. A sub-box $U'$ of $B'$. $S'$ and $S_1$: two sets of server points in $U'$, both with a cardinality of $\Phi_{B'}(U')$. Every point in $S'$ is not a center of a disk in $\mathcal{D}_{Pr}(B')$.
Output: Change the center of some of the disks in $\mathcal{D}_{Pr}(B')$, so that no disk in $\mathcal{D}_{Pr}(B')$ use any point in $S_1$ as the center.
1: Find an arbitrary bijective mapping $\xi : S_1 \setminus S' \to S' \setminus S_1$.
2: While: There exists a disk $D$ in $\mathcal{D}_{Pr}(B')$ whose center is a point $s$ in $S_1 \setminus S'$, DO:
    • Expand and recenter $D$ in $\mathcal{D}_{Pr}(B')$ to the minimum disk that is centered at $\xi(s)$ and contains $D$ in its interior.
```

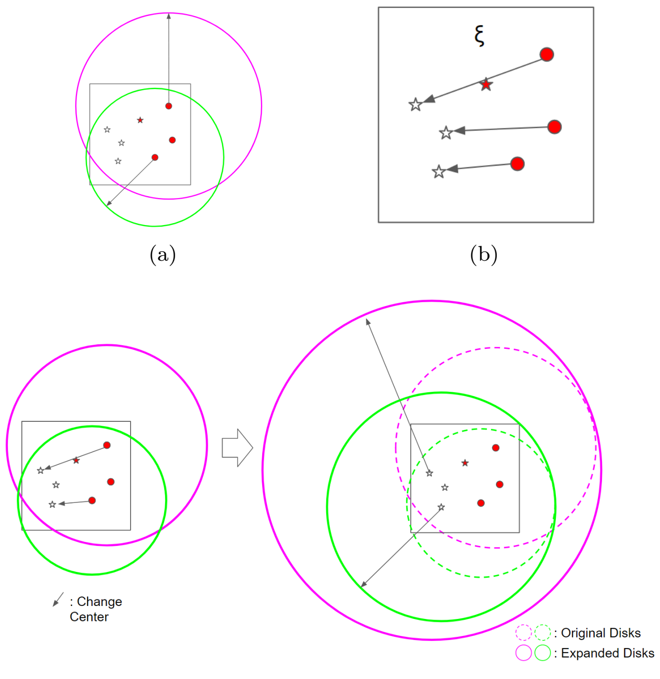

Figure 4: Illustration of Algorithm 3, which changes the center of disks in $D_{Pr}(B')$ so that none of them will be centered at a point in S_1. (a) A sub-box U' contains two sets of server points S_1 (in red) and S' (stars). Two disks in $D_{Pr}(B')$ are centered at points in $S' \setminus S_1$. (b) A bijective mapping ξ that maps $S_1 \setminus S'$ to $S' \setminus S_1$. ξ is used to determine the center of the expanding disk. (c) Disk expansion under ξ. After Re-Coordination, no point in S_1 (in red) is the center of any disk in $D_{Pr}(B')$.

Let B be a node of T and U be a sub-box of B. We denote by $K_{loc}(B,U)$ the number of A-Disks in $\mathcal{A}(B)$ that cover U. Note that by definition, we know that if $\mathcal{A}$ is a balanced AD-Cover, it is possible to assign a mapping K_B to every node B of T, such that K_B maps every inner sub-box U of B to an integer $0 \le K_B(U) \le m$, and the following hold.

- $K_{B_r}(U) = 0$ for any inner sub-box U of the root B_r.
- $\mathcal{A}^*(B)$ is a K_B-AD-cover of B.
- $K_B(U') = \min(K_B(U) + K_{loc}(B,U), m)$, where U' is a child sub-box of U and an inner sub-box of a child B' of B in T.

The mapping K_B is a key to the following analysis. The following definition and lemma imply that the resulting disk set of the Realization algorithm is a disk k-cover, and thus ensure the correctness of the algorithm.

Definition 8. *A disk set $\mathcal{D}$ is a K_B-**D-Cover** if for any client point c that lies in an inner sub-box U of B, there exists at least $k(c) - K_B(U)$ disks in $\mathcal{D}$ that are centered at distinct server points and cover c.*

Lemma 5.1. *$\mathcal{D}(B)$ is a K_B-D-Cover*

Proof. We prove this lemma by induction. We first consider the case where B is a leaf node of T. $\mathcal{D}(B)$ is generated in Algorithm 2 by performing a realization operation on every A-Disk in $\mathcal{A}(B)$. It is clear that no two disks in $\mathcal{D}(B)$ use the same server point for their realizations, since the algorithm ensures that each server point is never used multiple times. Note that $\mathcal{A}^*(B) = \mathcal{A}(B)$ is a K_B-AD-Cover. This means that for any client point c in B, we have $K_B(U) + K_{loc}(B,U) \ge k(c)$, where U is the inner sub-box of B containing c. Thus, $K_{loc}(B,U) \ge k(c) - K_B(U)$. Since $K_{loc}(B,U)$ is the number of A-Disks in $\mathcal{A}(B)$ that covers U, and an A-Disk is fully covered by its realization, therefore it is possible to find at least $K_{loc}(B,U)$ disks in $\mathcal{D}(B)$ (with distinct centers) that cover U, and thus cover c. The disks in $\mathcal{D}$ uses different server points as centers. The lemma for the case where B is a leaf node then follows.

Now, we consider the case where B is an internal node. We assume by induction that for any child B' of B, $\mathcal{D}(B')$ is a $K_{B'}$-D-Cover. Recall that in step 4 Algorithm 1, a set $\mathcal{D}_{Pr}(B')$ is defined for B' and initialized as $\mathcal{D}(B')$. After the call to Re-Coordination in step 4.2 of Algorithm 1, $\mathcal{D}_{Pr}(B')$ remains a $K_{B'}$-D-Cover, since the Re-Coordination always expand a disk in $\mathcal{D}_{Pr}(B')$ to one that fully covers the original disk, and the Re-Coordination ensures that disks in $\mathcal{D}_{Pr}(B')$ would continue to have different centers.

Consider any client point c that lies in an inner sub-box U of B. Let U' be the child sub-box of U that contains U' and B' be the child of B such that U' is an inner sub-box of B'. Let V_1 be the number of disks in $\mathcal{D}_{Pr}(B')$ (after step 4.2) with distinct centers that cover c, and V_2 be the number of disks in $\mathcal{D}_{loc}$ (after step 5) that cover c. The lemma follows if we can prove that $V_1 + V_2 \ge k(c) - K_B(U)$.

From the induction hypothesis, we know that $\mathcal{D}_{Pr}(B')$ is a $K_{B'}$-D-Cover. Thus, $V_1 \ge k(c) - K_{B'}(U') \ge k(c) - (K_B(U) + K_{loc}(B,U))$. In step 5, a realization operation is performed on every A-Disk in $\mathcal{A}(B)$ that uses U as the center and the generated disk is put into $\mathcal{D}_{loc}$. This means that there are at least $K_{loc}(B,U)$ disks with distinct centers in $\mathcal{D}_{loc}$ covering c, and none of them shares a same center with any disk in $\mathcal{D}_{Pr}(B')$. Thus, $V_2 \ge K_{loc}(B,U)$. Therefore, we have $V_1 + V_2 \ge k(c) - (K_B(U) + K_{loc}(B,U)) + K_{loc}(B,U) \ge k(c) - K_B(U)$, which completes the induction step and thus proves the lemma. ∎

The above lemma immediately leads to the conclusion that $\mathcal{D}(B_r)$ is a disk k-cover. This ensures the correctness of the Realization algorithm.

5.3 Cost of the Realized Disk k-Cover and Bubble Charging In this subsection, we show that the cost of $\mathcal{D}(B_r)$ is close to that of $\mathcal{A}$. This indicates that every Balanced AD-Cover can be transformed into a disk k-cover with similar cost.

The proof is based on a **bubble charging** scheme. The main idea of the bubble charging scheme is the follows. First, we know that if no disk expansion occurs, the set of disks $\mathcal{D}$ realized from the Balanced AD-Cover $\mathcal{A}$ would have a similar cost as $\mathcal{A}$ and we are done. Thus, we only need to estimate the total increased cost due to disk expansions. For this purpose, we create a **charging token** H at each time when there is a disk expansion. The token H has a cost equal to the increased cost of the expanded disk and is associated with the released server point s (in Algorithm 3) which will be used for the realization of a higher level A-Disk. The generated tokens will transfer from server points to server points (like bubbles) and monotonically increase their levels at the time when Algorithm 3 is called. The tokens associated with each server point s will eventually transfer to the disk D that uses s as the center during D's realization from some A-Disk. Since the tokens associated with s are all generated during disk expansions at lower levels, their total cost is thus small, comparing to the cost of D. Thus, we can bound the total increased cost due to disk expansions by an ϵ factor of $Cost_{OPT}$.

