FPT Approximation for Capacitated Sum of Radii

Ragesh Jaiswal
CSE, IIT Delhi, India

Amit Kumar
CSE, IIT Delhi, India

Jatin Yadav[1]
CSE, IIT Delhi, India

Abstract

We consider the capacitated clustering problem in general metric spaces where the goal is to identify k clusters and minimize the sum of the radii of the clusters (we call this the Capacitated k-sumRadii problem). We are interested in fixed-parameter tractable (FPT) approximation algorithms where the running time is of the form $f(k) \cdot \mathsf{poly}(n)$, where $f(k)$ can be an exponential function of k and n is the number of points in the input. In the uniform capacity case, Bandyapadhyay et al. recently gave a 4-approximation algorithm for this problem. Our first result improves this to an FPT 3-approximation and extends to a constant factor approximation for any L_p norm of the cluster radii. In the general capacities version, Bandyapadhyay et al. gave an FPT 15-approximation algorithm. We extend their framework to give an FPT $(4 + \sqrt{13})$-approximation algorithm for this problem. Our framework relies on a novel idea of identifying approximations to optimal clusters by carefully pruning points from an initial candidate set of points. This is in contrast to prior results that rely on guessing suitable points and building balls of appropriate radii around them.

On the hardness front, we show that assuming the Exponential Time Hypothesis, there is a constant $c > 1$ such that any c-approximation algorithm for the non-uniform capacity version of this problem requires running time $2^{\Omega\left(\frac{k}{polylog(k)}\right)}$.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases Approximation algorithm, parameterized algorithm, clustering

Digital Object Identifier 10.4230/LIPIcs.ITCS.2024.65

Related Version *Full Version*: `https://arxiv.org/abs/2401.06714`

Funding *Ragesh Jaiswal*: The author acknowledges the support from the SERB, MATRICS grant.

1 Introduction

Center-based clustering problems are important data processing tasks. Given a metric D on a set of n points $\mathcal{X}$ and a parameter k, the goal here is to partition the set of points into k *clusters*, say $C_1, \ldots, C_k$, and assign the points in each cluster to a corresponding *cluster center*, say $c_1, \ldots, c_k$, respectively. Some of the most widely studied objective functions are given by the k-center, the k-median and the k-means problems. The k-center problem seeks to find a clustering such that the maximum radius of a cluster is minimized. Here, radius of a cluster is defined as the farthest distance between the center of the cluster and a point in the same cluster. Another important center based objective function is the k-sumRadii problem where the goal is to minimize the *sum* of the radii of the k clusters. Besides being an interesting problem in its own right, the k-sumRadii problem is sometimes preferred over the k-center problem because it avoids the so-called *dissection effect*: an optimal k-center solution may place several pairs of close points in two different clusters ([12, 6]).

[1] corresponding author

The problem is known to be NP-hard even in metrics defined by weighted planar graphs and in metrics of constant doubling dimension [10]. Gibson et al. [10] gave an exact randomised algorithm with running time $n^{O(\log n \cdot \log \Delta)}$, where Δ denotes the aspect ratio (i.e., max. to min. interpoint distance). They also gave a randomised $(1+\varepsilon)$-approximation algorithm with running time $n^{O(\log n \cdot \log \frac{n}{\varepsilon})}$. Interestingly, there is a polynomial time algorithm [11] for a special case of points on a plane, where the problem is to cover points using disks.

The k-sumRadii problem was first considered from the approximation algorithms perspective by Doddi et al. [8]. They gave a bi-criteria logarithmic approximation algorithm with $O(k)$ clusters for k-sumRadii. Charikar and Panigrahy [5] improved this result to give the first constant factor 3.504-approximation algorithm for this problem. Using coreset based techniques of Bădoiu et al. [4], one can obtain a $(1+\varepsilon)$-approximation $2^{O(\frac{k \log k}{\varepsilon^2})} \cdot dn^{O(1)}$-time algorithm in the Euclidean setting. Note that all the works mentioned above have been on the uncapacitated version of the problem where there is no upper limit on the number of points that can be assigned to a center.

In this paper, we consider the Capacitated k-sumRadii problem, where each data point p also specifies a capacity U_p – this is an upper bound on the size of a cluster centered at p. It is worth noting that we only consider the so-called *hard* capacitated setting: multiple centers cannot be opened at the same point. Capacitated versions of clustering problems have been well-studied in the literature and arise naturally in practical settings. Although constant factor approximation algorithm is known for the Capacitated k-center problem, obtaining a similar result for the Capacitated k-median remains a challenging open problem.

Motivated by the above discussion, we consider the Capacitated k-sumRadii problem in the FPT setting. We seek algorithms which have constant approximation ratio but their running time can be of the form $f(k)\mathsf{poly}(n)$, where $f(k)$ can be an exponential function of k (and hence, such algorithms are FPT with respect to the parameter k). Inamdar and Vardarajan [14] had considered this problem in the special case of uniform capacities, i.e., each point in the input has the same capacity. They gave a 28-approximation algorithm with running time $2^{O(k^2)}\mathsf{poly}(n)$. This was subsequently improved by Bandyapadhyay et al. [1] who gave an FPT $(4+\varepsilon)$-approximation algorithm for the uniform capacity setting. The running time of their algorithm crucially uses the fact that objective function is the sum, i.e., the L_1-norm, of the radii of the clusters. Our first result improves this to an FPT $(3+\varepsilon)$-approximation algorithm. Further, our result extends to a constant factor approximation ratio for any L_p norm, where $p \geq 1$.

Bandyapadhyay et al. [1] also considered the general (i.e., non-uniform capacities) Capacitated k-sumRadii problem and gave an FPT $(15+\varepsilon)$-approximation algorithm for this problem. Their argument also extends to general L_p norm. However, it turns out that one of the arguments in their proof is incomplete and requires a deeper argument. Motivated by our insights, we refine and extend their proof which not only corrects this step but also yields a better FPT $(9+\varepsilon)$-approximation algorithm. A careful parameter balancing improves this to about 7.6-approximation. Our result also extends to any L_p norm of cluster radii, where $p \geq 1$.

Our final result shows that there exists a constant $C > 1$ such that, assuming ETH, any C-approximation algorithm for Capacitated k-sumRadii (in the non-uniform capacity setting) must have exponential running time. This also rules out polynomial time approximation algorithms with approximation ratio better than C.

1.1 Our Results and Techniques

In this section, we give a more formal description of our results. Our first result is concerned with the Capacitated k-sumRadii problem with uniform capacities. The first few steps in our algorithm are analogous to those of [1]. Suppose we can guess the radii of the clusters in an optimal solution, say $r_1^\star, \ldots, r_k^\star$. Let $C_i^\star$ denote the set of points in an optimal cluster. We call a cluster large if it has at least U/k^3 points (where U denotes the uniform capacity at each point). Again, we can guess the radii of the optimal large clusters in FPT time. Assume that there are k_L large clusters with radii $r_1^\star, \ldots, r_{k_L}^\star$. It is not hard to show that we can sample one point p_i from each large cluster with reasonably high probability. Thus, for each large cluster of radius say $r_i^\star$, we can find a ball B_i of radius $2r_i^\star$ containing it – the center of such a ball would be the sampled point p_i as above. Now suppose the union of all of the balls $B_i, i \in [k_L^\star]$, covers the entire set of points (although this is a special scenario, it captures the non-triviality of the algorithm). At this moment, we cannot assign all the points to the chosen centers p_i because of the capacity constraint. The approach of [1] is the following: consider a graph where the vertex set corresponds to the small optimal balls (i.e., $C_i^\star, i > k_L^\star$) and the balls $B_i, i \in [k_L^\star]$ constructed by our algorithm. We have an edge between two vertices here if the corresponding sets intersect. Since this is a graph on k vertices, we can guess this graph. Now for each connected component in this graph, we can have a single ball whose radius is the sum of the radii of the participating balls in this graph. It is not hard to show that such large radii balls can cover all the small balls and hence can account for the capacity deficit due to the balls $B_i, i \in [k_L^\star]$. Thus one incurs an additional 2-factor loss in the approximation ratio here, resulting in a 4-approximation algorithm. Further, the fact that the algorithm chooses a radius which is the sum of several optimal radii shows that this approach does not extend to L_p norm of radii.

Our approach is the following: the union of the balls B_i selected by our algorithm covers all the points. We can show that a small extension of these balls, say B_i' for each B_i, has the property that they can cover all the points (because even the original balls B_i has this property) and each such ball is assigned only slightly more than U points. Finally, we use a subtle matching based argument to show that the slightly extra points in each of these extended balls B_i' can be covered by a ball whose radius matches the radius of a unique small optimal ball. This allows us to get a better 3-approximation algorithm, and our approach extends to L_p norm of cluster radii as well. Thus, we get the following result:

▶ **Theorem 1.** *There is a randomized $(3+\varepsilon)$-approximation algorithm for* Capacitated k-sumRadii *with uniform capacities, where $\varepsilon > 0$ is any positive constant. For $p \geq 1$, there is a randomized $(1+\varepsilon)(2^{2p-1}+1)^{1/p}$-approximation algorithm when the objective function is the L_p norm of the cluster radii. The expected running time of both the algorithms is $2^{O(k\log(k/\varepsilon))} \cdot \mathsf{poly}(n)$.*

We now consider the general Capacitated k-sumRadii problem with non-uniform capacities. Again, our initial steps are similar to those in [1] – we initially guess the optimal cluster radii $r_1^\star, \ldots, r_k^\star$; and greedily find a set of balls $B_1, \ldots, B_\ell$, where $\ell \leq k$ and these balls cover the entire set of points. Further, each of the balls B_i is supposed to also contain the corresponding optimal cluster $C_i^\star$. At this moment, we need to find some more balls which can be used to satisfy the capacity constraints, i.e., balls which are approximations to $C_{\ell+1}^\star, \ldots, C_k^\star$ respectively. Bandyopadhyay et al. [1] approached this problem as follows: for an index $i \in [k] \setminus [\ell]$, one can guess the the set of balls $B_j, j \in [\ell]$ that intersect $C_i^\star$. If any of these balls B_j have smaller radius than $r_i^\star$, we can use a enlarged version B_j' of B_j to cover $C_i^\star$ and treat this as the ball B_i. The radius of B_j' can be charged to $r_i^\star$. Therefore, the non-trivial case arises when $r_i^\star$ is much less than the radii of each of the balls B_j intersecting $C_i^\star$. In such a case, they realized that $C_i^\star$ is contained in the intersection of the balls B_j' defined above. Therefore, they maintain a set of *replacement* balls, which are mutually disjoint, and are supposed to contain at least the same number of points as the corresponding cluster $C_i^\star$.

