JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 14, No. 1, March 2020, pp. 1-8

Minimum-Width Cuboidal Shells with Outliers

Sang Won Bae*
Division of Computer Science and Engineering, Kyonggi University, Suwon, Korea
swbae@kgu.ac.kr

Abstract

A (hyper-)cuboid, also known as a hyper-rectangle or a box, is a compact body of dimension three or higher, extending its two-dimensional analog, rectangles. A cuboidal shell is the compact volume between a cuboid and its inward offset. In this paper, we address the problem of computing a minimum-width cuboidal shell that encloses at least $n - k$ points out of n given points in $\mathbb{R}^d$ when $d \geq 3$. The number k is given as input and the k excluded points as a result are considered outliers of the n input points. Prior to our work, there was no known algorithm for the cuboidal shell problem considering outliers. We solve the problem for the first time by presenting two efficient algorithms. Our algorithms run in $O(k^{2d} n)$ time and $O(n)$ space or in $O(n \log^{d-1} n + k^{2d} \log^d n)$ time and $O(n \log^{d-1} n)$ space.

Category: Computer Graphics / Image Processing

Keywords: Algorithm; Computational geometry; High-dimensional geometry; Cuboidal shell; Outlier

I. INTRODUCTION

A family of optimization problems in computational geometry ask to find a certain geometric shape, such as a circle, a square, or a rectangle, that encloses a given set of points and optimizes a predefined objective function on the shapes. The *smallest enclosing circle* problem is such a well-known example, which involves determination of the smallest circle enclosing a given set of points in the plane. Encompassing a wide range of applications in multiple areas, including shape recognition, facility location, and data analysis, these problems have been studied extensively in computational geometry.

As a variation and an extension of these geometric problems, determination of a minimum shape covering all but a few input points is of great interest in view of outlier removal. More precisely, given n input points and an integer $k \geq 0$, we want to find a smallest shape enclosing at least $n - k$ out of the n input points. From the viewpoint of optimization, excluding the k points reduces the objective value the most, given the relatively high cost of inclusion. In this sense, such excluded points are considered as *outliers* of the given point set. From the concept of outliers, the input number k is often assumed to be very small relatively to the number n of data points, such as a constant or at most $k = O(\log n)$.

Specifically, the problem of finding an axis-parallel square or rectangle that encloses n given points in the plane $\mathbb{R}^2$ but excludes k outliers was of great interest in the 1990's. Aggarwal et al. [1] presented algorithms of running time $O((n-k)^2 n \log n)$ both for square and rectangle cases. For the rectangle case, Segal and Kedem [2] presented an $O(n + k^2(n-k))$-time algorithm and Atanassov et al. [3] and Ahn et al. [4] introduced an improvement to $O(n + k^3)$ time. For the square case, Chan [5] presented a randomized algorithm that runs in $O(n \log n)$ expected time, and was improved later by Ahn et al. [4] to $O(n + k \log k)$ time. More general frameworks

Open Access http://dx.doi.org/10.5626/JCSE.2020.14.1.1 http://jcse.kiise.org

Received 17 February 2020; **Accepted** 01 March 2020
*Corresponding Author

 pISSN: 1976-4677 eISSN: 2093-8020

to handle outliers as violations of constraints for LP-type optimization problems have also been introduced by Matoušek [6] and Chan [7]. The computation of a minimum enclosing circle, square, or rectangle falls into the LP-type problems.

Another recent popular shape studied in this research stem is the *annulus*, which is informally a ring-shaped region between two concentric circles. The width of an annulus is the difference in radii of the two circles defining the annulus. The *minimum-width annulus problem* seeks to determine an annulus of minimum width that encloses n input points. After early work on the problem with $O(n^2)$-time algorithms [8-10], the first subquadratic $O(n^{8/5+\varepsilon})$- time algorithm was proposed by Agarwal et al. [11] and the currently best algorithm takes $O(n^{3/2+\varepsilon})$ time by Agarwal and Sharir [12]. The minimum-width annulus problem can be extended by considering annuli of different shapes such as squares and rectangles. A square/rectangular annulus is defined as a closed region between a square/rectangle and its offset, and the width of such an annulus is defined by the distance between the two squares/rectangles. Abellanas et al. [13] presented an $O(n)$-time algorithm for the rectangular annulus problem and considered several variations of the problem. Gluchshenko et al. [14] proposed an $O(n \log n)$-time algorithm for the square annulus, and proved that this is optimal. Fig. 1 illustrates rectangular annuli.