For ease of analysis, we make some modifications to Algorithm 1. **Algorithm 1 now also outputs a disk set $\mathcal{D}_{del}(B)$. In the beginning of Algorithm 1, $\mathcal{D}_{del}(B)$ is initialized as $\bigcup_{B' \text{ is a child of } B} \mathcal{D}_{del}(B')$, or is empty set if B is a leaf node. At the end (step 8) of the original Algorithm 1, we put all those to-be-removed disks into $\mathcal{D}_{del}(B)$.** In the following, we let $\mathcal{D}_{all}(B)$ denote $\mathcal{D}_{del}(B) \cup \mathcal{D}(B)$. Our analysis will center around estimating the cost of $\mathcal{D}_{all}(B)$.

Observe that there is a 1-1 correspondence between the disks in $\mathcal{D}_{all}(B)$ and the A-Disks in $\mathcal{A}^*(B)$. During the bottom-up execution of the Realization algorithm, when an A-Disk A is realized to produce a disk D (in step 5 of Algorithm 1 or step 2 of Algorithm 2), A is called the **original A-Disk** of D and D the **transformation** of A. Note that it is possible that in Algorithm 3, a disk D_o with original A-Disk A_o will be expanded to a larger disk with a different center. D_o is still the transformation of A_o if this happens. It is not hard to see that every A-Disk in $\mathcal{A}^*(B)$ has exactly one transformation in $\mathcal{D}_{all}(B)$, and every disk in $\mathcal{D}_{all}(B)$ has exactly one original A-Disk. Note that the transformation of an A-Disk must fully cover its realization. Thus, when a disk expansion happens (in Algorithm 3), the expanded disk always fully covers the original disk. For any disk D in $\mathcal{D}_{all}(B)$ generated by Algorithm 1, we define its **level** as the level of its original A-Disk.

In the following, we show that the cost of $\mathcal{D}_{all}(B_r)$ is close to that of $\mathcal{A}$. We first let $S_\cup(B)$ be the union of server points in $S_B(U)$ for all sub-box U of B. For analysis purpose, we augment Algorithm 1, Algorithm 2 and Algorithm 3 so that after Realization($\mathcal{A}, B$), every $s \in S_\cup(B)$ and every disk in $\mathcal{D}_{all}(B)$ are associated with a set of charging tokens. A token H has a non-negative real-valued cost $Cost(H)$ and a non-negative integer-valued level $level(H)$.

We use $\mathcal{H}(B,s)$ to denote the set of tokens associated with a server point $s \in S_\cup(B)$ after executing Realization($\mathcal{A}, B$). Similarly, we use $\mathcal{H}(B,D)$ to denote the set of tokens associated with any disk $D \in \mathcal{D}_{all}(B)$. Below we discuss how tokens are generated and transferred, and how $\mathcal{H}(B,s)$ and $\mathcal{H}(B,D)$ are determined.

1. If B is a leaf node, after the execution of Realization($\mathcal{A}, B$) (Algorithm 1), $\mathcal{H}(B,s)$ is set to be empty for every $s \in S_\cup(B)$, and $\mathcal{H}(B,D)$ is set to be empty for every $D \in \mathcal{D}_{all}(B)$.

   For the case that B is an internal node, we modify the algorithm in the following way. For every child B' of B, let $\mathcal{H}(B',D)$ be the set of tokens associated with a disk $D \in \mathcal{D}_{all}(B)$ while realizing $\mathcal{A}^*(B')$, and $\mathcal{H}(B',s)$ be the set of tokens associated with a server point $s \in S_\cup(B')$. For any server point s covered by a sub-box of B' but not in $S_\cup(B')$, initialize an empty set $\mathcal{H}(B',s)$. All such sets $\mathcal{H}(B',s)$ could be changed in the following process.

2. Modify step 2 of Algorithm 3 to allow changes to some token sets $\mathcal{H}(B',s)$: (See Figure 5 for an example of this process.)
   For: every s in $S_1 \setminus S'$, **DO:**

   - If some disk in $\mathcal{D}_{Pr}(B')$ uses s as the center, expand D in $\mathcal{D}_{Pr}(B')$ to the minimum disk that is centered at $\xi(s)$ and contains D in its interior. **(Note that this expansion does not change the token(s) $\mathcal{H}(B',D)$ associated with D.) Create a new token H with the same level as B' and a cost that equals to the increased cost of D. Add H to $\mathcal{H}(B',s)$.**

- **Move all tokens from $\mathcal{H}(B', \xi(s))$ to $\mathcal{H}(B', s)$. That is, set $\mathcal{H}(B', s) \leftarrow \mathcal{H}(B', s) \cup \mathcal{H}(B', \xi(s))$ and $\mathcal{H}(B', \xi(s)) \leftarrow \{\}$.**

3. Modify step 5 of Algorithm 1 to create a token set $\mathcal{H}_{loc}(s)$ for every $s \in S_{loc}(U)$ and every sub-box U of B:
   For every $s \in S_{loc}(U)$: **(1) if s lies in some sub-box U' of a child B' of B, set $\mathcal{H}_{loc}(s)$ to be the union of $\mathcal{H}(B', s)$ for all children B' of B. (2) Otherwise, set $\mathcal{H}_{loc}(s)$ to be an empty set.** For every A-Disk A in $\mathcal{A}(B)$, let U be its center, realize A using a point s in $S_{loc}(U)$ as the center, and put the resulting disk D in $\mathcal{D}_{loc}$. **Set $\mathcal{H}(B, D)$ to be $\mathcal{H}_{loc}(s)$.** Avoid using the same server point multiple times in the realization.

4. Update $\mathcal{H}(B, s)$ and $\mathcal{H}(B, D)$ for every $s \in S_{\cup}(B)$ and $D \in \mathcal{D}_{all}(B)$ as follows.

   (a) For every $s \in S_{\cup}(B)$, set $\mathcal{H}(B, s)$ to be $\mathcal{H}_{loc}(s)$.

   (b) For any $D \in \mathcal{D}_{all}(B)$ that is from $\mathcal{D}_{Pr}(B')$ or $\mathcal{D}_{del}(B')$ for some child B' of B, set $\mathcal{H}(B, D)$ to be $\mathcal{H}(B', D)$.

   (c) Note: For any $D \in \mathcal{D}_{all}(B)$ that is generated at step 5 of Algorithm 1 and with center s, $\mathcal{H}(B, D)$ is already set in step 5. We do need to do anything for $\mathcal{H}(B, D)$ in this case.

Let $\mathcal{H}(B)$ denote the set of tokens associated with a disk from $\mathcal{D}_{all}(B)$ or $s \in S_{\cup}(B)$ from realizing the A-Disks of B. Our scheme for token creating/transferring ensures that all tokens from $\mathcal{H}(B')$ for any child B' of B would become a token in $\mathcal{H}(B)$. In fact, any token $H \in \mathcal{H}(B', D)$ for some disk $D \in \mathcal{D}_{all}(B')$ and a child B' of B becomes a token in $\mathcal{H}(B, D)$. Any token $H \in \mathcal{H}(B', s)$ for some $s \in S_{\cup}(B')$ would eventually go to $\mathcal{H}(B', s')$ after the execution of step 4, Realization$(\mathcal{A}, B)$, where s' is some server point in $S_{loc}(U)$ and U is the sub-box of B that contains s. Such a token would eventually either be associated with a disk of $\mathcal{D}_{all}(B)$ generated in step 5 using a server point s' as the center, or be placed in $\mathcal{H}(B, s')$ if s' is not used for the realization and thus $s' \in S_B(U)$.

It is also clear that any token in $\mathcal{H}(B)$ is either (1) from $\mathcal{H}(B')$ for some child B' of B, or (2) generated in Algorithm 3 called by step 4 of 1 where a disk expansion occurs. This allows us to show the following key fact. Below, we use $Cost(\mathcal{H}) = \Sigma_{H \in \mathcal{H}} Cost(H)$ to denote the total cost of a token set $\mathcal{H}$.

LEMMA 5.2. $Cost(\mathcal{D}_{all}(B)) = Cost(\mathcal{A}^*(B)) + Cost(\mathcal{H}(B))$.