Ensuring that one can maintain such a collection of disjoint replacement balls that are also disjoint with some desired subset of optimal balls turns out to be non-trivial. In fact, we feel that the proof of Lemma 14 in [1], without a non-trivial fix, has a technical flaw: while inserting a new replacement ball in the set $\mathcal{B}_2$, it may happen that the replacement intersects with an optimal ball with index in $\mathcal{I}_2$. [2] Our algorithm uses a more fine grained approach: we first identify a set P_j of points which could contain the desired replacement ball (this step is also there in [1]). Now we guess the set Z of optimal clusters which could intersect this candidate set P_j. The algorithm then runs over several iterations, where in each iteration it shrinks the candidate set of points P_j containing the replacement ball, and hence the list Z (see Algorithm 5). This process terminates when it either identifies a replacement ball or a ball containing a cluster $C^\star_r$. To get a better approximation ratio, we also maintain several set of balls with varying properties. We prove the following result:

▶ **Theorem 2.** *There is a randomized $(4+\sqrt{13}+\varepsilon)$-approximation algorithm for* Capacitated k-sumRadii*, where $\varepsilon > 0$ is any positive constant. For $p \geq 1$, there is a randomized $(4+\sqrt{13}+\varepsilon)$-approximation algorithm when the objective function is the L_p norm of the cluster radii. The expected running time of both the algorithms is $2^{O(k^3+k\log(k/\varepsilon))} \cdot \mathsf{poly}(n)$.*

Our final result shows that the exponential dependence on running time is necessary if we want a c-approximation algorithm, where c is a constant larger than 1. This result assumes that the Exponential Time Hypothesis holds and uses a reduction from Vertex Cover on bounded degree graphs. A nearly exponential time lower bound for approximating Vertex Cover on such graphs follows from [7]. More formally, we have the following result:

▶ **Theorem 3.** *Assume that ETH holds. Then there exist positive constants c_1, c_2, where $c_1 > 1$, such that any algorithm for* Capacitated k-sumRadii *with running time at most $2^{c_2 k/\mathsf{polylog} k} \cdot \mathsf{poly}(n)$ must have approximation ratio at least c_1.*

Due to space limitations, we defer the formal proof to the full version of the paper.

1.2 Preliminaries

We define the Capacitated k-sumRadii problem. An instance $\mathcal{I}$ of this problem is specified by a set of n points P in a metric space, a parameter k and capacities U_p for each point $p \in P$. A solution to such an instance needs to find a subset $C \subseteq P$, denoted "centers", such that $|C| = k$. Further, the solution assigns each point in P to a unique center in C such that for any $c \in C$, the total number of points assigned to it is at most its capacity U_c. It is worth noting that we are interested in the so-called "hard" capacity version, where one cannot locate multiple centers at the same point.

Assuming $C = \{c_1, \ldots, c_k\}$, let C_j denote the set of points assigned to the center c_j by this solution. We refer to C_j as the *cluster* corresponding to C_j and define its radius r_j as $\max_{p \in P_j} d(p, c_j)$. The goal is to find a solution for which the sum of cluster radii is minimized, i.e., we wish to minimize $\sum_{j=1}^{k} r_j$. In the L_p-norm version of this problem, the specification

[2] In the proof of Lemma 14 in [1] (Lemma 2.5 in the arXiv version), the last line of the first paragraph reads "If B_x does not intersect any ball $B^\star_{i'}$ with $i' \in [k] \setminus (I_1 \cup I_2)$, then by the above discussion we have that setting $B_i = B_x$, $(I_1, I_2 \cup \{i\}, \mathcal{B}_1, \mathcal{B}_2 \cup \{B_i\})$ is a valid configuration." However, by doing so, one cannot rule out the intersection of B_i with a ball $B^\star_{i'}$ with $i' \in I_2$, contradicting the fourth invariant property of a valid configuration: "For every $i \in I_2$ and $s \notin I_1$, $B^\star_s$ and B_i do not intersect." It is tricky to note why this is the case. They indeed rule out the intersection of $B^\star_i$ with any ball in $\mathcal{B}_2$, but they do not ensure that B_i does not intersect an optimal ball $B^\star_s$ corresponding to the index $s \in I_2$.

of an instance remains as above, but the objective function is given by $\left(\sum_{j=1}^{k} r_j^p\right)^{1/p}$. Given an instance $\mathcal{I}$ as above, let $\mathsf{OPT}(\mathcal{I})$ denote an optimal solution to $\mathcal{I}$. Let $r_1^\star, \ldots, r_k^\star$ be the radii of the k clusters in this optimal solution. By a standard argument, we can show that one can guess close approximations to these radii.

▶ **Lemma 4** ([1]). *Given a positive constant $\varepsilon > 0$, there is an $O(2^{O(k\log(k/\varepsilon))} \cdot n^3)$ time algorithm that outputs a list $\mathcal{L}$, where each element in the list is a sequence $(r_1, \ldots, r_k)$ of non-negative reals, such that the following property is satisfied: there is a sequence $(r_1, \ldots, r_k) \in \mathcal{L}$ such that for all $j \in [k]$, $r_j \geq r_j^\star$ and $\sum_{j=1}^{k} r_j \leq (1+\varepsilon)\sum_{j=1}^{k} r_j^\star$. Further, the size of the list $\mathcal{L}$ is $O(2^{O(k\log(k/\varepsilon))} \cdot n^2)$.*

Given a point x and non-negative value r, let $B(x, r)$ denote the *ball* of radius r around x, i.e., $B(x, r) := \{p \in P : d(p, x) \leq r\}$, where $d(a, b)$ denote the distance between points a and b. For a positive integer r, we shall use $[r]$ to denote the set $\{1, \ldots, r\}$.

1.3 Related Work

In this section, we mention a few related research works on the sum-of-radii problem that were not discussed earlier in the introduction. Note that most of the work has been for the uncapacitated case. Behsaz and Salavatipour [3] give an exact algorithm for the sum of radii problem in metrics induced by unweighted graphs for the case when singleton clusters are disallowed. Friggstad and Jamshidian [9] give a 3.389-approximation algorithm for the sum of radii problem in general metrics. This is an improvement over the 3.504 approximation of Charikar and Panigrahy [5], the first constant factor approximation algorithm for this problem. Bandyapadhyay and Varadarajan [2] discuss the variant of the problem where the objective function is the sum of the α^{th} power of the radii. The algorithm can output $(1+\varepsilon)k$ centers in the variant they study. Henzinger et al. [13] give a constant approximation algorithm for metric spaces with bounded doubling dimension in the *dynamic setting* where points can appear and disappear. There also have been works for special metrics such as two-dimensional geometric settings (e.g., [10]). The related problem of sum-of-diameters has also been studied (e.g., [8, 3]).

We now give an outline of rest of the paper. In Section 2, we prove Theorem 1, i.e., we present an FPT approximation algorithm for CAPACITATED k-SUMRADII in the uniform capacity case. The more general non-uniform capacity case is considered in Section 3, where we prove Theorem 2.

2 Uniform Capacities

In this section, we consider the special case of CAPACITATED k-SUMRADII when all the capacities U_p are the same, say U. We give some notation first. Fix an optimal solution $\mathcal{O}$ to an instance $\mathcal{I}$ given by a set of n points P. Let $c_1^\star, \ldots, c_k^\star$ be the k centers chosen by the optimal solution. For an index $j \in [k]$, let $C_j^\star$ denote the set of points assigned to $c_j^\star$ by this solution (i.e., the cluster corresponding to the center $c_j^\star$). Let $r_1^\star, \ldots, r_k^\star$ be the radii of the corresponding clusters $C_1^\star, \ldots, C_k^\star$. We also fix a parameter $\gamma := \frac{1}{k^2}$.

▶ **Definition 5.** *Call the optimal cluster $C_j^\star$* large *if $|C_j^\star| \geq \frac{\gamma U}{k}$; otherwise call it* small. *Let $k_L^\star$ denote the number of large clusters in the optimal solution $\mathcal{O}$.*

Algorithm 1 An iteration of the algorithm for CAPACITATED k-SUMRADII when all capacities are U.

```
1.1  Input: Set $P$ of $n$ points, parameter $k, k_L^\star$, radii $r_1, \ldots, r_k$, capacity $U$.
1.2  Initialize a set $\mathcal{B}$ to empty.
1.3  Initialize an index set $I$ to $\{1, \ldots, k_L^\star\}$.
1.4  for $j = 1, \ldots, k_L^\star$ do
1.5      Choose a point $c_j \in P$ uniformly at random.
1.6      Add $B_j := B(c_j, 2r_j)$ to $\mathcal{B}$ .
1.7  while there is a point $x$ not covered by the balls in $\mathcal{B}$ do
1.8      (If $I = [k]$, output fail).
1.9      Choose an index $j \in [k] \setminus I$ uniformly at random.
1.10     Add $B_j := B(x, 2r_j)$ to $\mathcal{B}$ and add $j$ to $I$ and define $c_j := x$.
1.11 Initialize sets $T_j \subset [k], j \in I$, to emptyset.
1.12 for each $i \in [k] \setminus I$ do
1.13     Choose an index $j \in I$ uniformly at random and add $i$ to $T_j$.
1.14 Define $I' := \{j \in I : |T_j| > 0\}$.
1.15 for each $j \in I$ do
1.16     if $j \in I'$ then
1.17         Define $B'_j := B(x, 2r_j + 2R_j)$ and $U'_j := U(1 + \gamma)$, where $R_j := \max_{i \in T_j} r_i$.
1.18     else
1.19         Define $B'_j := B_j, U'_j = U$.
1.20     Find disjoint subsets $G_j \subseteq B'_j$ for each $j \in I$ such that $|G_j| \le U'_j$ and $P = \cup_j G_j$.
1.21     (terminate with failure if such subsets $G_j$ do not exist)
1.22     Let $I'' \subseteq I'$ be the index set consisting of indices $j$ such that $|G_j| > U$.
1.23     Call Redistribute($\{G_j : j \in I''\}, \{c_j : j \in I\}, \{r_j : j \in [k_L]\}$).
1.24     (terminate with failure if Redistribute outputs fail)
1.25     Let $\{(w_j, A_j) : j \in I''\}$ be the clustering returned by Redistribute.
1.26     Output $\{(c_j, G_j) : j \in I \setminus I''\} \cup \{(c_j, G_j \setminus A_j) : j \in I''\} \cup \{(w_j, A_j) : j \in I''\}$.
```

Assume without loss of generality that the clusters $C_1^\star, \ldots, C_{k_L^\star}^\star$ are large (and the rest are small). Using Lemma 4, we can assume that we know radii $r_1, \ldots, r_k$ satisfying the condition that $r_j \ge r_j^\star$ for all $j \in [k]$ and $\sum_j r_j \le (1+\varepsilon) \sum_j r_j^\star$. By cycling over the k possible choices of $k_L^\star$, we can also assume that we know this value. The algorithm is given in Algorithm 1. It begins by guessing the center c_j of each large optimal cluster $C_j^\star$ and defines B_j as the ball of radius $2r_j$ around c_j (line 1.4). The intuition is that c_j may not be equal to the center $c_j^\star$ but will lie inside the cluster $C_j^\star$ with reasonable probability, and in this case, the ball B_j will contain $C_j^\star$. Let us assume that this event happens. Now the algorithm adds some more balls to the set $\mathcal{B}$ (that maintains the set of balls constructed so far). Whenever there is a point x that is not covered by the balls in $\mathcal{B}$, we guess the index j of the optimal cluster $C_j^\star$ containing x (line 1.9). We add a ball of radius $2r_j$ around x to $\mathcal{B}$: again the intuition is that if $x \in C_j^\star$, then this ball contains $C_j^\star$.