The minimum-width annulus problem with outliers has also been studied recently. The author considered the square and rectangular annulus problem with outliers and presented the first algorithms [15]. Later, these algorithms were improved by Ahn et al. [16]. Specifically, given n points in the plane and an integer $k \geq 0$, a minimum-width axis-parallel square annulus that encloses at least $n - k$ input points can be computed in $O(k^2 n \log n)$ time and a minimum-width axis-parallel rectangular annulus with the same property can be computed in $O(n \log n + k^4 \log^2 n)$ or $O(nk^2 \log k + k^4 \log^2 k)$ time. See Fig. 1(b).

In this paper, we consider the extension of the minimum-width rectangular annulus into three or higher dimensions with outliers, namely the *minimum-width cuboidal shell problem with outliers.* A *cuboid*, also often called a box, a hyper-rectangle, or an orthotope, is represented by a Cartesian product of three or more intervals, while a rectangle is represented by a Cartesian product of two intervals. A *cuboidal shell* is a closed volume between a cuboid and its inward offset. Specifically, given a set P of n points in $\mathbb{R}^d$ for $d \geq 2$ and an integer $k \geq 0$, the k-CUBSHELL problem asks to find a cuboidal shell of minimum width that encloses at least $n - k$ points in P. Note that for $d = 2$ the k-CUBSHELL problem is equivalent to the minimum-width rectangular annulus with k outliers.

We present two algorithms to solve the k-CUBSHELL problem in $O(k^{2d}n)$ and $O(n \log^{d-1} n + k^{2d} \log^d n)$ time respectively, for any fixed $d \geq 3$ and any input k. To our best knowledge, no nontrivial algorithm was known for the problem with $d \geq 3$ and $k \geq 1$. Note that k is assumed to be very small relative to n. When no outlier was allowed, that is, $k = 0$, a linear-time algorithm for the problem was presented by Mukherjee et al. [17].

The remainder of the paper is organized as follows. After some preliminaries are given in Section II, algorithms for the k-CUBSHELL problem are described and analyzed in Sections III and IV. Finally, Section V concludes the paper with remarks and discussion.

II. PRELIMINARIES

In this paper, we consider the Euclidean space $\mathbb{R}^d$ for fixed $d \geq 3$ with d orthogonal axes, called the $x_1, x_2, \ldots, x_d$*−axes*, so any point $p \in \mathbb{R}^d$ is represented by a tuple of d coordinates $(x_1, x_2, \ldots, x_d)$. Throughout the paper, we mean by a *(hyper-)cuboid* a compact subset defined to be a Cartesian product of d closed intervals:

$$[a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_d, b_d] \subset \mathbb{R}^d$$

such that $a_i \leq b_i$ for each $i = 1, \ldots, d$. Hence, a cuboid is determined by two points $a = (a_1, \ldots, a_d)$, $b = (b_1, \ldots, b_d) \in \mathbb{R}^d$ as shown in Fig. 2(a). A cuboid is also represented by the intersection of $2d$ half-spaces whose bounding hyperplane is orthogonal to one of the axes: more

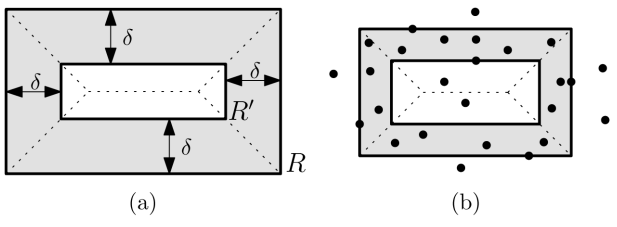

Fig. 1. Illustrations of (a) a rectangular annulus defined by a rectangle R and its offset R' by δ (shaded area), (b) a minimum-width rectangular annulus enclosing all input points but 2 inside outliers and 5 outside outliers. Note that a rectangle is a 2-dimensional cuboid and a rectangular annulus is a 2-dimensional analog of a cuboidal shell.