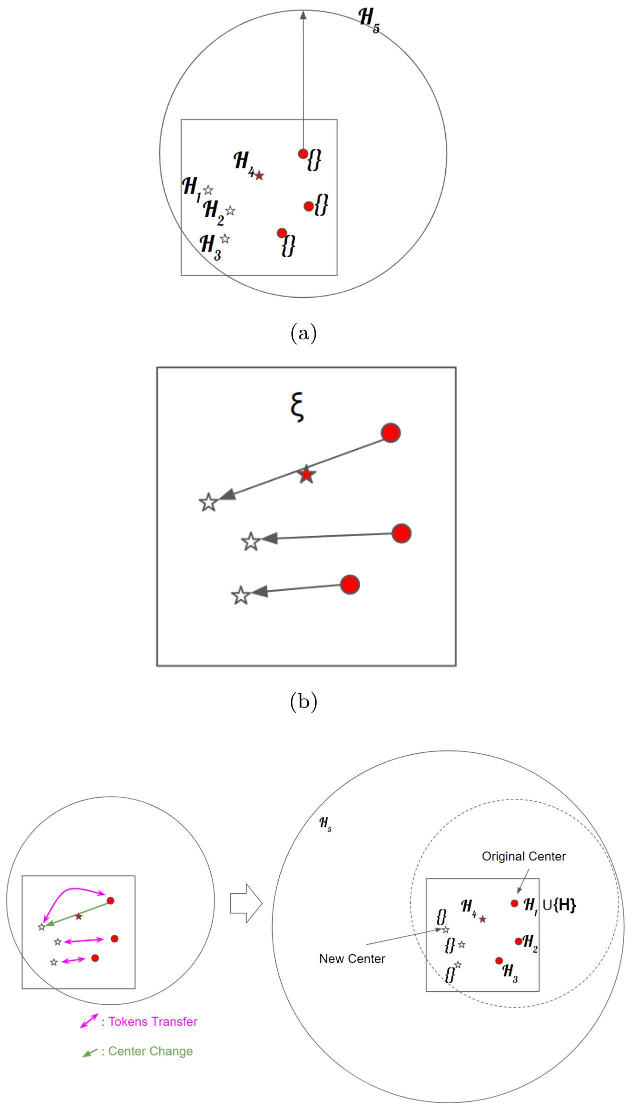

Figure 5: An illustration of the augmented Algorithm 3. (a) A sub-box U' contains two sets of server points S' (star shaped) and S_1 (in red). A disk in $D_{Pr}(B')$ centered at a point in $S' \setminus S_1$. Every server point s or disk D is associated with a set (e.g. H_1,H_2, etc) of tokens in $\mathcal{H}(B', s)$ or $\mathcal{H}(B', D)$ before the execution of Re-Coordination. For any $s \in S_1 \setminus S'$, $\mathcal{H}(B', s)$ is empty. (b) A bijective mapping ξ that maps $S_1 \setminus S'$ to $S' \setminus S_1$. ξ is used to determine the center of the expanding disk, and also the source/destination of transferring tokens. (c) Disk expansion/token transfer under ξ. After the Re-Coordination, no point in S_1 (in red) is the center of any disk in $D_{Pr}(B')$. Every s or D is associated with a token set $\mathcal{H}(B', s)$ or $\mathcal{H}(B', D)$ after the execution of Re-Coordination, which shows how tokens are transferred. A new token H is created while expanding the disk.

Proof. We prove this lemma by induction. We first consider the case where B is a leaf node. For this case, we clearly have $\text{Cost}(\mathcal{D}_{all}(B)) = Cost(\mathcal{D}(B)) = Cost(\mathcal{A}^*(B)) + \sum_{H\in\mathcal{H}(B)} Cost(H)$, since realizing A-Disks at a leaf node does not generate any token (i.e., $\sum_{H\in\mathcal{H}(B)} Cost(H) = 0$).

We now assume that B is an internal node. Let $Ch(B)$ be the set of children of B. $\mathcal{H}(B)$ can be partitioned as $\mathcal{H}(B) = (\bigcup_{B'\in Ch(B)} \mathcal{H}(B')) \cup \mathcal{H}_{exp}(B)$, where $\mathcal{H}_{exp}(B)$ is the set of tokens in $\mathcal{H}(B)$ that comes from disk expansion. Thus, we have

(5.2)
$$Cost(\mathcal{H}(B)) = (\Sigma_{B'\in Ch(B)} Cost(\mathcal{H}(B'))+Cost(\mathcal{H}_{exp}(B)).$$

$\mathcal{D}_{all}(B)$ can also be partitioned as $\mathcal{D}_{all}(B) = \mathcal{D}_{loc}(B) \cup \mathcal{D}_{chd}(B)$, where $\mathcal{D}_{loc}(B)$ is the set of disks generated in step 5 of Algorithm 1 and $\mathcal{D}_{chd}(B)$ is the set of disks from realizing the children of B (some of them might be expanded in step 4 of Algorithm 1). Since $\mathcal{D}_{loc}(B)$ is generated while realizing each A-Disk in $\mathcal{A}(B)$, thus we get $Cost(\mathcal{D}_{loc}(B)) = Cost(\mathcal{A}(B))$. Therefore, we have

$$Cost(\mathcal{D}_{all}(B)) = Cost(\mathcal{A}(B)) + Cost(\mathcal{D}_{chd}(B)). \tag{5.3}$$

Note that when a token is generated, its cost is the cost difference of the original disk and the expanded disk. Thus, we obtain $Cost(\mathcal{H}_{exp}(B)) = Cost(\mathcal{D}_{chd}(B)) - \Sigma_{B'\in Ch(B)} Cost(\mathcal{D}_{all}(B'))$. By induction hypothesis, we know that $Cost(\mathcal{D}_{all}(B')) = Cost(\mathcal{A}^*(B')) + Cost(\mathcal{H}(B'))$. Putting all together, we have

$$\begin{aligned}
&Cost(\mathcal{D}_{all}(B))\\
&= Cost(\mathcal{A}(B)) + Cost(\mathcal{D}_{chd}(B))\\
&= Cost(\mathcal{A}(B)) + Cost(\mathcal{H}_{exp}(B))+\\
&\quad \Sigma_{B'\in Ch(B)} Cost(\mathcal{D}_{all}(B'))\\
&= Cost(\mathcal{A}(B)) + Cost(\mathcal{H}_{exp}(B))+\\
&\quad \Sigma_{B'\in Ch(B)} (Cost(\mathcal{A}^*(B')) + Cost(\mathcal{H}(B')))\\
&= (Cost(\mathcal{A}(B)) + \Sigma_{B'\in Ch(B)} Cost(\mathcal{A}^*(B')))+\\
&\quad (Cost(\mathcal{H}_{exp}(B)) + \Sigma_{B'\in Ch(B)} Cost(\mathcal{H}(B')))\\
&= Cost(\mathcal{A}^*(B)) + Cost(\mathcal{H}(B)).
\end{aligned}$$

□

Lemma 5.3. *Let B be a level-l node of T. After executing Realization($\mathcal{A}, B$), for any $s \in S_\cup(B)$,*

1. *$\mathcal{H}(B, s)$ contains only tokens of level strictly larger than l;*

2. *$\mathcal{H}(B, s)$ contains at most $2^{(l'-l)d}$ tokens of level l' for any integer $l' > l$.*

Proof. We prove the lemma by induction. We first consider the case where B is a leaf node of T. The lemma for this case is trivially true, since $\mathcal{H}(B, s)$ is an empty set for any $s \in S_\cup(B)$.

Next, we consider the case where B is an internal node. Assume by induction hypothesis that for any child B' of B, the lemma holds. Any server point $s \in S_\cup(B)$ must be a point in $S_{loc}(U)$ after step 4 of Algorithm 1, where U is the sub-box of B containing s. There are two sources for s to gain tokens: (1) Generated in step 2 of Algorithm 3 during disk expansion, (2) Transferred/Inherited from some child B' of B: the token is already in $\mathcal{H}(B', s)$ before step 4 of Algorithm 1, or is transferred to $\mathcal{H}(B', s)$ from $\mathcal{H}(B', \xi(s))$ in Algorithm 3. Below, we count the number of tokens from each source.

For source (1), we first observe that this can only happen in Algorithm 3: For some child B' of B whose sub-box contains s, a token is created when expanding a disk D in $\mathcal{D}_{Pr}(B')$ (which means that D is in $\mathcal{D}(B')$) originally centered at s. There are at most 2^d such B', and in $\mathcal{D}(B')$ there is at most 1 disk centered at s. Thus, this gives at most 2^d tokens of level $l + 1$ to $\mathcal{H}(B, s)$.