The index set I maintains the set of indices j for which we have approximation to $C_j^\star$ in $\mathcal{B}$. At this moment, the set of balls in $\mathcal{B}$ cover the point set P but we are not done yet because we need to assign points to balls while maintaining the capacity constraints. Now, for each index $i \notin I$, we guess the index of a ball $B_j \in \mathcal{B}$ such that B_j intersects $C_i^\star$ (such a ball must exist since the balls in $\mathcal{B}$ cover P, and in case there are more than one candidates for B_j, we pick one arbitrarily). We add the index i to a set T_j (line 1.13). Now we define I' as the subset of I consisting of those indices j for which T_j is non-empty (line 1.14). Now, for each $j \in I'$, let R_j denote the maximum radius of any of the balls corresponding to T_j. We replace B_j by a larger ball B'_j by extending the radius of B_j by $2R_j$ – this ensures that B'_j contains $C^\star_i$ for any $i^\star \in T_j$. Further we allow B'_j to have $U'_j := U(1+\gamma)$ points assigned to it (line 1.17). For indices $j \in I \setminus I'$, we retain B'_j, U'_j as B_j, U_j respectively (line 1.19). Now, we find a subset G_j of each ball B'_j such that $|G_j| \le U'_j$ and $\cup_j G_j$ covers all the points P (line 1.20). We shall show such a subset exists if all our guesses above our correct. Otherwise we may not be able to find such sets G_j; and in this case the iteration terminates with failure. The intuition is that one feasible choice of G_j is as follows: for each $j \in I \setminus I'$, $G_j := C^\star_j$, whereas for an index $j \in I'$, we set G_j to be the union of $C^\star_j$ and all the clusters $C^\star_i$, where $i \in T_j$. Since each small cluster has at most $\gamma U/k$ points, the total number of points in the latter clusters is at most γU. But we had set U'_j to $(1+\gamma)U$. It is also easy to check that we can find such sets G_j by a simple b-matching formulation. Now we consider those subsets G_j for which $|G_j| > U$ (this can only happen if $j \in I'$) and let I'' denote the corresponding index set (line 1.22). Finally, we call the procedure **Redistribute** in line 1.23 to redistribute the points in $G_j, j \in I''$ such that each such set has at most U points. Note that so far we have only constructed $|I|$ balls in $\mathcal{B}$ and we can still add $k - |I|$ extra balls. The procedure **Redistribute** returns these extra balls – we finally return these balls and remove suitable points from $G_j, j \in I''$ (line 1.20). We shall show later that $|I''|$ is less than k_L. Again, note that if our random choices were bad, it is possible that **Redistribute** returns failure, in which case the iteration ends with failure.

Algorithm 2 Algorithm **Redistribute**.

```
2.1  Input: $(G_1, \dots, G_h)$, where $G_i$'s are pair-wise disjoint subsets of $P$, where $|G_i| > U$
     for all $i$; a subset $C$ of $P$, and a set $R = \{r_1, \dots, r_{k_L}\}$ of radii.
2.2  for each ordered subset $(r_{\sigma_1}, \dots, r_{\sigma_h})$ of $R$ do
2.3      Construct a bipartite graph $H = (V_L, V_R, E)$ as follows.
2.4      $V_L$ has one vertex $v_i$ for each $G_i, i \in [h]$. $V_R$ is defined as $P \setminus C$.
2.5      Add an edge $(v_i, w), v_i \in V_L, w \in V_R$, iff $|B(w, r_{\sigma_i}) \cap G_i| \ge \gamma U$.
2.6      if $H$ has a matching that matches all vertices in $V_L$ then
2.7          Suppose $v_i$ is matched with $w_i \in R$ for each $i \in [h]$.
2.8          Let $A_i$ be a subset of $B(w_i, r_{\sigma_i}) \cap G_i$ such that $|A_i| = \gamma U$.
2.9          Return $\{(w_i, A_i) : i \in [h]\}$ (and end the procedure).
2.10 Return fail.
```

We now describe the algorithm **Redistribute** in Algorithm 2. The procedure receives three parameters: the first parameter is a class of subsets $\{G_1, \dots, G_h\}$ which are mutually disjoint. These correspond to the sets $G_i, i \in I''$ constructed in Algorithm 1 (and hence $h = |I''|$). The second parameter is a set C, which corresponds to the set of centers chosen by Algorithm 1 (till the call to this procedure) and the third parameter is a set of radii $\{r_1, \dots, r_{k_L}\}$ which correspond to the radii of the large clusters. We shall ensure that $k_L \ge h$. Recall that the goal is to identify balls which can take away γU points from each of the sets G_i. The radii of these balls shall come from the set $R := \{r_1, \dots, r_{k_L}\}$, and we shall show that we can associate a unique radius with each of the desired balls. Thus, the procedure tries out all ordered subsets of size h of R (line 2.2). For each such ordered subset $(r_{\sigma_1}, \dots, r_{\sigma_h})$, we try to remove γU points from each subset G_i by using a suitable ball of radius r_{σ_i}. Thus, we construct a bipartite matching instance as follows: the left side has one vertex for each subset G_i and the right side has one vertex for each potential center of a ball (line 2.4). Now, we add an edge between a vertex v_i corresponding to G_i on left and a vertex w on the right side if $G_i \cap B(w, r_{\sigma_i})$ has size at least γU (line 2.5). If this graph has a perfect matching, then we identify the desired subsets $A_i \subseteq G_i$ of size γU (line 2.8) and return these.

2.1 Analysis

In this section, we prove correctness of the algorithm. We shall show that with non-zero probability the algorithm outputs a 3-approximate solution. We first define the set of desirable events during the random choices made by Algorithm 1:

- $\mathcal{E}_1$: For each $j \in [k_L^\star]$, the point c_j chosen in line 1.6 of Algorithm 1 belongs to the cluster $C_j^\star$.
- $\mathcal{E}_2$: For each point x chosen in line 1.7 of Algorithm 1, the index j chosen in line 1.9 satisfies the property that $x \in B_j^\star$. Further, the algorithm does not output **fail** in line 1.8.
- $\mathcal{E}_3$: For each index i considered in line 1.12 in Algorithm 1, the index $j \in I$ selected in line 1.13 satisfies the property that $B_j \cap B_i^\star$ is non-empty.
- $\mathcal{E}_4$: The set of centers $\{c_j : j \in I\}$ selected by Algorithm 1 is disjoint from the set of optimal centers of the large balls, i.e., $\{c_j^\star : j \in [k_L^\star]\}$.

We first show that all of these desirable events happen with non-zero probability.

▶ **Lemma 6.** *Assuming $n \geq 2k^5$, all the events $\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_4$ together happen with probability at least $\frac{1}{k^{O(k)}}$.*

Proof. We first check event $\mathcal{E}_4 \cap \mathcal{E}_1$. Consider an iteration j of the **for** loop in line 1.4. Conditioned on the choices in the previous $j-1$ iterations, the probability that $c_j \in C_j^\star \setminus \{c_j^\star : j \in [k_L^\star]\}$ is at least $\frac{|C_j^\star|-k}{n}$. Since $C_j^\star$ is large, we know that $|C_j^\star| \geq \gamma U/k \geq n/k^4$ (since $\gamma = 1/k^2$ and $U \geq n/k$). Using the fact that $n \geq 2k^5$, we get

$$\Pr[\mathcal{E}_1 \cap \mathcal{E}_4] = \Pr[c_j \in C_j^\star \setminus \{c_j^\star : j \in [k_L^\star]\}, \text{for all } j = 1, \ldots, k_L^\star] \geq \left(\frac{1}{2k^4}\right)^k = \frac{1}{2^k k^{4k}}$$

Now we consider $\mathcal{E}_2$. We condition on the coin tosses before the **while** loop in line 1.7 such that $\mathcal{E}_1 \cap \mathcal{E}_4$ occur. This implies that for every $j \in [k_L^\star]$, $C_j^\star \subseteq B_j$.

The probability that we correctly guess the index j such that the cluster $C_j^\star$ contains x (in line 1.7) is $1/k$. Since there can be at most k iterations of the **while** loop, the probability that this guess is correct for each point x chosen in line 1.7 is at least $1/k^k$. Further, if this guess is always correct, then B_j contains $C_j^\star$ for all $j \in I$. This shows that if I becomes equal to $[k]$, then the balls B_j would cover P and hence, we won't output **fail**. Thus, we see that

$$\Pr[\mathcal{E}_2 | \mathcal{E}_1 \cap \mathcal{E}_4] \geq 1/k^k.$$

Finally we consider $\mathcal{E}_3$. Again condition on the events before the **for** loop in line 1.12 and assume that $\mathcal{E}_1 \cap \mathcal{E}_2 \cap \mathcal{E}_4$ occur. There are at most k iterations of the **for** loop. Since $\cup_{j \in I} B_j = P$, there must exist an index $j \in I$ such that B_j intersects $C_i^\star$. Therefore, the probability that we guess such an index j in line 1.13 is at least $1/k$. Thus, we get

$$\Pr[\mathcal{E}_3 | \mathcal{E}_1 \cap \mathcal{E}_2 \cap \mathcal{E}_4] \geq 1/k^k.$$

Combining the above inequalities, we see that $\Pr[\mathcal{E}_1 \cap \mathcal{E}_2 \cap \mathcal{E}_3 \cap \mathcal{E}_4] \geq \frac{1}{2^k k^{6k}}$. This implies the desired result. ◀

We now show that the sets G_j as required in line 1.20 exist and can be found efficiently.

▷ Claim 7. Assume that the events $\mathcal{E}_1, \ldots, \mathcal{E}_4$ occur. Let B'_j, U'_j be as defined in lines 1.15–1.19 of Algorithm 1. Then there exist mutually disjoint subsets $G_j \subseteq B'_j$ for each $j \in I$ such that $P = \cup_{j \in I} G_j$ and $|G_j| \le U'_j$ for each $j \in I$. Further, the subsets G_j can be found in $\mathsf{poly}(n)$ time.

Proof. Since events $\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3$ occur, it is easy to check that for each $j \in I$, B'_j contains $C^\star_j \cup \bigcup_{h \in T_j} C^\star_h$. Further, if T_j is non-empty, then

$$|C^\star_j| + \sum_{h \in T_j} |C^\star_h| \le U(1+\gamma),$$

because each of the clusters $C^\star_h$ is small. Thus, one feasible choice for the subsets G_j is as follows: for each $j \in I$ such that T_j is empty, define $G_j = C^\star_j$, otherwise define $G_j = C^\star_j \cup \bigcup_{h \in T_j} C^\star_h$. This proves the existence of the desired subsets G_j. It is easy to check that such a collection of subsets can be found by standard flow-based techniques, and hence, would take $\mathsf{poly}(n)$ time. ◁

We now show that the **Redistribute** outputs the desired subsets.