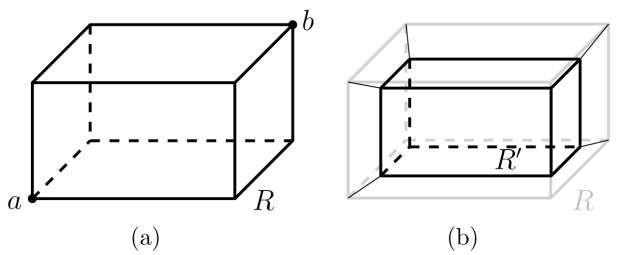

Fig. 2. (a) A 3-dimensional cuboid R. (b) A 3-dimensional cuboidal shell defined by outer cuboid R (gray) and its offset R' (black).

precisely, if a cuboid is determined by two points a and b as above, it is equal to the set

$$\{(x_1,...,x_d) \in \mathbb{R}^d \mid a_i \le x_i \le b_i,\ i = 1,...,d\},$$

where each of the $2d$ inequalities corresponds to such a half-space orthogonal to an axis. Note that when $d = 2$, a cuboid is known as a rectangle in the plane $\mathbb{R}^2$.

Consider a cuboid R defined by two points $a, b \in \mathbb{R}^d$. Note that R has $2d$ facets and 2^d vertices. For $i = 1,...,d$, we call $b_i - a_i$ the x_i-*span* of R. The *dimension* of R is d minus the number of its x_i-spans that are zero; R is a d-dimensional cuboid if none of its x_i-spans is zero.

Suppose that R is a d-dimensional cuboid and w is the minimum value of the x_i-spans of R over $i = 1,...,d$. The *(inward) offset* of R by δ for any $0 \le \delta \le w/2$ is a cuboid obtained by sliding the $2d$ facets of R inwards by δ. The offset of R by $\delta = w/2$ is degenerated to a cuboid of dimension less than d. Fig. 1(a) will be helpful for intuitive understanding of the inward offset of R by δ.

For any positive $\delta \le w$, consider the inward offset R' of R by δ. Then, the closed volume A between R and R', including its boundary, is called a *cuboidal shell* with the *outer cuboid* R and the *inner cuboid* R', as shown in Fig. 2(b). Note that when $d = 2$, a cuboidal shell is called a *rectangular annulus* as shown in Fig. 1. The distance δ between R and R' is the *width* of the cuboidal shell A. The complement $\mathbb{R}^d \setminus A$ of the shell A is separated into two connected components. We shall call the outside of R the *outside* of A and the inside of R' the *inside* of A.

Given a set P of n points in $\mathbb{R}^d$ and a non-negative integer k, the k-CUBSHELL problem asks to find a minimum-width cuboidal shell that encloses at least $n - k$ points of P. The k points that are not covered by the resulting shell are called *outliers*. Since the complement of any cuboidal shell is separated into its inside and outside, such an outlier may lie either in the inside or the outside of the shell. We call an outlier an *outside outlier* if it lies in the outside of the resulting shell, or an *inside outlier*, otherwise. In some applications, no inside outlier would be allowed while outside outliers are allowed, and vice versa, or even the numbers of inside and outside outliers are prescribed. This motivates to a variation of the problem, called the $(k_{\text{in}}, k_{\text{out}})$-CUBSHELL problem for non-negative integers k_{in} and k_{out}, in which at most k_{in} inside outliers and at most k_{out} outside outliers are allowed.

When no outlier is allowed, i.e., $k = 0$, the 0-CUBSHELL problem can be solved in $O(n)$ time by Mukherjee et al. [17]. Their linear-time algorithm is based on the following observation, which will be useful for further discussions.

LEMMA 1 (Mukherjee et al. [17]). *There exists a minimum-width cuboidal shell A enclosing all points in a given set P of points in $\mathbb{R}^d$ such that the outer cuboid of A is the smallest cuboid enclosing P.* □

III. FIRST ALGORITHM

In this section, we present a first algorithm for the k-CUBSHELL problem and the $(k_{\text{in}}, k_{\text{out}})$-CUBSHELL problem. Recall that the dimension $d \ge 3$ is a fixed constant and we are given a set P of n points in $\mathbb{R}^d$.

Note that the k-CUBSHELL problem can be solved by solving $k + 1$ instances of the $(k_{\text{in}}, k_{\text{out}})$-CUBSHELL problem. Any feasible solution to the k-CUBSHELL problem is a cuboidal shell with k_{in} inside outliers and at most $k - k_{\text{in}}$ outside outliers. Hence, an algorithm for the $(k_{\text{in}}, k_{\text{out}})$-CUBSHELL problem can be used to develop an optimal solution to the k-CUBSHELL problem by invoking it for the $(k_{\text{in}}, k - k_{\text{in}})$-CUBSHELL problem for $0 \le k_{\text{in}} \le k$.