Now consider source (2). Let U' be the level-$(l+1)$ sub-box that contains s. s can gain tokens from a child B' of B only if U' is a sub-box of B' and $\mathcal{H}(B', s)$ is non-empty after step 4 of Algorithm 1. Recall that there are at most 2^d children that can have U' as one of their sub-boxes. For each child B', if $\mathcal{H}(B', s)$ is non-empty after step 4 of Algorithm 1, one of the following two cases must be true: (1) s is a member of both $S'_B(U')$ (which is the parameter S' while calling Re-Coordination) and S_1 when executing Algorithm 3. The token set $\mathcal{H}(B', s)$ is not changed during this process. (2) s is in S_1 but not in $S'_B(U')$. This means that $\mathcal{H}(B', s)$ is initially empty, and obtains all the tokens from $\mathcal{H}(B', \xi(s))$ in Algorithm 3. No matter which case happens, after step 4 of Algorithm 1, the set of tokens in $\mathcal{H}(B', s)$ obtained in this manner must equal $\mathcal{H}(B', s')$ for some $s' \in S_\cup(B')$ when realizing $\mathcal{A}^*(B)$. We can apply the induction hypothesis here: for each level $l' > l + 1$, $\mathcal{H}(B, s')$ has at most $2^{(l'-l-1)d}$ tokens of level l' for any $s' \in S_\cup(B')$. Thus, $\mathcal{H}(B, s)$ can gain at most $2^{(l'-l)d}$ tokens of level l' for any $l' > l + 1$ through this way from at most 2^d of its children.

Combining the above 2 cases, we have the lemma.

□

Lemma 5.4. *Let B be a level-l node of T. After executing Realization($\mathcal{A}, B$), for any $D \in \mathcal{D}_{all}(B)$,*

1. *$\mathcal{H}(B, D)$ contains only tokens of level strictly larger than l;*

2. *$\mathcal{H}(B, D)$ contains at most $2^{(l'-l)d}$ tokens of level l'*

for any integer $l' > l$.

Proof. If B is a leaf node, the lemma is trivially true. Thus, we can assume that B is an internal node. For this case, we only need to prove that when D is generated in step 5 of Algorithm 1, it gains at most $2^{(l'-l)d}$ tokens of level l' for any integer $l' > l$. This is because in the bottom-up process of realization, once a disk is created and associated with tokens (this happens only in step 5 of Algorithm 1), the tokens will never be transferred.

To show that D gains at most $2^{(l'-l)d}$ tokens of level l' for any integer $l' > l$, we can use the same argument given in the proof of Lemma 5.3 for the server point s which is used in step 5 of Algorithm 1 as the center to realize D. Note that D gets its tokens from $\mathcal{H}_{loc}(s)$ in (the modified) step 5 of Algorithm 1, and from the argument in the proof of Lemma 5.3, it can be shown $\mathcal{H}_{loc}(s)$ contains only tokens of level strictly larger than l, and contains at most $2^{(l'-l)d}$ tokens of level l' for any integer $l' > l$. Thus, the lemma follows. ◻

LEMMA 5.5. *Let B be a node of T at level l, L_l be the edge length of a sub-box at level l, and R be the radius of a disk in $\mathcal{D}_{all}(B)$. Then, $R \le 6\sqrt{d}\Delta L_l$.*

Proof. We prove this by induction. We first consider the case that B is a leaf node. For this case, any disk $D \in \mathcal{D}_{all}(B)$ is the realization of some A-disk of B. By the definition of outer sub-boxes, it is easy to see that the distance between the centers of an outer sub-box and an inner sub-box cannot exceed $4\sqrt{d}\Delta L_l$. Thus, the realized disk has a radius no larger than $4\sqrt{d}\Delta L_l + 2\sqrt{d}L_l \le 6\sqrt{d}\Delta L_l$.

Then, we consider the case that B is an internal node. For induction hypothesis, we assume that the radius R of any disk in $\mathcal{D}_{all}(B')$, where B' is a child of B, satisfies the inequality $R \le 6\sqrt{d}\Delta L_{l+1}$, where $L_{l+1} = L_l/\Gamma$ is the edge length of any sub-box at level $l+1$. There are two cases for any disk D in $\mathcal{D}_{all}(B)$: (1) D is from $\mathcal{D}_{all}(B')$ for some child B' of B, and (2) D is created in step 5 of Algorithm 1.

For case (1), we first note that any disk D in $\mathcal{D}_{all}(B)$ cold be expanded to a larger disk in step 4.2 of Algorithm 1 while calling Algorithm 3. Let R' be the radius of D before the expansion in step 4.2. Then, $R' \le 6\sqrt{d}\Delta L_l/\Gamma$. If the expansion never occurs, then $R = R'$ and we trivially have $R \le 6\sqrt{d}\Delta L_l$. Now suppose that in step 4.2 of Algorithm 1, Algorithm 3 expands D by moving its center to a point that has a distance to the original center at most $\sqrt{d}L_l/\Gamma$. Thus, after changing the center, the radius R of D should satisfy the inequality of $R \le R' + \sqrt{d}L_l/\Gamma \le 6\sqrt{d}\Delta L_l/\Gamma + \sqrt{d}L_l/\Gamma = (6\Delta/\Gamma + 1/\Gamma)\sqrt{d}L_l$. With $\Gamma \ge 3$ and $\Delta \ge \Gamma$, clearly we have $R \le 6\sqrt{d}\Delta L_l$.

For case (2), Since D is the realization of an A-Disk of B, its radius R is thus $R \le 6\sqrt{d}\Delta L_l$ (by the same argument for the case that B is leaf node). ◻

LEMMA 5.6. *Let L_l denote the edge length of a sub-box at level l in T. Then, for any token H at level l, if $\alpha \ge 1$, $Cost(H) \le ((6\Delta+1)^\alpha - (6\Delta)^\alpha)(\sqrt{d}L_l)^\alpha$. Otherwise (i.e., $0 < \alpha < 1$), $Cost(H) \le (\sqrt{d}L_l)^\alpha$.*

Proof. Note that the cost of H at level l comes from expanding a disk $D \in \mathcal{D}_{all}(B)$ for some node B at level l. If the original disk has a radius R, then after the expansion, the radius should not exceed $R + \sqrt{d}L_l$. Therefore, the cost of H is at most $(R + \sqrt{d}L_l)^\alpha - R^\alpha$. Since $R \le 6\sqrt{d}\Delta L_l$, from the convexity/concavity of the function $f(R) = R^\alpha$, the lemma follows immediately. ◻

With the above lemmas, we can now bound the total cost of tokens associated with each disk. Let $\epsilon > 0$ be any small constant. Assume that Γ and Δ are carefully chosen constants (whose values will be determined later).

LEMMA 5.7. *For any $0 < \epsilon < 1$, it is possible to choose $\Gamma, \Delta \in O((2^d(1/\epsilon))^{O(1/\alpha)})$ in the construction of the Γ-tree T so that $\sum_{H \in \mathcal{H}(B,D)} Cost(H) \le \epsilon \cdot Cost(D)$ for any disk D in $\mathcal{D}_{all}(B)$.*

Proof. Let l_D be the level of D (recall that the level of D is the level of the original A-Disk of D), and L_{l_D} be the edge length of a sub-box at level l_D. Then, the radius of D is at least $(\Delta/2\Gamma^2)L_{l_D}$. Thus, $Cost(D) \ge (\Delta/2\Gamma^2)^\alpha L_{l_D}^\alpha$.

Let $\epsilon_0 = ((6\Delta+1)^\alpha - (6\Delta)^\alpha)(\sqrt{d})^\alpha$ if $\alpha \ge 1$ or $\epsilon_0 = (\sqrt{d})^\alpha$ if $0 < \alpha < 1$. For any token H at level $l' > l_D$, $Cost(H) \le (\epsilon_0/\Gamma^{(l'-l_D)\alpha})L_{l_D}^\alpha$. Thus, the total cost $COST_{l'}$ of tokens at level l' in $\mathcal{H}(B,D)$ is at most $(\epsilon_0 2^{(l'-l_D)d}/\Gamma^{(l'-l_D)\alpha})L_{l_D}^\alpha$. Let $\epsilon_1 = \epsilon_0/(\Delta/2\Gamma^2)^\alpha$. Then, $COST_{l'}/Cost(D) \le \epsilon_1(2^d/\Gamma^\alpha)^{l'-l_D}$. Thus, $(\sum_{H\in\mathcal{H}(B,D)} Cost(H))/Cost(D) \le \sum_{l'>l_D} \epsilon_1(2^d/\Gamma^\alpha)^{l'-l_D} = \sum_{i>0} \epsilon_1(2^d/\Gamma^\alpha)^i = \epsilon_1 \sum_{i>0}(2^d/\Gamma^\alpha)^i$.