▶ **Lemma 8.** *Assume that the events $\mathcal{E}_1, \ldots, \mathcal{E}_4$ occur. Then Algorithm 2 does not fail.*

Proof. We first consider the following bipartite graph $H' = (V'_L, V'_R, E')$ (this is for the purpose of analysis only): the vertex set V'_L has one vertex v_i for each of subsets G_i, and hence, is same as the set V_L considered in line 2.4 of Algorithm 2. V'_R has a vertex w_j for each optimal cluster $C^\star_j, j \in k^\star_L$. We add an edge (v_i, w_j) iff $|G_i \cap C^\star_j| \ge \gamma U$. We claim that there is a matching in this graph that matches all the vertices in V'_L. Suppose not. Then there is a subset X of V'_L such that $|N(X)| < |X|$, where $N(X)$ denotes the neighborhood of X. Now,

$$\begin{aligned}
\sum_{i \in X} |G_i| &= \sum_{i \in X} \sum_{j \in [k]} |C^\star_j \cap G_i| = \sum_{j \in [k]} \sum_{i \in X} |C^\star_j \cap G_i| \\
&\le \sum_{j \in N(X)} |C^\star_j| + \sum_{j \in [k] \setminus N(X)} \sum_{i \in X} \gamma U \\
&\le |N(X)| U + k^2 \gamma U \\
&\le U(|X|-1) + U = U|X|,
\end{aligned}$$

where the second inequality uses the fact that if $i \in [k] \setminus N(X)$, then $|C^\star_j \cap G_i| \le \gamma U$. Indeed, if $C^\star_j$ is small, this follows from the observation that $|C^\star_j| \le \gamma U$. Otherwise, $j \in [k^\star_L]$ and hence we have a vertex w_j corresponding to $C^\star_j$ in H'. Since (v_i, w_j) is not an edge in H', it follows that $C^\star_j \cap G_i$ has size at most γU. Now we get a contradiction because for each i, $|G_i| > U$ and hence, $\sum_{i \in X} |G_i| > U|X|$

Thus we have shown that H' has a matching that matches all the vertices in V'_L – let w_{σ_i} be the vertex in V'_R which is matched to $v_i \in V'_L$ by this matching. Now consider the iteration of the **for loop** in line 2.2 in Algorithm 2 where the sequence of radii is given by $(r_{\sigma_1}, \ldots, r_{\sigma_h})$. We claim that the graph H constructed in this iteration (in line 2.4) has a matching that matches all the vertices in V_L. Indeed, we can match the vertex $v_i \in L$ with $c^\star_{\sigma_i}$ because $\gamma U \le |G_i \cap C^\star_{\sigma_i}| \le |G_i \cap B(c^\star_{\sigma_i}, r_{\sigma_i})|$ – we use the fact that the event $\mathcal{E}_4$ occurred, and hence, $c^\star_{\sigma_i} \in C$.

Now, let M be the matching found in line 2.6 and suppose v_i is matched with a vertex $w_i \in R$. By definition of H, $B(w_i, r_{\sigma_i}) \cap G_i$ has size at least γU. Thus, we can find the desired subset A_i in line 2.8. Note that the subsets A_i are mutually disjoint since the sets G_i are mutually disjoint. Further the points $w_i, i \in H$, are also distinct since M is a matching. Thus, the procedure **Redistribute** returns h clusters, each containing γU points. ◀

It follows from the results above that the set of k clusters returned by Algorithm 1 in line 1.26 cover all the points in P and satisfy the capacity constraints. We now consider the objective function value of this solution.

▶ **Lemma 9.** *Assume that the events $\mathcal{E}_1, \ldots, \mathcal{E}_4$ occur. Then the total sum of the radii of the clusters returned by Algorithm 1 is at most $3\sum_{j\in[k]} r_j$. Further, for any $p \geq 1$, the total L_p norm of the radii of the clusters returned by this algorithm is at most $(2^{2p-1} + 1)^{1/p} \left(\sum_{j\in[k]} r_j^p\right)^{1/p}$.*

Proof. For each $j \in I$, the ball B_j constructed during lines 1.4–1.10 has radius at most $2r_j$. Now, the ball B'_j constructed during the **for** loop in line 1.15 has radius at $2r_j + \sum_{h\in T_j} 2r_h$. Therefore, the total radii of the balls in B'_j is at most

$$\sum_{j\in I}\left(2r_j + \sum_{h\in T_j} 2r_h\right) \leq 2\sum_{j\in[k]} r_j,$$

where the inequality follows from the fact that sets $\{j\} \cup T_j$, where $j \in I$, are mutually disjoint. Finally, the total sum of the radii of the sets A_i returned by **Redistribute** is at most $\sum_{j\in[k]} r_j$. This proves the first statement in the Lemma.

Further, the L_p-norm of the radii of the clusters output by this algorithm is at most

$$\sum_{j\in I}(2r_j + 2R_j)^p + \sum_{j\in[k]} r_j^p \leq \sum_{j\in[k]} 2^{2p-1} r_j^p + \sum_{j\in[k]} r_j^p,$$

where R_j is as defined in line 1.17 of Algorithm 1. This completes the proof of the desired result. ◀

Thus, with probability at least $1/k^{O(k)}$ (Lemma 6), Algorithm 1 outputs a feasible solution with approximation guarantees as given by Lemma 9. Repeating Algorithm 1 $k^{O(k)}$ times and using Lemma 4 yields Theorem 1.

3 Non uniform capacities

In this section, we consider the general case of Capacitated k-sumRadii when points can have varying capacities. Recall that for a point p, U_p denotes the capacity of p. We first give an informal description of the algorithm. Let $r_1^\star, \ldots, r_k^\star$ denote the radii of the k optimal clusters $C_1^\star, \ldots, C_k^\star$. Let $c_1^\star, \ldots, c_k^\star$ be the centers of these clusters respectively. Let $B_j^\star$ denote the ball $B(c_j^\star, r_j^\star)$ – note that $C_j^\star \subseteq B_j^\star$. As in the case of uniform capacities (Section 2), we begin by assuming that we know radii $r_1, \ldots, r_k$ satisfying the conditions of Lemma 4, i.e., $r_j \geq r_j^\star$ and $\sum_{j=1}^k r_j \leq (1+\varepsilon)\sum_{j=1}^k r_j^\star$, where $\varepsilon > 0$ is an arbitrarily small constant.

Our algorithm maintains three subsets of balls, namely $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$. Each ball in these sets corresponds (in a sense that we shall make clear later) to a unique optimal ball $B_j^\star$. Thus we denote balls in $\mathcal{B}_1 \cup \mathcal{B}_2 \cup \mathcal{B}_3$ as B_j, where the index j denotes the fact that B_j

corresponds to $B_j^\star$. Further, we use c_j to denote the center of B_j. We maintain three sets of (mutually disjoint) indices $I_1, I_2, I_3 \subseteq [k]$ respectively. We also maintain an index set $I_4 := [k] \setminus (I_1 \cup I_2 \cup I_3)$ denoting those indices for which we have not assigned a ball yet.

The set $\mathcal{B}_1$ is obtained by the following greedy procedure: while there are points not covered by the balls in $\mathcal{B}_1$, we pick an uncovered point p, guess the index of the optimal cluster covering it, say j, and add the ball $B(c_j, 3r_j)$ to $\mathcal{B}_1$, where c_j is a high capacity point close to p (the actual process is slightly nuanced, but the details will be clarified in the actual algorithm description). When this process ends, we have a set of balls covering P such that each ball in $\mathcal{B}_1$ covers a corresponding (unique) optimal ball. The set $\mathcal{B}_1$ remains unchanged during rest of the algorithm. We now need to add more balls to this solution in order to satisfy the capacity constraints.

Such balls shall be added to the sets $\mathcal{B}_2$ and $\mathcal{B}_3$ respectively (and the index sets I_2 and I_3 shall maintain the correspondence with unique clusters in the optimal solution respectively). Roughly, a ball $B_j \in \mathcal{B}_2$ of the form $B(c_j, r)$ shall satisfy the condition that $r \leq 5r_j$ and $B_j^\star \subseteq B_j$. Our algorithm shall never remove a ball that once gets added to $\mathcal{B}_2$. A typical setting when we can add a ball to $\mathcal{B}_2$ is the following: Suppose there is an index $j \in I_4$ such that the optimal ball $B_j^\star$ intersects a ball $B_h \in \mathcal{B}_1 \cup \mathcal{B}_2$ and r_j is at least the radius of B_h (we do not know this fact a priori, but our algorithm can *guess* such cases). In this case, we find a high capacity point x in the vicinity of c_h, such that $B_j = B(x, 5r_j)$ covers $B_j^\star$.

A ball B_j added to the set $\mathcal{B}_3$ shall have the property that it does not intersect any of the balls $B_h^\star, h \in I_3 \cup I_4$ (again, we cannot be certain here since we do not know the optimal balls, but we shall show that in one of the cases guessed by our algorithm, this invariant will be satisfied). A ball B_j once added to $\mathcal{B}_3$ can get deleted, but in this case we will add a corresponding ball to $\mathcal{B}_2$, i.e., we shall remove the index j from I_3 and add it to I_2. This can happen because of the following reason: suppose we have identified a ball $B_h, h \in I_4$ (centered around a point c_h), that we want to add to $\mathcal{B}_3$. But it intersects the optimal ball $B_j^\star$ for an index $j \in I_3$ (again, we *guess* this fact). Now if $r_j \geq r_h$, it follows that, as before, we can find a high capacity point x in the vicinity of c_h such that $B_j = B(x, 5r_j)$ covers $B_j^\star$.

Identifying a ball which gets added to $\mathcal{B}_3$ is at the heart of our technical contribution. Essentially we start with a suitable ball B (picked from a certain set of possibilities) which we would like to add to $\mathcal{B}_3$. As long as there is an optimal ball $B_j^\star, j \in I_3 \cup I_4$ intersecting B, we able transfer an index from I_3 to I_2 or suitably shrink the possibilities for identifying the ball B.

We mention one final technicality. For each ball $B_j \in \mathcal{B}_3$ maintained by the algorithm, we shall maintain a subset C_j. The subset C_j is meant to capture the subset of points that will be actually be part of the j^{th} cluster output by the algorithm (although the final output may be a subset of C_j). The reason for this is as follows. We would like to maintain the following invariant during the algorithm:

Invariant 1: Let $j, j' \in I_3$ be two distinct indices. Then $C_j \cap C_{j'} = \emptyset$. Further, for any $j \in I_3$ and $i \in I_3 \cup I_4$, $B_i^\star \cap C_j = \emptyset$.

3.1 Algorithm Description

We now give a formal description of our algorithm for Capacitated k-sumRadii in Algorithm 4. Here we describe one iteration of the algorithm and shall show that it outputs the desired solution with probability at least $\frac{1}{2^{O(k^3)}}$. As mentioned above, there are three sets of balls $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ and corresponding (mutually disjoint) index sets I_1, I_2, I_3. The set I_4 which is initialized to $[k]$ stores the index sets for which we haven't added a ball in $\mathcal{B}_1 \cup \mathcal{B}_2 \cup \mathcal{B}_3$.