Thus, we mainly discuss the $(k_{\text{in}}, k_{\text{out}})$-CUBSHELL problem. We start with a crucial observation.

LEMMA 2. *There exists a minimum-width cuboidal shell that is an optimal solution to the $(k_{\text{in}}, k_{\text{out}})$-CUBSHELL problem for points P such that each facet of its outer cuboid contains at least one point in P.*

Proof. Let A be any minimum-width cuboidal shell that is an optimal solution to the $(k_{\text{in}}, k_{\text{out}})$-CUBSHELL problem for point set P. Let $P' := P \cap A$ be the set of points in P that are contained in A. As required, we have $|P'| \ge n - (k_{\text{in}} + k_{\text{out}})$.

Now, consider a minimum-width cuboidal shell A' that contains all points in P' such that its outer cuboid R' is the smallest cuboid enclosing all points of P'. Such a cuboidal shell A' is guaranteed to exist by Lemma 1. Since R' is the smallest cuboid enclosing P', each side of R' must contain at least one point of P'. Note that the width of A' is at most that of A by definition.

The last step of our proof is done by observing that A' is also an optimal solution to our original $(k_{\text{in}}, k_{\text{out}})$-CUBSHELL problem. Let Q_{in} and Q_{out} be the sets of inside and outside outliers of P with respect to A. Similarly, let Q'_{in} and Q'_{out} be the sets of inside and outside outliers of P with respect to A'. First, observe that $Q'_{\text{out}} = Q_{\text{out}}$ since the outer cuboid R' of A' is the smallest cuboid enclosing $P' = P \cap A$. However, note that $P \cap A = P' = P \setminus (Q_{\text{out}} \cup Q_{\text{in}})$ and $P \cap A = P' \subseteq P \cap A' = P \setminus (Q'_{\text{out}} \cup Q'_{\text{in}})$. We thus have

$$\begin{aligned} Q'_{\text{in}} &= (P \setminus Q'_{\text{out}}) \setminus (P \cap A') \\ &\subseteq (P \setminus (Q_{\text{out}}) \setminus (P \cap A) \\ &= Q_{\text{in}}, \end{aligned}$$

since $(P \cap A, Q_{\text{in}}, Q_{\text{out}})$ forms a partition of P and so does $(P \cap A', Q'_{\text{in}}, Q'_{\text{out}})$. Hence, we conclude that $|Q'_{\text{in}}| \le |Q_{\text{in}}| \le k_{\text{in}}$, which means that A' is also a feasible solution to the original $(k_{\text{in}}, k_{\text{out}})$-CUBSHELL problem for a point set P. This implies that A' is another optimal cuboidal shell that satisfies the condition stated in the lemma. □

The above lemma enables us to consider a finite

number of multiple candidate outer cuboids. Note that a cuboid is bounded by $2d$ facets or hyperplanes that are orthogonal to each of the d axes in $\mathbb{R}^d$. Thus, our approach is to determine the best outer cuboid among those candidates by testing each of them. If we fix a candidate outer cuboid, the problem is reduced to finding a best inner cuboid for the fixed outer one. This problem is formulated into a more general form, called the *offset cuboid problem.*

A. The Offset Cuboid Problem

In the offset cuboid problem, we are given a set P of n points in $\mathbb{R}^d$, a d-dimensional cuboid R, and an integer $k_{\text{in}} \geq 0$, and we want to find the largest inward offset R' of R that contains the maximum k_{in} points of P in its interior.

This problem can be easily solved in $O(n)$ time, independently of k_{in}, as follows. For each $p \in P \cap R$, let $\delta(p, R) \geq 0$ be the real number such that the offset of R by $\delta(p, R)$ contains p on its boundary. Note that those $p \in P$ lying outside of R can be ignored to solve the offset cuboid problem. Suppose that p is the point with the m-th largest $\delta(p, R)$ value among points in $P \cap R$. Then, the offset of R by $\delta(p, R)$ is the largest offset of R containing the maximum $m - 1$ points of P in its interior. Hence, the offset cuboid problem can be solved by selecting the $(k_{\text{in}} + 1)$-th largest value among $\delta(p, R)$ over $p \in P \cap R$.