It is not hard to see that for any $\epsilon > 0$, the term $\sum_{i>0}(2^d/\Gamma^\alpha)^i$ can the made to be $< \epsilon/3$ if we set $\Gamma > (3 \cdot 2^{d+1}/\epsilon)^{1/\alpha}$. If $\alpha < 0$, we have $\epsilon_1 = (2\Gamma^2\sqrt{d}/\Delta)^\alpha$, which can be made to be $< \epsilon/3$ if we set $\Delta > (6\sqrt{d}\Gamma^2/\epsilon)^{1/\alpha}$. Otherwise if $\alpha \ge 1$, we have $\epsilon_1 = [(6+1/\Delta)^\alpha - 6^\alpha](\Gamma^2\sqrt{d})^\alpha$, which can be made to be $< \epsilon/3$ if we set $\Delta > O(\sqrt{d}\Gamma^2/\epsilon)^{1/\alpha}$.

Thus in any case, it is possible choose values $\Delta, \Gamma \in O((2^d(1/\epsilon))^{O(1/\alpha)})$, such that $(\sum_{H\in\mathcal{H}(B,D)} Cost(H))/Cost(D) \le \epsilon_1 \sum_{i>0}(2^d/\Gamma^\alpha)^i \le \epsilon$. The lemma then follows. ◻

LEMMA 5.8. *For any $0 < \epsilon < 1$, it is possible to choose $\Gamma, \Delta \in O((2^d(1/\epsilon))^{O(1/\alpha)})$ in the construction of the Γ-tree T so that $Cost(\mathcal{D}_{all}(B_r)) \leq (1+\epsilon)Cost(\mathcal{A})$.*

Proof. From Lemma 5.2, we know that $Cost(\mathcal{D}_{all}(B_r)) = Cost(\mathcal{A}^*(B_r)) + Cost(\mathcal{H}(B_r))$. Note that $\mathcal{A} = \mathcal{A}^*(B_r)$ is a balanced AD-Cover, which means $\Phi_{B_r}(U) = 0$ for any sub-box U of B_r. Thus, $S_{B_r}(U)$ is empty. Therefore, any token in $\mathcal{H}(B_r)$ must be associated with some disk in $\mathcal{D}_{all}(B_r)$. From Lemma 5.7, we know that it is possible to choose $\Gamma, \Delta \in O((2^d(1/\epsilon))^{O(1/\alpha)})$ so that $Cost(\mathcal{H}(B_r)) \leq (1 - \frac{1}{1+\epsilon})Cost(\mathcal{D}_{all}(B_r))$. Thus, we have $Cost(\mathcal{D}_{all}(B_r)) = Cost(\mathcal{A}^*(B_r)) + Cost(\mathcal{H}(B_r)) \leq Cost(\mathcal{A}) + (1 - \frac{1}{1+\epsilon})Cost(\mathcal{D}_{all}(B_r))$. The lemma then follows. □

Since $\mathcal{D}(B_r) \subseteq \mathcal{D}_{all}(B_r)$, we immediately have the following.

COROLLARY 5.1. *Any Balanced AD-Cover of T with cost M can be transformed to a disk k-cover of cost no larger than $(1+\epsilon)M$ by Algorithm 1 with carefully chosen $\Gamma, \Delta \in O((2^d(1/\epsilon))^{O(1/\alpha)})$.*

5.4 Time Complexity Analysis In this subsection, we show that given any AD-Cover $\mathcal{A}$ on $T \in \mathcal{T}(S, C, k)$ constructed using Γ, Δ, λ, Algorithm 1 produces a disk k-cover within polynomial time (in terms of m and n).

Let B be a level-l node of T, and $\mathcal{D}(B)$, $S_B(\cdot)$ be the output of Realization($\mathcal{A}, B$). Let l_{max} be the maximum level of any leaf node in T.

LEMMA 5.9. *The number of disks in $\mathcal{D}(B)$ is at most $2m(2\Delta)^d\Gamma^{d(l_{max}-l)}$.*

Proof. Let T_B denote the subtree of T rooted at B. From the algorithm, we know that $\mathcal{D}(B)$ is the set of disks that realize the A-disks in $\mathcal{A}^*(B)$. Each disk of $\mathcal{D}(B)$ (or more precisely $\mathcal{D}_{all}$) corresponds to an original A-Disk of some node of T_B. Thus, the lemma follows if we can show that $\mathcal{A}^*(B)$ contains no more than $2m(2\Delta)^d\Gamma^{d(l_{max}-l)}$ A-Disks.

Since T_B is a Γ^d-ary tree with height $l_{max} - l$, its number of nodes is no more than $2\Gamma^{d(l_{max}-l)}$. For each node B' of T_B, and each sub-box U' of B', there are at most m A-Disk from $\mathcal{A}^*(B)$ whose center is U' (Otherwise, $\mathcal{A}$ cannot be a balanced AD-Cover). Also, the number of sub-boxes of B' is no more than $(2\Delta)^d$. Thus, for any node B' of T_B, the number of A-Disks of B' in $\mathcal{A}^*(B)$ is at most $m(2\Delta)^d$. Since the number of nodes in T_B is no more than $2\Gamma^{d(l_{max}-l)}$, the lemma thus follows. □

Next, we analyze each step of Algorithm 1.

1. **Step 1:** This step calls Algorithm 2, which realizes every A-Disks of B. It takes $O(m(2\Delta)^d)$ time, since there are at most $m(2\Delta)^d$ A-Disks of B in $\mathcal{A}$, and the realization of each A-Disk takes $O(1)$ time (recall that each A-Disk can be described by using 2 sub-boxes).

2. **Step 2:** This step takes $O((2\Delta)^d)$ time which is the number of sub-boxes of B.

3. **Step 3:** This step needs only constant time.

4. **Step 4:** This step first initializes $\mathcal{D}_{Pr}(B')$. It takes $O((2\Delta)^{O(d)}(m\rho/\lambda)^{O(d\log\Gamma)})$ time, since the number of children of B is Γ^d, which is $O(\Delta^{O(d)})$, the number of disks in $\mathcal{D}(B')$ for every child B' is $2m(2\Delta)^d\Gamma^{d(l_{max}-l)}$, and l_{max} is $O(\log(m\rho/\lambda))$ (where constant ρ is the bound factor of (S, C, k)). Then, a loop is performed to enumerate every one of the $(2\Delta)^d$ sub-boxes of B.

   - If 4.1 is executed, then it needs only $O(m)$ time, since there are at most m server points.
   - If 4.2 is executed, it runs a double loop; the outer loop iterates over all (at most $(2\Delta)^d$) sub-boxes of B', and the inner loop iterates over a set (at most Γ^d) of children of B. In each iteration, Algorithm 4 is called to change the center of some disks in $\mathcal{D}_P(B')$, whose cardinality is at most $(m\rho/\lambda)^{O(d\log\Gamma)}$.

   Thus, step 4 takes a total of $O((2\Delta)^{O(d)}(m\rho/\lambda)^{O(d\log\Gamma)})$ time.

5. **Step 5:** This step realizes a number of A-Disks and avoids using the same server point multiple times as the center. Since the number of server points is m, this step takes $O(m)$ time.

6. **Step 6:** This step determines the server point set $S_B(U)$ for every sub-box U of B. The number of sub-boxes is no more than $(2\Delta)^d$, and the total number of server points is m. Thus, this step takes $O((2\Delta)^d + m)$ time.

7. **Step 7:** This step outputs $\mathcal{D}(B)$. Since the number of disks in $\mathcal{D}(B)$ is $O((2\Delta)^{O(d)}(m\rho/\lambda)^{O(d\log\Gamma)})$, the running time of this step has thus the same bound.

8. **Step 8:** The step removes some of the disks in $\mathcal{D}(B)$. Thus, it can be done in $O((2\Delta)^{O(d)}$ $(m\rho/\lambda)^{O(d\log\Gamma)})$ time.

Also, the algorithm is run on each node or T. T is a Γ-ary tree with height $O(\log(m\rho/\lambda))$. Thus the total number of nodes is $(m\rho/\lambda)^{O(d\log\Gamma)}$.

Summarizing the above discussion, we have the following lemma.