We first explain a subroutine that we shall be using repeatedly during the algorithm: **InsertBall**(p, j, r, u)). Here u is an index in $\{1, 2, 3\}$. The procedure is supposed to insert the ball $B(p, r)$ centered at p to the set $\mathcal{B}_u$ (and add the index j to I_u – the index j before this step lies in I_4, the set of *unused* indices). However, there is a caveat – we want to avoid the case when p happens to be same as the optimal center $c^\star_h$ for some other $h \neq j$. The reason is that in a subsequent step, the algorithm may try to insert a ball that is an approximation of $B^\star_h$. At this moment, it may happen that $c^\star_h$ is the only feasible choice for a center because all other points close to $c^\star_h$ have very low capacity. Therefore, the procedure **InsertBall** needs to check if this is the case. In particular, it *guesses* this fact, and if indeed p happens to be same as $c^\star_h$, it places a ball of radius r_h around $c^\star_h$ in $\mathcal{B}_1$. The algorithm maintains a global index set $I^\star$ consisting of those indices j for which it has executed this step, i.e., for which the center of the corresponding ball $B_j \in \mathcal{B}_1$ is same as $c^\star_j$. Note that $I^\star$ is a subset of I_1. Whenever it adds such an index to $I^\star$, it sets I_1 to $I^\star$, $\mathcal{B}_1$ to $\{B(c_i, r_i), i \in I^\star\}$ and resets $I_2, I_3, \mathcal{B}_2, \mathcal{B}_3$ to empty (see line 3.5 in Algorithm 3), and then restarts the Algorithm 4 from line 4.4

For sake of clarity of the algorithm, we shall always assume that the choice among the two options (i) and (ii) taken by this algorithm is correct, and also that whenever the algorithm chooses option (ii), it chooses h, satisfying $p = c^\star_h$. The reason is that we will later see that this procedure is called $O(k^2)$ times, and it chooses the option (ii) at most k times. Hence, the probability that procedure makes the correct choices each time is $1/2^{O(k^2)}$, which suffices for our purpose.

Algorithm 3 The procedure **InsertBall**(p, j, r, u)).

```
3.1 Input: Candidate center $p$, radius $r$, index $j \in I_4$, index $u \in \{1, 2, 3\}$.
3.2 Perform one of the following two steps with equal probability:
3.3 (i) Add the ball $B(p, r)$ to $\mathcal{B}_u$ and the index $j$ to $I_u$ (and remove $j$ from $I_4$). Set
      $c_j = p$.
3.4 (ii) Guess an index $h \in [k] \setminus I^*$ uniformly at random. Set $c_h = p$.
3.5       set $I_2, I_3, \mathcal{B}_2, \mathcal{B}_3$ to empty.
3.6       Add $h$ to $I^\star$
3.7       Set $I_1 = I^\star$ and $\mathcal{B}_1 = \{B(c_i, r_i), i \in I^\star\}$
3.8       Go to line 4.4 of  Algorithm 3
```

We now describe Algorithm 3.

In lines 4.4–4.9, we add balls to $\mathcal{B}_1$ whose union covers P. Each such ball B_j, corresponding to an index j (which is maintained in the index set I_1), is supposed to contain the optimal ball $B^\star_j$. Further, the center c_j of B_j should have capacity at least that of $c^\star_j$. In each iteration of this **while** loop, we first pick an uncovered point p and guess the index j such that $p \in C^\star_j$ (line 4.7). We now find the highest capacity point $x \in B(p, r_j)$ (while avoiding the centers already chosen). We now call **InsertBall**$(x, j, 3r_j, 1)$, i.e., we would like to insert the ball $B(x, 3r_j)$ to $\mathcal{B}_1$.

In an iteration during lines 4.10–4.19, we add one ball to our solution for a remaining index $j \in I_4$. In particular, we pick the index $j \in I_4$ with the highest radius r_j (line 4.11) and guess a random subset T_j of $I_1 \cup I_2$ (line 4.12). The set T_j is supposed to denote the indices $h \in I_1 \cup I_2$ such that the ball B_h intersects the optimal ball $B^\star_j$. Assume that the algorithm guesses the set T_j correctly (which again happens with probability at least $\frac{1}{2^k}$). Now two cases arise: (i) The radius r_j is at least the radius of some ball $B_h, h \in T_j$, (ii) The radius r_j is less than the latter quantity for all $h \in T_j$. Note that $\mathsf{radius}(B_h)$ is either $3r_h$ (when $h \in I_1$) or $5r_h$ (when $h \in I_2$).

In the first case (line 4.13), we identify a ball B_j containing $B_j^\star$. Let h be the index in T_j such that r_j is at least $\mathsf{radius}(B_h)$. We find the highest capacity point $x \in B(c_h, \mathsf{radius}(B_h) + r_j)$ (line 4.14) except the already chosen centers – since B_h intersects $B_j^\star$, we note that $c_j^\star$ is a possible candidates for the point x, and hence $U_x \geq U_{c_j^\star}$ Finally, we add the ball $B_j = B(x, 5r_j)$ to $\mathcal{B}_2$ (line 4.15). It is not difficult to show that B_j contains $B_j^\star$. The second case, when $r_j < \mathsf{radius}(B_h)$ for all $h \in T_j$, is more challenging. We first identify a candidate set of points P_j which contain the optimal ball $B_j^\star$ (line 4.18) and then prune enough points from it to identify a ball B_j. For each $h \in T_j$ (line 4.17), we define an *extended* ball E_h of radius $9r_h$ around c_h (clearly, E_h contains the ball B_h, which is of radius either $3r_h$ or $5r_h$ around c_h). We shall show later E_h contains $B_j^\star$. Thus, $B_j^\star \subseteq \bigcap_{h \in T_j} E_h$. By our assumption on the correct guess of the set T_j, it follows that $B_i \cap B_j^\star$ is empty for $i \in (I_1 \cup I_2) \setminus T_j$ Further, **Invariant 1** implies that for any subset C_i in $\mathcal{B}_3$, $i \in I_3$, $B_j^\star \cap C_i$ is empty. Thus, the set P_j as defined in line 4.18 contains $B_j^\star$. We now call the subroutine **UpdateBalls** to add a suitable ball corresponding to $B_j^\star$ to $\mathcal{B}_3$ (or move a ball from $\mathcal{B}_3$ to $\mathcal{B}_2$).

Algorithm 4 An iteration of the algorithm for Capacitated k-sumRadii for general capacities.

```
4.1  Input: Set $P$ of $n$ points, parameter $k$, radii $r_1, \ldots, r_k$, capacities $U_p$ for each point
     $p \in P$.
4.2  Initialize $I_1, I_2, I_3$ to emptyset and $I_4$ to $[k]$.
4.3  Initialize the sets $\mathcal{B}_1, \mathcal{B}_3, \mathcal{B}_3$ to emptyset.
4.4  while the balls in $\mathcal{B}_1$ do not cover $P$ do
4.5      Pick a point $p \in P \setminus \left(\bigcup_{j \in I_1} B_j\right)$.
4.6       (Output fail if $I_4$ is empty).
4.7      Choose $j \in I_4$ uniformly at random.
4.8      Let $x$ be the highest capacity point in $(P \setminus \{c_i : i \in I_1\}) \cap B(p, r_j)$.
4.9      Call InsertBall$(x, j, 3r_j, 1)$
4.10 while $I_4$ is non-empty do
4.11     Let $j \in I_4$ with the highest $r_j$ value.
4.12     Choose a random subset $T_j \subseteq I_1 \cup I_2$.
4.13     if there is an index $h \in T_j$ with $r_j \geq \mathsf{radius}(B_h)$ then
4.14         Let $x$ be the highest capacity point (other than $\{c_i : i \notin I_4\}$) in
              $B(c_h, \mathsf{radius}(B_h) + r_j)$.
4.15         Call InsertBall$(x, j, 5r_j, 2)$.
4.16     else
4.17         For an index $h \in T_j$, define $E_h := B(c_h, 9r_h)$.
4.18         Define $P_j := \left(\bigcap_{h \in T_j} E_h\right) \setminus \left(\bigcup_{i \in (I_1 \cup I_2) \setminus T_j} B_i \cup \bigcup_{i \in I_3} C_i\right)$.
4.19         Call UpdateBalls$(j, P_j)$.
4.20 Output the balls $\{B(c_j, 9r_j) : j \in [k]\}$
```

We now give details of the **UpdateBalls** procedure. We also emphasize that this is step where the technical novelty of our contribution lies. We assume that the sets $I_1, \ldots, I_4$, radii $r_1, \ldots, r_k$ and $\mathcal{B}_1, \ldots, \mathcal{B}_3$ can be accessed or modified by this procedure. The parameters given to this procedure are an index $j \in I_4$ and a set P_j of points which should contain the optimal ball $B_j^\star$. The procedure can terminate in the following manner: (i) find an index $i \in I_3$ and a ball containing $B_i^\star$ – in this case, we move i from I_3 to I_2 (and change $\mathcal{B}_2, \mathcal{B}_3$

accordingly), or (ii) corresponding to the index j, add a ball B_j and a subset $C_j \subseteq B_j$ to $\mathcal{B}_3$. Note that **Invariant 1** requires that C_j should be disjoint from $B_i^\star$ for all $i \in I_3 \cup I_4$. Therefore the algorithm maintains a set Z (initialized to $I_3 \cup I_4$) of indices that could potentially intersect the intended ball B_j (line 5.3). The set Z gets pruned as we run through the iterations of the **while** loop (lines 5.4–5.20). We now describe each iteration of this **while** loop. In line 5.5, we identify a point x of a high capacity such that $B(x, r_j)$ has a large intersection with P_j; and we let C_j denote this intersection. Now, we guess the subset L of Z consisting of the indices $i \in Z$ such that $B_i^\star \cap C_j$ is not empty (line 5.6). Assume that this guess is correct. If L is empty (line 5.7), we can add B_j to $\mathcal{B}_3$. The procedure terminates in this case.

Hence assume that the set L is not empty. There are two sub-cases now. In the first sub-case (line 5.12), there is an index $t \in L$ such that $r_t > r_j$. We shall show that $t \in I_3$ (recall that L is a subset of $I_3 \cup I_4$). Since B_j and $B_t^\star$ intersect, the ball $B(x, r_t + r_j)$ contains $c_t^\star$. Thus, we pick a high capacity point y in this ball (line 5.13) and add the ball $B(y, 5r_t)$ to $\mathcal{B}_2$ (line 5.14) – the fact that $r_t > r_j$ ensures that this ball contains $B_t^\star$. Thus, the index t moves from I_3 to I_2. The procedure terminates in this sub-case.

Finally, consider the sub-case when the set L is non-empty and every index $t \in L$ satisfies $r_t \leq r_j$. In line 5.17, we guess whether the ball $B_j' := B(x, 3r_j)$ intersects $B_j^\star$. Assume that this guess is correct. If $B_j' \cap B_j^\star$ is non-empty (line 5.19), then it is easy to see that the ball $B(x, 5r_j)$ contains $B_j^\star$. Thus, we add this ball to $\mathcal{B}_2$ and terminate. Otherwise, we can remove B_j from P_j (recall that P_j contains a subset of points which are guaranteed to contain $B_j^\star$). Now that P_j has shrunk (and hence, the possible set of points in the desired ball B_j also reduces), we can update the set Z. Recall that Z stores indices $i \in I_3 \cup I_4$ such that $B_i^\star$ potentially intersects $B_j^\star$. Since $r_t \leq r_j$ and $B_t^\star$ intersects $B(x, r_j)$ for all $t \in L$, it follows that $B_t^\star \subseteq B_j'$. Thus, $B_t^\star$ does not intersect the updated set P_j. Thus, we can remove the L from Z (line 5.20). This is the only sub-case where we perform another iteration of the **while** loop. Since we reduce the size of Z by at least 1, there can be at most k iterations of the **while** loop. This completes the description of our algorithm.