LEMMA 5.10. *If $\mathcal{A}$ is an AD-Cover of (S,C,k) on $T \in \mathcal{T}(S,C,k)$, then Algorithm 1 generates a disk set from A within $O((2\Delta)^{O(d)}(m\rho/\lambda)^{O(d\log\Gamma)})$ time.*

6 Discretization and Balanced AD-Cover

In this section, we mainly present the following lemma, which shows that the discretization of a disk k-cover is a balanced AD-Cover.

LEMMA 6.1. *Let T be a Γ-tree in $\mathcal{T}(S,C,k)$, and $\mathcal{D}$ be a disk k-cover of (S,C,k) such that every disk in $\mathcal{D}$ fits T. Then, the discretization of $\mathcal{D}$ on T is a Balanced AD-cover of (S,C,k).*

Combining this lemma with Lemma 3.6, we have the following corollary.

COROLLARY 6.1. *Let $Cost_{OPT}$ denote the minimum cost of any disk k-cover of (S,C,k). There exists a Γ-tree T in $\mathcal{T}(S,C,k)$ and a Balanced AD-Cover $\mathcal{A}$ on T such that:*
$Cost(\mathcal{A}) \le (1+\lambda)(1+2\sqrt{d}\Gamma(\Gamma-1)/\Delta)^{\alpha}((1-(\Gamma-1)^{-1})^{d}+2^{d}(1-(1-(\Gamma-1)^{-1})^{d}))Cost_{OPT}$.

For any $\epsilon > 0$, it is possible to set $\lambda = \epsilon/2$ and find $\Gamma, \Delta \in O((2^{d}(1/\epsilon))^{O(1/\alpha)})$ which makes $(1+\lambda)(1+2\sqrt{d}\Gamma(\Gamma-1)/\Delta)^{\alpha}((1-(\Gamma-1)^{-1})^{d}+2^{d}(1-(1-(\Gamma-1)^{-1})^{d})) \le (1+\epsilon)$. Combining the above corollary with Corollary 5.1, we have the following corollary, which shows that an approximate minimum cost disk k-cover can be computed by finding a minimum cost Balanced AD-Cover $\mathcal{A}$ on a Γ-tree T in $\mathcal{T}(S,C,k)$.

COROLLARY 6.2. *Let $Cost_{OPT}$ be the minimum cost of any disk k-cover of (S,C,k). For any $\epsilon > 0$, by setting $\lambda = \epsilon/2$, it is possible to find a value for Γ and Δ in $O((2^{d}(1/\epsilon))^{O(1/\alpha)})$ and build $\mathcal{T}(S,C,k)$ based on λ,Γ,Δ such that there exists a Balanced AD-Cover $\mathcal{A}$ whose cost is the minimum among all possible Balanced AD-Covers for trees in $\mathcal{T}(S,C,k)$ and whose realization $\mathcal{D}$ has a cost $Cost(\mathcal{D}) \le (1+\epsilon)Cost_{OPT}$.*

7 Finding Minimum Cost Balanced AD-Cover via Dynamic Programming

In this section, we show that given a Γ-Tree T for an MCMC instance (S,C,k), how to find a Balanced AD-Cover for (S,C,k) on T with minimum cost. The obtained Balanced AD-Cover (which includes a pair of mappings Φ_B and η_B for every node B of T) is then used to call the Realization algorithm to yield the desired disk k-cover of (S,C,k). Below we assume that T is from $\mathcal{T}(S,C,k)$ constructed with certain constants Γ, Δ and λ.

To find a Balanced AD-Cover, we consider any node B of T, and show how to obtain a minimum cost (K,Φ)-AD-Cover for B in a bottom-up manner. The (K,Φ)-AD-Cover for the root B_r will then be the Balanced AD-Cover. For this purpose, we let $\mathcal{U}_{-}(B)$ denote the set of sub-boxes with each of them being a child sub-box of some sub-box of B, and η_B be a mapping that maps every sub-box in $\mathcal{U}_{-}(B)$ to an integer in $\{0,1,2,\ldots,m\}$. Denote by $\mathbb{A}(B,\Phi)$ the set of all possible multi-sets $\mathcal{A}$ of A-Disks of B that satisfy the following conditions: for every sub-box U of B, the number of A-Disks in $\mathcal{A}$ that use U as the center is no larger than $w(U)-\Phi(U)$. Let $B_1, B_2, \ldots B_{\Gamma^d}$ be the children of B in T, $\Phi_1, \Phi_2, \ldots\Phi_{\Gamma^d}$ be a set of mappings with each Φ_i mapping every sub-box of B_i to an integer in $\{0,1,2,\ldots,m\}$, and $K_1, K_2, \ldots K_{\Gamma^d}$ be a set of mappings with each K_i mapping every sub-box of B_i to an integer in $\{0,1,2,\ldots,n\}$. For any $\mathcal{A} \in \mathbb{A}(B,\Phi)$ and any sub-box U of B, we use $\Phi_{loc}(\mathcal{A},U)$ to denote the number of A-Disks in $\mathcal{A}$ that are centered at U. If U is an inner sub-box of B, we also use $K_{loc}(\mathcal{A},U)$ to denote the number of A-Disks in $\mathcal{A}$ that cover U. We say that the tuple $(K,\Phi,\eta_B,\mathcal{A};K_1,K_2,\ldots,K_{\Gamma^d};\Phi_1,\Phi_2,\ldots\Phi_{\Gamma^d})$ is *locally balanced* at B if the following hold.

- For any inner sub-box U of B and any $U' \sqsubset U$ that is an inner sub-box of a child B' of B, $K_i(U') = K(U) + K_{loc}(\mathcal{A},U)$.
- For every sub-box U of B, $\sum_{U' \sqsubset U} \eta_B(U') - \Phi_{loc}(\mathcal{A},U) = \Phi(U)$.
- For any $U' \in \mathcal{U}_{-}(B)$ that is a sub-box of some child of B, $\Phi_i(U') = \eta_B(U')$.
- For any sub-box U of B, $\Phi(U)+\Phi_{loc}(\mathcal{A},U) \le w(U)$.

The following 2 lemmas are directly obtained from the definition of (K,Φ)-AD-Cover.

LEMMA 7.1. *Let $\mathcal{A}^*$ be an A-Disk multi-set, and $\mathcal{A}$ be the set of A-Disks of B in $\mathcal{A}^*$. $\mathcal{A}^*$ is a (K,Φ)-AD-Cover of an internal node B if and only if it is possible to find $\eta_B;K_1,K_2,\ldots,K_{\Gamma^d};\Phi_1,\Phi_2,\ldots\Phi_{\Gamma^d}$ satisfying the following conditions.*

- *The tuple $(K,\Phi,\eta_B,\mathcal{A};K_1,K_2,\ldots,K_{\Gamma^d};\Phi_1,\Phi_2,\ldots\Phi_{\Gamma^d})$ is locally balanced at B.*
- *$\mathcal{A}_i$ is a (K_i,Φ_i)-AD Cover of B_i for each $i = 1,2,\ldots,\Gamma^d$, where $\mathcal{A}_i$ denotes the subset of $\mathcal{A}^*$ that contains all the A-Disks of B_i or its descendants.*

LEMMA 7.2. *If the tuple $(K,\Phi,\eta_B,\mathcal{A};K_1,K_2,\ldots,K_{\Gamma^d};\Phi_1,\Phi_2,\ldots\Phi_{\Gamma^d})$ is locally balanced at B, and $\mathcal{A}_i$ is a (K_i,Φ_i)-AD Cover of B_i for each $i = 1,2,\ldots,\Gamma^d$, then $(\bigcup_{i=1,2,\ldots,\Gamma^d}\mathcal{A}_i)\cup\mathcal{A}$ is a (K,Φ)-AD-Cover of B.*

The above two lemmas suggest that the problem of finding a minimum cost (K, Φ)-AD-Cover of B can be reduced to the problem of finding a minimum cost (K, Φ)-AD-Cover for each child of B. This indicates that we can obtain a Balanced AD-Cover through the following dynamic programming scheme. Particularly, we construct a table $DP[B, K, \Phi]$ and fill each entry with the information that allows us to easily obtain a minimum cost (K, Φ)-AD-Cover of B. Each entry is indexed by a tuple $[B, K, \Phi]$, where

1. B is a node of T.
2. K is a mapping from the set $\mathcal{I}(B)$ of inner sub-boxes of B to $\{0, 1, 2, \ldots, m\}$.
3. Φ is a mapping from the set $\mathcal{S}(B)$ of sub-boxes of B to $\{0, 1, 2, \ldots, m\}$.

Given any index tuple $[B, K, \Phi]$, the corresponding table entry, denoted by $DP[B, K, \Phi]$, has the following information:

1. A non-negative real number $C[B, K, \Phi]$, which is the minimum cost of a (K, Φ)-AD-Cover of B. The field is set to be infinity if there is no (K, Φ)-AD-Cover of B.
2. A multi-set $\mathcal{A}[B, K, \Phi]$ of the A-Disks of B. If B is a leaf node, $\mathcal{A}[B, K, \Phi]$ is a minimum cost (K, Φ)-AD-Cover of B. Otherwise (i.e., B is an internal node), $\mathcal{A}[B, K, \Phi]$ is the A-Disk multi-set satisfies the conditions listed in Item 3.
3. Mappings $\eta[B, K, \Phi]$, $K_1[B, K, \Phi]$, $K_2[B, K, \Phi]$, $\ldots$, $K_{\Gamma^d}[B, K, \Phi]$, $\Phi_1[B, K, \Phi], \ldots, \Phi_{\Gamma^d}[B, K, \Phi]$ (only when B is internal) such that
   - The tuple $(K, \Phi, \eta[B, K, \Phi], \mathcal{A}[B, K, \Phi];$ $K_1[B, K, \Phi], K_2[B, K, \Phi], \ldots, K_{\Gamma^d}[B, K, \Phi];$ $\Phi_1[B, K, \Phi], \ldots, \Phi_{\Gamma^d}[B, K, \Phi])$ is locally balanced at B.
   - If a (K, Φ)-AD-Cover of B exists, these mappings will allow us to determine a minimum cost (K, Φ)-AD-Cover of B recursively as follows. Assume that a minimum cost $(K_i[B, K, \Phi], \Phi_i[B, K, \Phi])$-AD-Cover $\mathcal{A}_i$ is already available for every child B_i of B, $i = 0, 1, \ldots, \Gamma^d$. Then, the A-Disk multi-set $(\bigcup_{i=1,2,\ldots,\Gamma^d} \mathcal{A}_i) \cup \mathcal{A}[B, K, \Phi]$ is a minimum cost (K, Φ)-AD-Cover of B.

Below we show that the dynamic programming table can be filled in polynomial time. We first demonstrate that the size of the table is polynomial in terms of n and m (note that Γ and Δ are constants). T is a Γ^d-ary tree with a height of $O(\log(m\rho/\lambda))$. Thus, the total number of nodes in the tree is $O((m\rho/\lambda)^{O(d \log \Gamma)})$. For any node B of T, the total number of mappings from $\mathcal{I}(B)$ to $\{0, 1, 2, \ldots, m\}$ is $O(m^{\Delta^d})$. The total number of mappings from $\mathcal{O}(B)$ to $\{0, 1, 2, \ldots, m\}$ is $O(m^{\Delta^{O(d)}})$. Thus, we have the following lemma.

Lemma 7.3. *The total number of entries in* $DP[B, K, \Phi]$ *is* $O((m\rho/\lambda)^{O(d \log \Gamma)} m^{\Delta^{O(d)}})$.

Below we present a method to fill the entries of $DP[B, K, \Phi]$ in a bottom-up manner. We consider separately the case where B is a leaf node and the case where B is an internal node of T. For the latter case, we assume that the entry $[B_i, K_i, \Phi_i]$ for any child B_i of B has already been filled for all possible mappings K_i and Φ_i.

Filling entry $[B, K, \Phi]$ for a leaf node B: Try all possible A-Disk multi-sets $\mathcal{A} \in \mathbb{A}(B, \Phi)$ and let $\mathcal{A}[B, K, \Phi]$ be the one that is a minimum cost (K, Φ)-AD-Cover of B. Set $C[B, K, \Phi]$ to be the cost of $\mathcal{A}[B, K, \Phi]$. If no such an $\mathcal{A}[B, K, \Phi]$ exists (i.e. there does not exist a (K, ϕ)-AD-Cover of B), $C[B, K, \Phi]$ is set to be positive infinity, and $\mathcal{A}[B, K, \Phi]$ is set to be NULL.

Lemma 7.1 and Lemma 7.2 suggest the following approach to find a minimum cost (K, Φ)-AD-Cover for any internal node B.

Filling entry $[B, K, \Phi]$ for an internal node B: Enumerate all possible mappings $\eta_B; K_1, K_2, \ldots,$ $K_{\Gamma^d}; \Phi_1, \Phi_2, \ldots \Phi_{\Gamma^d}$ and $\mathcal{A} \in \mathbb{A}(B, \Phi)$ that make the tuple $(K, \Phi, \eta_B, \mathcal{A}; K_1, K_2, \quad \ldots, K_{\Gamma^d}; \Phi_1, \Phi_2, \ldots \Phi_{\Gamma^d})$ locally balanced. Among all such tuples, identify the one that minimizes $Cost(\mathcal{A}) + \sum_{i=1}^{\gamma^d} DP[B_i, K_i, \Phi_i]$. Set $\eta[B, K, \Phi]; K_1[B, K, \Phi], \ldots, K_{\Gamma^d}[B, K, \Phi]; \Phi_1[B, K, \Phi],$ $\ldots, \Phi_{\Gamma^d}[B, K, \Phi], \mathcal{A}[B, K, \Phi]$ accordingly and $C[B, K, \Phi]$ to be the minimum value of $Cost(\mathcal{A}) + \sum_{i=1}^{\gamma^d} DP[B_i, K_i, \Phi_i]$. If no such a tuple exists, set $C[B, K, \Phi]$ to be infinity and $\eta[B, K, \Phi];$ $K_1[B, K, \Phi], \ldots, K_{\Gamma^d}[B, K, \Phi]; \Phi_1[B, K, \Phi], \ldots \Phi_{\Gamma^d}[B, K, \Phi],$ $\mathcal{A}[B, K, \Phi]$ to be NULL.

The following lemma bounds the cardinality of $\mathbb{A}(B, \Phi)$.

Lemma 7.4. $|\mathbb{A}(B, \Phi)|$ *is* $O(m^{\Delta^{O(d)}})$.

Proof. We estimate the number of possible ways to construct a different A-Disk multi-set $\mathcal{A}$ such that for every sub-box U of B, the number of A-Disks in $\mathcal{A}$ that use U as the center is no larger than $w(U)-\Phi(U)$, where $w(U)$ is the number of server points in U. We use $\mathcal{A}_U$ to denote the set of A-Disks in $\mathcal{A}$ that uses U as the center. Thus, $\mathcal{A}$ can be partitioned as $\mathcal{A} = \bigcup_{U\in\mathcal{S}(B)} \mathcal{A}_U$. The process of constructing $\mathcal{A}$ can be viewed as first building $\mathcal{A}_U$ for each U and then concatenating them.

We claim that for each U, the number of different ways to construct $\mathcal{A}_U$ is $O(m^{\Delta^{O(d)}})$. To see this, we first observe that it takes two sub-boxes, U and another sub-box $U' \in \mathcal{S}(B)$, to jointly determine an A-Disk that is centered at U. Since there are a total of $O(\Delta^{O(d)})$ sub-boxes in $\mathcal{S}(B)$, the number of distinct A-Disks in $\mathcal{A}_U$ is thus $O(\Delta^{O(d)})$. This means that the number of ways to construct $\mathcal{A}_U$ can be viewed as the number of ways to establish a mapping $P(U')$ for each sub-box $U' \in \mathcal{S}(B)$, where $P(U')$ represents the number of copies of the A-Disk determined by U' and U that will be added to the multi-set $\mathcal{A}_U$. Since $\mathcal{A}_U \leq m$, $P(U')$ can take values only from $\{0, 1, 2, \ldots, m\}$. Thus, the total number of different mappings for all sub-boxes $U' \in \mathcal{S}_B$ is $O(m^{\Delta^{O(d)}})$.

Since the number of ways to construct $\mathcal{A}_U$ is $O(m^{\Delta^{O(d)}})$ for each sub-box U, and the total number of sub-boxes is $O(\Delta^{O(d)})$, the number of ways to construct $\mathcal{A} = \bigcup_{U\in\mathcal{S}(B)} \mathcal{A}_U$ is thus $O(m^{\Delta^{O(d)}})$. ◻

With the above lemma, we can now bound the time needed for filling one entry of the dynamic programming table.