3.2 Analysis

We now analyse the algorithm. We begin with the following key observation about the procedure **InsertBall**:

▷ Claim 10. Suppose the algorithm calls **InsertBall** with parameters (p, j, r, u), then the point $p \neq c_i$ for any $i \in I_1 \cup I_2 \cup I_3$.

Proof. The claim follows from the fact that whenever we call **InsertBall**(p, j, r, u), we make sure p has not been selected as the center of any ball in $\mathcal{B}_1 \cup \mathcal{B}_2 \cup \mathcal{B}_3$. This can be easily seen by considering each call to this procedure:

- Line 4.9 of Algorithm 4: At this point I_2, I_3 are empty and we make sure that the point $x \neq c_i$ for any $i \in I_1$ (line 4.8).
- Line 4.15 of Algorithm 4: we ensure that $x \neq c_i, i \in [k] \setminus I_4$ in line 4.14.
- Line 5.8 or line 5.20 of Algorithm 5: in line 5.5, we make sure that $x \neq c_i, i \in [k] \setminus I_4$.
- Line 5.14 of Algorithm 5: we ensure in line 5.13 that $y \neq c_i, i \in [k] \setminus I_4$. ◁

We now bound the number of iterations in the procedure **UpdateBalls**.

▷ Claim 11. The **while** loop in a particular invocation of **UpdateBalls** has at most k iterations. The procedure **InsertBall** is called at most $O(k^2)$ times, and the number of times it chooses the option (ii) is at max k.

Algorithm 5 Procedure **UpdateBalls**(j, P_j) adds a ball to either $\mathcal{B}_2$ or $\mathcal{B}_3$.

```
5.1  Input: An index $j \in I_4$, a subset $P_j$ of points.
5.2  Initialize a variable update to false
5.3  Initialize an index set $Z := I_3 \cup I_4$.
5.4  while update is false do
5.5      Let $x$ be the point in $P \setminus \{c_i : i \notin I_4\}$ which maximizes $\min(U_x, |B(x, r_j) \cap P_j|)$.
          Define $C_j := B(x, r_j) \cap P_j$.
5.6      Pick a subset $L \subseteq Z$ uniformly at random.
5.7      if $L$ is empty then
5.8          Call InsertBall$(x, j, r_j, 3)$.
5.9          If $B(x, r_j)$ was added to $\mathcal{B}_3$, set $C_j$ as defined above in line 5.5.
5.10         Set update to true.
5.11     else
5.12         if there is an index $t \in L$ with $r_t > r_j$ then
5.13             Let $y$ be the point in $B(x, r_j + r_t) \setminus \{c_i : i \notin I_4\}$ with the maximum
                  capacity $U_y$.
5.14             Call InsertBall$(y, t, 5r_t, 2)$;
5.15             If the index $t$ was added to $I_2$, remove it from $I_3$ and remove the
                  corresponding ball from $\mathcal{B}_3$.
5.16             Set update to true.
5.17         else
5.18             Perform exactly one of the following steps with equal probability:
5.19             (i) Call InsertBall$(x, j, 5r_j, 2)$. Set update to true.
5.20             (ii) Update $Z = Z \setminus L$ and $P_j = P_j \setminus B(x, 3r_j)$.
```

Proof. First consider the **while** loop in the **UpdateBalls** procedure. If a particular iteration of this loop does not end the procedure, then it must execute line 5.20. Since the set $L \subseteq Z$ is non-empty, the set Z reduces in size during this iteration. Since the initial size of Z was at most k, there can be at most k iterations of this **while loop**.

The number of times **InsertBall** chooses the option (ii) is at max k, because every time the second option is chosen, the size of $I^\star$ increases by 1, and we never remove anything from $I^\star$. During any consecutive stretch of option (i) choices, each call to **InsertBall**, either removes an element from I_4 or moves an element from I_3 to I_2 (line 5.15). Hence, the number of consecutive calls to **InsertBall** that make the first choice can be at max $2k$. ◁

We first state the desirable events during an iteration of Algorithm 4:

- $\mathcal{E}_1$: For each point p considered in line 4.5, the index j chosen in line 4.7 satisfies the condition that $p \in C_j^\star$.
- $\mathcal{E}_2$: For each index j considered in line 4.11, the subset T_j chosen in line 4.12 satisfies the property that $T_j = \{h \in I_1 \cup I_2 : B_h \cap B_j^\star \neq \emptyset\}$.
- $\mathcal{E}_3$: For each iteration of the **while** loop in Algorithm 5, the subset L picked in line 5.6 in Algorithm 5 is equal to the set $\{i \in Z : B_i^\star \cap C_j \neq \emptyset\}$.
- $\mathcal{E}_4$: In each iteration of the **while** loop in Algorithm 5, if we reach line 5.17, then the choice (i) is taken iff the ball $B(x, 3r_j) \cap B_j^\star$ is non-empty.
- $\mathcal{E}_5$: Whenever **InsertBall**(p, j, r, u) is called, it chooses option (ii) in line 3.4 iff p is same as $c_u^\star$ for some $u \in I_2 \cup I_3 \cup I_4$. Further, the index h selected in this step is equal to u.

▷ Claim 12. If the event $\mathcal{E}_1 \cap \mathcal{E}_5$ happens, then for each $j \in I_1$, the ball $B_j \in \mathcal{B}_1$ contains the optimal ball $B^\star_j$. Hence, if $\mathcal{E}_1$ happens, the algorithm does not output **fail** in line 4.6. Further, the probability that all the events $\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_4, \mathcal{E}_5$ happen is at least $1/2^{O(k^4)}$.

Proof. The proof is similar to that of Lemma 6. Suppose $\mathcal{E}_1 \cap \mathcal{E}_5$ happens. For each index $i \in I^\star$, $B(c_i, r_i)$ contains $B^\star_i = B(c^\star_i, r^\star_i)$ because $c_i = c^\star_i$ for such indices and $r_i \geq r^\star_i$. Now if **InsertBall** executes option (ii) in line 3.4, we simply replace I_1 by $I^\star$ and start from scratch. The only other case when when we add a ball to $\mathcal{B}_1$ is during the **while** loop in line 4.4 in Algorithm 4. Consider such an iteration and assume that **InsertBall** executes option (i). Let x be the point chosen in line 4.8. Then, $d(x, c^\star_j) \leq d(x,p) + d(p, c^\star_j) = 2r_j$. Therefore, $B_j := B(x, 3r_j)$ contains $B^\star_j$. Therefore, there will be at most k iterations of the **while** loop in line 4.4. Thus, the algorithm won't output **fail** in line 4.6.

The probability that **InsertBall** always chooses the option (i) or (ii) correctly is at least $\frac{1}{2^{O(k^2)}}$, as the number of calls is $O(k^2)$ (Claim 11) and the probability of making the correct choice in a single call is $\frac{1}{2}$. Given this happens, the probability that it correctly guesses the index h is at least $1/k$ for any call where it chooses option (ii). Hence, $\Pr[\mathcal{E}_5] \geq \frac{1}{k^k} \frac{1}{2^{O(k^2)}} = \frac{1}{2^{O(k^2)}}$. Now, the probability that the chosen index j in line 4.7 is correct (i.e., $p \in C^\star_j$) in each iteration of this **while** loop is at least $1/k$. Hence, $\Pr[\mathcal{E}_1|\mathcal{E}_5] \geq 1/k^{O(k^2)}$, as each such iteration leads to a call to **InsertBall**. Conditioned on $\mathcal{E}_1$, the probability that the choice of T_j in line 4.12 is correct is at least $1/2^k$ (since there are at most 2^k possibilities for T_j). Since every iteration of the **while** loop in line 4.10 leads to a call to **InsertBall**, we see that $\Pr[\mathcal{E}_2|\mathcal{E}_1, \mathcal{E}_5] \geq \frac{1}{2^{O(k^3)}}$.

Given events $\mathcal{E}_1, \mathcal{E}_2$, whenever we reach line 5.6, we guess L correctly with probability at least $\frac{1}{2^k}$. There are at most $O(k^2)$ calls to **UpdateBalls**, and there are at most k iterations of the **while** loop in Algorithm 5. Hence, $\Pr[\mathcal{E}_3|\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_5] \geq \frac{1}{2^{O(k^4)}}$. Also, whenever reach line 5.17, we guess the choice (i) or (ii) correctly with probability 1/2. Hence, $\Pr[\mathcal{E}_4|\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_5] \geq \frac{1}{2^{O(k^3)}}$ ◁

Now we write down the invariant conditions satisfied during our algorithm. The condition **Invariant 1** was mentioned earlier, but we restate it here:

- **Invariant 1**: Let $j, j' \in I_3$ be two distinct indices. Then $C_j \cap C_{j'} = \emptyset$. Further, for any $j \in I_3$ and $i \in I_3 \cup I_4$, $B^\star_i \cap C_j = \emptyset$.
- **Invariant 2**: For any index $j \in I_1$, $B^\star_j \subseteq B_j$, where B_j is the ball corresponding to index j in $\mathcal{B}_1$. Further, the radius of B_j is at most $3r_j$.
- **Invariant 3**: For any index $j \in I_2$, $B^\star_j \subseteq B_j$, where B_j is the ball corresponding to index j in $\mathcal{B}_2$. Further, the radius of B_j is at most $5r_j$.
- **Invariant 4**: For any index $j \in I_4$, and any index $i \in I_2$, $r_j \leq r_i$.

We now show that these invariant conditions are maintained by the algorithm. The following result follows from the proof of Claim 12.

▷ Claim 13. Suppose event $\mathcal{E}_1$ happens. Then **Invariant 2** is maintained by the algorithm.

▶ **Lemma 14.** *Suppose events $\mathcal{E}_1, \ldots, \mathcal{E}_5$ occur. Then the algorithm maintains all the four invariant conditions mentioned above.*

Proof. Assume that the events $\mathcal{E}_1, \ldots, \mathcal{E}_5$ occur. Claim 13 already shows that **Invariant 2** is maintained by our algorithm. We now show that the other three invariant conditions hold by induction on the number of iterations of the **while** loop in line 4.10 in Algorithm 4. Just before executing this **while** loop for the first time, the index sets I_2, I_3 are empty, and so the three invariant conditions hold vacuously.

Now consider a particular iteration of this **while** loop and assume that the invariant conditions hold at the beginning of this iteration. During this **while** loop, we shall call **InsertBall** procedure exactly once. In this procedure, if the second option is chosen (i.e., line 3.4), then we shall insert a ball B_h in $\mathcal{B}_1$ and an index h in I_1 such that B_h contains $B_h^\star$ (since $\mathcal{E}_5$ occurs). Now it is easy to check that all the invariants continue to hold. Therefore, we shall assume that this procedure executes the first option in line 3.3.

Now, two cases arise depending on whether the condition in line 4.13 is true. First assume it is true, i.e., there is an index $h \in T_j$ such that $r_j \geq \mathsf{radius}(B_h)$. Let x be the point selected in line 4.14. Then

$$d(x, c_{j^\star}) \leq d(x, c_h) + d(c_h, c_j^\star) \leq \mathsf{radius}(B_h) + r_j + \mathsf{radius}(B_h) + r_j \leq 2 \cdot \mathsf{radius}(B_h) + 2r_j,$$

where we have used the fact that $d(c_h, c_j^\star) \leq \mathsf{radius}(B_h) + r_j^\star$ because B_h and $B_j^\star$ intersect. Since $r_j \geq \mathsf{radius}(B_h)$, the above is at most $4r_j$. It follows that $B(x, 5r_j)$ contains $B_j^\star$. This shows that **Invariant 3** is satisfied. Since j was chosen to have the highest r_j value among all indices in I_4, **Invariant 4** is also satisfied. Finally, we do not change I_3 and only remove an index from I_4. Therefore, **Invariant 1** continues to hold.

We now consider the more involved case when the outcome of the **if** condition in line 4.13 is false. In this case, $r_j < \mathsf{radius}(B_h)$ for all $h \in T_j$. We first argue that $B_j^\star \subseteq E_h$ for any $h \in T_j$ (where $E_h = B(c_h, 9r_h)$ as defined in line 4.17). To see this, note that, $B_j^\star \subseteq B(c_h, \mathsf{radius}(B_h) + 2r_j)$, as $B_j^\star$ and B_h intersect. If $h \in I_1$, from **Invariant 2**, $\mathsf{radius}(B_h) \leq 3r_h$, so $\mathsf{radius}(B_h) + 2r_j \leq 3\mathsf{radius}(B_h) \leq 9r_h$. Otherwise, $\mathsf{radius}(B_h) \leq 5r_h$ from **Invariant 3**, and $r_j \leq r_h$ from **Invariant 4**. Hence, $\mathsf{radius}(B_h) + 2r_j) \leq 5r_h + 2r_h \leq 9r_h$ Thus, $B_j^\star \subseteq \cap_{h \in T_j} E_h$. Since the event $\mathcal{E}_2$ has occurred, we know that $B_j^\star \cap B_h = \emptyset$ for all $h \in I_1 \cup I_2 \setminus T_j$, and the induction hypothesis about **Invariant 1** implies that $B_j^\star \cap C_i = \emptyset$ for all $i \in I_3$. Thus, the set P_j defined in line 4.18 of Algorithm 4 contains $B_j^\star$.

▷ Claim 15. $B_j^\star \subseteq P_j$ during the execution of the procedure **UpdateBalls**(j, P_j). Further, for every index $i \in (I_3 \cup I_4) \setminus Z$, $B_i^\star \cap P_j = \emptyset$.

Proof. We show this by induction on the number of iterations of the **while** loop in the procedure **UpdateBalls**(j, P_j). Since Z is initialized to $I_3 \cup I_4$, and $B_j^\star \subseteq P_j$ when this procedure is called, the claim is true at the beginning.

Now suppose the claim is true at the beginning of an iteration of this **while** loop. Assume that we go through one iteration of the **while** loop and come back to line 5.4. Then we must have executed the case in line 5.20. Since event $\mathcal{E}_3 \cap \mathcal{E}_4$ happens, we know that $L = \{i \in Z : B_i^\star \cap C_j \neq \emptyset\}$, $r_t \leq r_j$ for all $t \in L$ and $B(x, 3r_j) \cap B_j^\star$ is empty. Therefore, $B_j^\star \subseteq P_j' := P_j \setminus B(x, 3r_j)$. We claim that for any $t \in L$, $B_t^\star \cap P_j'$ is empty. Indeed, we know that $B_t^\star$ intersects C_j (by event $\mathcal{E}_4$) and $r_t \leq r_j$. Since $C_j \subseteq B(x, r_j)$, it follows that $B_t^\star \subseteq B(x, 3r_j)$. Therefore, $B_t^\star$ does not intersect P_j'. Thus, we see that the desired claim holds at the end of line 5.20 as well. ◁

Now we consider the various cases on how this procedure terminates:

- The ball $B(x, r_j)$ was added in line 5.8: Since event $\mathcal{E}_3$ occurred, we know that $B_i^\star \cap C_j$ is empty for all $i \in Z$. Combining this with Claim 15 and the fact that $C_j \subseteq P_j$, we see that $B_i^\star \cap C_j$ is empty for all $i \in I_3 \cup I_4$. Since P_j, as defined in line 4.18, is disjoint from $\cup_{i \in I_3} C_j$, we see that adding the index j to I_3, and $B(x, r_j)$ along with C_j to $\mathcal{B}_3$ maintains **Invariant 1**. Since we do not change I_1, I_2 and I_4 only reduces, the other three invariants continue to be satisfied.

- the ball $B_t := B(y, r_j + r_t)$ was added to $\mathcal{B}_2$ in line 5.14 (and the index t and the corresponding ball was removed from $\mathcal{B}_3$): Since we are removing a ball from $\mathcal{B}_3$, **Invariant 1** continues to be satisfied. No new ball is added to $\mathcal{B}_1$ and hence, **Invariant 2** is also satisfied. We check **Invariant 3** now: we know that $r_t > r_j$ and $B_t^\star$ intersects B_j (since $\mathcal{E}_3$ has occurred). Therefore,

$$d(y, c_t^\star) \leq d(y, x) + d(x, c_t^\star) \leq (r_j + r_t) + (r_j + r_t).$$

  Since $r_j \leq r_t$, the above is at most $4r_t$. Therefore, $B(y, 5r_t)$ contains $B_t^\star$. Thus, **Invariant 3** is satisfied. It remains to check **Invariant 4**: note that the index $j \in I_4$ was chosen because it has the highest r_j value among all the indices in I_4 (line 4.11). Since $r_t \geq r_j$, we see that **Invariant 4** is also satisfied.
- The ball $B(x, 5r_j)$ is added to $\mathcal{B}_2$ in line 5.19: Since I_1, I_3 do not change and the set I_4 shrinks, invariants **Invariant 1**, **Invariant 2** continue to be satisfied. Since $\mathcal{E}_4$ occurs, we know that $B(x, 3r_j) \cap B_j^\star$ is non-empty. Hence, $B(x, 5r_j)$ contains $B_j^\star$ and so, **Invariant 3** is satisfied. Finally, **Invariant 4** is satisfied because of the manner index $j \in I_4$ was chosen (line 4.11).

This completes the proof of the desired lemma. ◀

We now show that the centers of the balls added by our algorithm have sufficient capacities:

▶ **Lemma 16.** *Assume that the events $\mathcal{E}_1, \ldots, \mathcal{E}_5$ occur. For each ball $B_j \in \mathcal{B}_1 \cup \mathcal{B}_2 \cup \mathcal{B}_3$ centered at c_j, the capacity of c_j is at least $|C_j^\star|$.*

Proof. We first observe that the following property is maintained during the execution of the algorithm:

▷ Claim 17. For any index $i \in I_4$, the optimal center $c_i^\star$ is not the center of any ball in $\mathcal{B}_1 \cup \mathcal{B}_2 \cup \mathcal{B}_3$.

Proof. We add a ball to $\mathcal{B}_2 \cup \mathcal{B}_3$ only during case (i) of **InsertBall**. But if p happens to be $c_i^\star$ for some $i \in I_2 \cup I_3 \cup I_4$, then $\mathcal{E}_5$ implies that we invoke case (ii), a contradiction. Therefore, the claim holds. ◁

When we invoke case (ii) of **InsertBall** and a ball B_j to $\mathcal{B}_1$, **Invariant 5** implies that the center of this ball is $c_j^\star$ and hence, the Lemma holds trivially. Therefore, for rest of the proof, consider only the cases when we execute option (i) (i.e., line 3.3) when the **InsertBall** procedure is called. We now go through all such possibilities:

- We add the ball $B(x, 3r_j)$ to $\mathcal{B}_1$ in line 4.9: we know by the property shown above that $c_j^\star$ has not been used as the center of any selected ball yet. Since $p \in B_j^\star$ (by event $\mathcal{E}_1$), $c_j^\star$ is a candidate for the point x selected in line 4.8. Therefore, $U_x \geq U_{c_j^\star} \geq |C_j^\star|$.
- We insert the ball $B(x, 5r_j)$ in line 4.15: again, we know that $c_j^\star$ has not been used as a center yet. Since $B_h \cap B_j^\star$ is non-empty (since event $\mathcal{E}_2$ occurs), $d(c_h, c_j^\star) \leq \mathsf{radius}(B_h) + r_j$. Therefore, $c_j^\star$ is one of the candidates for the point x. Thus, $U_x \geq U_{c_j^\star} \geq |C_j^\star|$.
- We add a ball centered at x in line 5.8 or line 5.20 in **UpdateBalls** procedure: We know by Claim 15 that $B_j^\star \subseteq P_j$. Therefore, for $x = c_j^\star$, the quantity $\min(U_x, |B(x, r_j) \cap P_j|)$ is at least $|C_j^\star|$. Since $c_j^\star$ has not been chosen as a center yet, it follows from line 5.5 that $U_x \geq |C_j^\star|$.
- We add the index t to I_2 and a ball centered at y to $\mathcal{B}_2$ in line 5.13: Since $B_j \cap B_t^\star \neq \emptyset$ (by event $\mathcal{E}_3$), $d(x, c_t^\star) \leq r_j + r_t$. We claim that $c_t^\star$ has not been used a center of any ball in $\mathcal{B}_1 \cup \mathcal{B}_2 \cup \mathcal{B}_3$. Indeed, we know that $t \in I_3 \cup I_4$. If $t \in I_4$, Claim 17 implies that $c_t^\star$ has

not been used as a center. If $t \in I_3$, then **Invariant 1** implies that $c_t^\star$ is not contained in the ball $B_t \in \mathcal{B}_3$, and hence, has not been used a center yet. Since $d(x, c_t^\star) \le r_j + r_t$, $c_t^\star$ is a candidate for the point y chosen in line 5.13. Therefore, $C_y \ge C_{c_t^\star} \ge |C_t^\star|$. ◀

▶ **Lemma 18.** *Suppose events $\mathcal{E}_1, \dots, \mathcal{E}_5$ occur. Let $\{B(c_j, 9r_j) : j \in [k]\}$ be the set of balls output by Algorithm 4. Then there is a feasible solution to this instance where each center c_j is assigned at most U_j points, and if a point p is assigned to a center c_j, then $p \in B_j$.*

Proof. For the balls in $\mathcal{B}_1$ and $\mathcal{B}_2$, **Invariant 2** and **Invariant 3** shows that they cover the corresponding optimal balls. However this is not true for the balls in $\mathcal{B}_3$ (in fact **Invariant 1** shows quite the opposite). Thus handling these balls require more nuanced argument. For each $j \in I_3$, we claim that $|C_j| \ge |C_j^\star|$. Indeed, as the proof of Lemma 16 shows, when we define C_j as in line 5.5, $c_j^\star$ is also a candidate for the point x chosen in this step. Therefore, $\min(U_{c_j}, |C_j|) \ge \min(U_{c_j^\star}, |C_j^\star|)$, which implies that $|C_j| \ge |C_j^\star|$. Therefore, for each $j \in I_3$, let D_j be an arbitrary subset of C_j with $|D_j| = |C_j^\star|$. It follows from **Invariant 1** that $D_i \cap D_j = \phi$ for any $i, j \in I_3, i \ne j$. For any $j \in I_3, i \in T_j$, define $S_{ji} := D_j \cap C_i^\star$. Note that $\cup_{i \in T_j} S_{ji} = D_j$ because T_j is the set of indices i for which $C_j \cap C_i^\star$ is non-empty. Also, the sets S_{ji} are disjoint as $C_i^\star$ are disjoint, and therefore $\sum_{i \in T_j} |S_{ji}| = |D_j| = |C_j^\star|$. Hence, we can partition the set $C_j^\star$ as $C_j^\star = \cup_{i \in T_j} V_{ji}$, where V_{ji}'s are mutually disjoint over different i and $|V_{ji}| = |S_{ji}|$.