Lemma 7.5. *Each entry $[B, K, \Phi]$ can be filled in $O(nm^{\Delta^{O(d)}})$ time.*

Proof. If B is a leaf node, the process of filling the entry $[B, K, \Phi]$ needs to first enumerate all A-Disk multi-sets in $|\mathbb{A}(B, \Phi)|$, and then test each of them to see which one is the minimum cost K-AD-Cover. Since there are $O(m^{\Delta^{O(d)}})$ such multi-sets, and it takes at most $O(nm\Delta^{O(d)})$ time to determine whether a given multi-set (of size at most m) is a K-AD-Cover. Thus, it takes $O(nm^{\Delta^{O(d)}})$ time if B is a leaf node.

If B is an internal node, the process needs to enumerate all possible mappings $\eta; K_1, K_2, \ldots, K_{\Gamma^d}; \Phi_1, \Phi_2, \ldots\Phi_{\Gamma^d}$ and $\mathcal{A} \in \mathbb{A}(B, \Phi)$. The number of different mappings for η is $O(m^{\Delta^{O(d)}})$, since there are $O(\Delta^{O(d)})$ sub-boxes, and η maps each sub-box to an integer in $\{0, 1, 2, \ldots, m\}$. By the same argument, we know that for each Φ_i or K_i, the number of different mappings is $O(m^{\Delta^{O(d)}})$. Thus, the number of combinations for $\eta; K_1, K_2, \ldots, K_{\Gamma^d}; \Phi_1, \Phi_2, \ldots\Phi_{\Gamma^d}$ is $O(m^{\Delta^{O(d)}})$. For each of such combinations, we need to determine whether the tuple $(K, \Phi, \eta, \mathcal{A}; K_1, K_2, \ldots, K_{\Gamma^d}; \Phi_1, \Phi_2, \ldots\Phi_{\Gamma^d})$ is locally balanced at B. A straightforward way of testing takes no more than $O(nm\Delta^{O(d)})$ time. Thus, it is possible to fill the entry in $O(nm^{\Delta^{O(d)}})$ time for an internal node B. ◻

Combining Lemma 7.5 and Lemma 7.3, we can bound the total time for the dynamic programming process.

Lemma 7.6. *Given a Γ-tree T of a (S, C, k) instance of MCMC, a minimum cost Balanced AD-cover on T can be found in $O(n(m\rho/\lambda)^{O(d\log\Gamma)} m^{\Delta^{O(d)}})$ time.*

Combining this lemma with previous discussions, we have the following.

1. By setting Γ, Δ to be $O((2^d(1/\epsilon))^{O(1/\alpha)})$, and $\lambda = \epsilon/2$, the minimum cost Balanced AD-Cover of one of the Γ-trees in $\mathcal{T}(S, C, k)$ can be transformed to a $(1+\epsilon)$-approximate disk k-cover of (S, C, k).

2. The transformation takes $O((2\Delta)^{O(d)}(m\rho/\lambda)^{O(d\log\Gamma)})$ time, where ρ is a fixed constant.

3. There are Γ^d such Γ-trees.

4. For each tree T, a minimum cost Balanced AD-Cover can be found in $O(n(m\rho/\lambda)^{O(d\log\Gamma)} m^{\Delta^{O(d)}})$ time.

This leads to the following theorem. Note that so far our discussion assumes that (S, C, k) is a bounded instance.

Theorem 7.1. *Given a bounded MCMC instance (S, C, k), it is possible to find a $(1+\epsilon)$-approximate minimum cost k-cover in $O(nm^{O(1/\epsilon)^{O(d/\alpha)}})$ time.*

Finally, it can be shown that every MCMC instance (S, C, k) is decomposable to to at most m bounded instances in $(mn)^{O(1)}$ time. This immediately leads to Theorem 1.1.

References

[1] Vittorio Bilò, Ioannis Caragiannis, Christos Kaklamanis, and Panagiotis Kanellopoulos. Geometric clustering to minimize the sum of cluster sizes. In *European Symposium on Algorithms*, pages 460–471. Springer, 2005.

[2] Nissan Lev-Tov and David Peleg. Polynomial time approximation schemes for base station coverage with minimum total radii. *Computer Networks*, 47(4):489–501, 2005.

[3] Moses Charikar and Rina Panigrahy. Clustering to minimize the sum of cluster diameters. *Journal of Computer and System Sciences*, 68(2):417–441, 2004.

[4] Helmut Alt, Esther M Arkin, Hervé Brönnimann, Jeff Erickson, Sándor P Fekete, Christian Knauer, Jonathan Lenchner, Joseph SB Mitchell, and Kim Whittlesey. Minimum-cost coverage of point sets by disks. In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 449–458. ACM, 2006.

[5] A Karim Abu-Affash, Paz Carmi, Matthew J Katz, and Gila Morgenstern. Multi cover of a polygon minimizing the sum of areas. *International Journal of Computational Geometry & Applications*, 21(06):685–698, 2011.

[6] Santanu Bhowmick, Kasturi Varadarajan, and Shi-Ke Xue. A constant-factor approximation for multi-covering with disks. In *Proceedings of the twenty-ninth annual symposium on Computational geometry*, pages 243–248, 2013.

[7] Santanu Bhowmick, Tanmay Inamdar, and Kasturi Varadarajan. On metric multi-covering problems. *arXiv preprint arXiv:1602.04152*, 2016.

[8] Matt Gibson, Gaurav Kanade, Erik Krohn, Imran A Pirwani, and Kasturi Varadarajan. On clustering to minimize the sum of radii. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 819–825, 2008.

[9] Chandra Chekuri, Kenneth L Clarkson, and Sariel Har-Peled. On the set multicover problem in geometric settings. *ACM Transactions on Algorithms (TALG)*, 9(1):9, 2012.

[10] Kasturi Varadarajan. Weighted geometric set cover via quasi-uniform sampling. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 641–648. ACM, 2010.

[11] Timothy M Chan, Elyot Grant, Jochen Könemann, and Malcolm Sharpe. Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1576–1585. Society for Industrial and Applied Mathematics, 2012.

[12] Nikhil Bansal and Kirk Pruhs. Weighted geometric set multi-cover via quasi-uniform sampling. In *European Symposium on Algorithms*, pages 145–156. Springer, 2012.

[13] Menghong Li, Yingli Ran, and Zhao Zhang. Approximation algorithms for the minimum power partial cover problem. In *International Conference on Algorithmic Applications in Management*, pages 179–191. Springer, 2019.

[14] Pradeesha Ashok, Aniket Basu Roy, and Sathish Govindarajan. Local search strikes again: Ptas for variants of geometric covering and packing. *Journal of Combinatorial Optimization*, 39(2):618–635, 2020.

[15] Sariel Har-Peled and Mira Lee. Weighted geometric set cover problems revisited. *JoCG*, 3(1):65–85, 2012.

[16] Jian Li and Yifei Jin. A ptas for the weighted unit disk cover problem. In *International Colloquium on Automata, Languages, and Programming*, pages 898–909. Springer, 2015.

[17] Nabil H Mustafa, Rajiv Raman, and Saurabh Ray. Quasi-polynomial time approximation scheme for weighted geometric set cover on pseudodisks and halfspaces. *SIAM Journal on Computing*, 44(6):1650–1669, 2015.

[18] Pankaj K Agarwal and Jiangwei Pan. Near-linear algorithms for geometric hitting sets and set covers. In *Proceedings of the thirtieth annual symposium on Computational geometry*, pages 271–279, 2014.

[19] Sada Narayanappa and Petr Vojtechovskỳ. An improved approximation factor for the unit disk covering problem. In *CCCG*, 2006.

[20] Tomás Feder and Daniel Greene. Optimal algorithms for approximate clustering. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 434–444. ACM, 1988.

[21] Dorit S Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *Journal of the ACM (JACM)*, 32(1):130–136, 1985.

[22] Timothy M Chan and Elyot Grant. Exact algorithms and apx-hardness results for geometric packing and covering problems. *Computational Geometry*, 47(2):112–124, 2014.

[23] Nabil H Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010.

[24] Shashidhara K Ganjugunte. *Geometric hitting sets and their variants*. PhD thesis, Duke University, 2011.

[25] Saurabh Ray. Weak and strong epsilon-nets for geometric range spaces. 2009.

[26] Ari Freund and Dror Rawitz. Combinatorial interpretations of dual fitting and primal fitting. In *International Workshop on Approximation and Online Algorithms*, pages 137–150. Springer, 2003.

[27] Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric graphs. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 671–679. Society for Industrial and Applied Mathematics, 2001.

[28] Reuven Bar-Yehuda and Dror Rawitz. A note on multicovering with disks. *Computational Geometry*, 46(3):394–399, 2013.