Now, let us define the clusters $X_1, X_2, \dots X_k$ as follows:

- For each $i \in I_3$, $X_i = D_i$
- For each $i \in I_1 \cup I_2$, $X_i = (C_i^\star \setminus \cup_{j:i \in T_j} S_{ji}) \cup (\cup_{j:i \in T_j} V_{ji})$. Note the following properties:
  - $S_{ji} = C_i^\star \cap D_j \subseteq C_i^\star$
  - $V_{ji} \subseteq C_j^\star$ meaning $V_{ji} \cap C_i^\star = \phi$
  - For for two indices $x, y \in I_3$ such that $i \in T_x \cap T_y$, $S_{xi} \cap S_{yi} = \phi$, as $D_x \cap D_y = \phi$ and $V_{xi} \cap V_{yi} = \phi$, as $C_x^\star \cap C_y^\star = \phi$

  Hence, $|X_i| = |C_i^\star| - \sum_{j:i \in T_j} |S_{ji}| + \sum_{j:i \in T_j} V_{ji} = C_i^\star$ as $|S_{ji}| = |V_{ji}|$ for j such that $i \in T_j$.

We complete the proof by claiming that X is a valid assignment, that is:

1. For each $i \in [k]$, $|X_i| \le U_{c_i}$.
2. For each $i \in [k]$, $X_i \subseteq B(c_i, 9r_i)$
3. $X_i \cap X_j = \phi$ for any $i \ne j \in [k]$ and $\cup_{i \in [k]} X_i = P$.

The first part can be proved by noting that for all $i \in [k]$, $|X_i| = |C_i^\star|$ and then using Lemma 16.

For the second part, if $i \in I_3$, $X_i = D_i \subseteq C_i \subseteq B_i = B(c_i, r_i) \subseteq B(c_i, 9r_i)$. Otherwise, note that $C_i^\star \subseteq B_i^\star \subseteq B(c_i, 5r_i) \subseteq B(c_i, 9r_i)$ holds from **Invariant 3**. Also, for any j with $i \in T_j$, note that $V_{ji} \subseteq C_j^\star \subseteq P_j \subseteq B(c_i, 9r_i)$.

For the third part, consider $i \ne j \in [k]$. There are three cases:

- $i, j \in I_3$: In this case, $X_i \cap X_j = D_i \cap D_j = \phi$
- $i, j \in I_1 \cup I_2$: In this case, $C_i^\star \cap C_j^\star = \phi$, and for any two indices x, y, with $i \in T_x$ and $j \in T_y$, $V_{xi} \cap V_{yj} = \phi$ because if $x \ne y$, then $V_{xi} \subseteq C_x^\star$ and $V_{yj} \subseteq C_y^\star$, and otherwise if $x = y$, V_{xi} and V_{yj} are disjoint sets in a partition of $C_x^\star$
- $i \in I_1 \cup I_2, j \in I_3 \cup I_2$ or vice versa. Assume wlog that $i \in I_1 \cup I_2, j \in I_3$. Here, $D_j \cap (C_i^\star \setminus \cup_{l:i \in T_l} S_{li}) = \phi$ because we remove $S_{ji} = D_j \cap C_i^\star$ from $C_i^\star$ when forming X_i. Also, $D_i \cap V_{li} = \phi$ for any l with $i \in T_l$, because $V_{li} \subseteq C_l^\star$, $D_j \subseteq C_j$ and $C_j \cap C_l^\star = \phi$, from **Invariant 1**.

Now, since the clusters are disjoint and $|X_i| = |C_i^\star|$ for all $i \in [k]$, $|\cup_{i\in[k]} X_i| = \sum_{i\in[k]} |X_i| = \sum_{i\in[k]} |C_i^\star| = |P|$. This means that $\cup_{i\in[k]} X_i = P$, as $X_i \subseteq P$ for all $i \in [k]$. ◀

▶ **Lemma 19.** *The total cost of the solution produced by Algorithm 4 is at most* $9\sum_{j\in[k]} r_j$. *Further, the total* L_p *norm of the radii of the clusters produced by this algorithm is at most* $9\left(\sum_{j\in[k]} r_j^p\right)^{1/p}$.

Proof. Clearly, the sum of p^{th} powers of the radii of the balls in the algorithm's output is: $\sum_{j\in[k]} (9r_j)^p = 9^p \sum_{j\in[k]} r_j^p$. ◀

It follows from Claim 12, Lemma 18 and Lemma 19 that the algorithm outputs a $(9+\varepsilon)$ approximate solution with probability at least $1/2^{O(k^4)}$. This probability bound can be improved to $1/2^{O(k^3)}$ through a more fine-grained analysis. For this, note that the invariants $1, 3$ and 4 only need to be satisfied after the last call to **InsertBall** that chooses option (ii). This allows us to relax the definitions of $\mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_5$, such that $\Pr[\mathcal{E}_1 \cap \mathcal{E}_2 \cap \mathcal{E}_3 \cap \mathcal{E}_4 \cap \mathcal{E}_5] \geq 1/2^{O(k^3)}$

3.3 Improvement in approximation ratio from 9 to $4+\sqrt{13}$

In this section, we briefly show that with a more careful choice of parameters, we can improve the approximation ratio from 9 to about 7.606. We use a parameter α that shall be fixed later. We make the following changes in Algorithm 4:

- The condition in line 4.13 becomes "$r_j \geq \alpha \cdot \mathsf{radius}(B_h)$".
- Line 4.14 remains unchanged, i.e., x is the maximum capacity point in $B(c_h, \mathsf{radius}(B_h) + r_j)$ (other than $\{c_i : i \notin I_4\}$). But line 4.15 changes to "Call **UpdateBalls**$(x, j, 2\cdot \mathsf{radius}(B_h) + 3r_j, 2)$."
- The definition of the ball E_h in line 4.17 changes. Recall that h belongs to the set T_j here. Now we differentiate between two cases. When $h \in I_1$, then we define E_h as $B(c_h, 3(1+2\alpha)r_h)$. The other case is when $h \in I_2$. In this case, we define E_h as $B(c_h, (5+\frac{2}{\alpha})r_h)$.
- In line 4.20, we now output $\{B(c_j, 3(1+2\alpha)r_j) : j \in I_1\} \cup \{B(c_j, (5+\frac{2}{\alpha})r_j) : j \in I_2\} \cup \mathcal{B}_3$

A routine modification of the analysis in the previous section shows that the approximation ratio now becomes $\max(3(1+2\alpha), 5+\frac{2}{\alpha})$, whose minimum possible value is $4+\sqrt{13} \approx 7.606$ at $\alpha = \frac{1+\sqrt{13}}{6}$.

References

1 Sayan Bandyapadhyay, William Lochet, and Saket Saurabh. FPT constant-approximations for capacitated clustering to minimize the sum of cluster radii. In Erin W. Chambers and Joachim Gudmundsson, editors, *39th International Symposium on Computational Geometry, SoCG 2023, June 12-15, 2023, Dallas, Texas, USA*, volume 258 of *LIPIcs*, pages 12:1–12:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.SoCG.2023.12.

2 Sayan Bandyapadhyay and Kasturi Varadarajan. Approximate Clustering via Metric Partitioning. In Seok-Hee Hong, editor, *27th International Symposium on Algorithms and Computation (ISAAC 2016)*, volume 64 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:13, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ISAAC.2016.15.

3 B. Behsaz and M.R. Salavatipour. On minimum sum of radii and diameters clustering. *Algorithmica*, 73:143–165, 2015. doi:10.1007/s00453-014-9907-3.

4 Mihai Bădoiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 250–257, New York, NY, USA, 2002. Association for Computing Machinery. doi:10.1145/509907.509947.

5 Moses Charikar and Rina Panigrahy. Clustering to minimize the sum of cluster diameters. *J. Comput. Syst. Sci.*, 68(2):417–441, 2004. doi:10.1016/j.jcss.2003.07.014.

6 C.L.Monma and S.Suri. Partitioning points and graphs to minimize the maximum or the sum of diameters. *Graph Theory, Combinatorics and Applications, pages 880-912*, 1991.

7 Irit Dinur. The pcp theorem by gap amplification. *J. ACM*, 54(3):12–es, June 2007. doi:10.1145/1236457.1236459.

8 Srinivas Doddi, Madhav V. Marathe, S. S. Ravi, David Scot Taylor, and Peter Widmayer. Approximation algorithms for clustering to minimize the sum of diameters. *Nord. J. Comput.*, 7(3):185–203, 2000.

9 Zachary Friggstad and Mahya Jamshidian. Improved Polynomial-Time Approximations for Clustering with Minimum Sum of Radii or Diameters. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms (ESA 2022)*, volume 244 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 56:1–56:14, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2022.56.

10 Matt Gibson, Gaurav Kanade, Erik Krohn, Imran A. Pirwani, and Kasturi Varadarajan. On metric clustering to minimize the sum of radii. *Algorithmica*, 57:484–498, 2010. doi:10.1007/s00453-009-9282-7.

11 Matt Gibson, Gaurav Kanade, Erik Krohn, Imran A. Pirwani, and Kasturi Varadarajan. On clustering to minimize the sum of radii. *SIAM Journal on Computing*, 41(1):47–60, 2012. doi:10.1137/100798144.

12 Pierre Hansen and Brigitte Jaumard. Cluster analysis and mathematical programming. *Math. Program.*, 79:191–215, 1997. doi:10.1007/BF02614317.

13 M. Henzinger, D. Leniowski, and C. Mathieu. Dynamic clustering to minimize the sum of radii. *Algorithmica*, 82:3183–3194, 2020. doi:10.1007/s00453-020-00721-7.

14 Tanmay Inamdar and Kasturi R. Varadarajan. Capacitated sum-of-radii clustering: An FPT approximation. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 62:1–62:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.62.