LEMMA 3. *Given a set P of n points in $\mathbb{R}^d$, a d-dimensional cuboid R, and an integer $k_{\text{in}} \geq 0$, the offset cuboid problem can be solved in $O(n)$ time and $O(n)$ space.*

Proof. First, we collect all the points in $P \cap R$ in $O(n)$ time. For all $p \in P \cap R$, the values of $\delta(p, R)$ are computed and stored in an array in $O(n)$ time. Finally, the $(k_{\text{in}} + 1)$-th largest value among the array is determined by executing any linear-time selection algorithm such as the median-of-medians algorithm described in [18]. The time and space spent is $O(n)$.□

We can devise algorithms for the $(k_{\text{in}}, k_{\text{out}})$-CUBSHELL problem by using the algorithm described in Lemma 3 above for the offset cuboid problem as a subroutine. The resulting offset of R *is observed*, together with R, forming a minimum-width rectangular annulus with the maximum k_{in} inside outliers. In case where no outside outlier is allowed, i.e., $k_{\text{out}} = 0$, an optimal cuboidal shell exists with its outer cuboid being the smallest cuboid enclosing P by Lemma 2. Thus, this special case can be handled by solving the offset cuboid problem with the smallest cuboid enclosing P in $O(n)$ time.

LEMMA 4. *For any fixed $d \geq 3$, a set P of n points in $\mathbb{R}^d$, and an integer $k_{\text{in}} \geq 0$, the $(k_{\text{in}}, 0)$-CUBSHELL problem can be solved in $O(n)$ time and $O(n)$ space.*

Proof. The smallest cuboid enclosing P can be computed in $O(n)$ time by computing the maximum and the minimum values of the coordinates of n points in P. We then apply Lemma 3 for the set P of n points and integer k_{in} with the cuboid enclosing P obtained above. The time and space spent is bounded by $O(n)$ by Lemma 3. □

B. The First Algorithm

We then solve the general case of $k_{\text{out}} > 0$. For each $i = 1,...,d$, let $X_i = (X_i[0],...,X_i[n-1])$ be the sorted array consisting of the x_i-coordinates of n points in P, so that $X_i[0] \leq X_i[1] \leq \cdots \leq X_i[n-1]$. Consider any hyperplane H of $\mathbb{R}^d$ that is orthogonal to the x_i-axis, that is, represented by $\{x_i = a\}$ for some $a \in \mathbb{R}$. Such a hyperplane H is uniquely determined by a x_i-coordinate. Suppose that the x_i-coordinate of H is equal to $X_i[j]$ for some $0 \leq j < n$. For each $1 \leq i \leq d$ and $0 \leq j \leq n-1$, let $H_{i,j}$ be the hyperplane in $\mathbb{R}^d$ represented as $H_{i,j} = \{x_i = X_i[j]\}$, that is, $H_{i,j}$ is orthogonal to the x_i-axis and its x_i-coordinate is equal to $X_i[j]$. Let $H^+_{i,j}$ and $H^-_{i,j}$ be the two open half-spaces separated by $H_{i,j}$ in the $+x_i$-direction and $-x_i$-direction, respectively. $H^-_{i,j}$ consists of at most j points in P and $H^+_{i,j}$ consists of the maximum $n-j-1$ points in P.

By selecting $2d$ indices $j^-_1, j^-_2, \ldots, j^-_d, j^+_1, \ldots, j^+_d$, with $0 \leq j^-_1 \leq j^+_1 \leq n-1$ for each $1 \leq i \leq d$, a cuboid $R(j^-_1, \ldots, j^-_d, j^+_1, \ldots, j^+_d)$ is defined as the intersection of $2d$ half-spaces, namely,

$$\begin{aligned} &R(j^-_1,\ldots,j^-_d,j^+_1,\ldots,j^+_d) \\ &\quad := \bigcap_{i=1,\ldots,d} H^-_{i,j^-_i} \cap H^+_{i,j^+_i} \\ &\quad = [X_1[j^-_1], X_1[j^+_1]] \times \cdots \times [X_d[j^-_d], X_d[j^+_d]]\,. \end{aligned}$$

In order to solve our problem, by Lemma 2, it suffices to try all cuboids $R(j^-_1,\ldots,j^-_d,j^+_1,\ldots,j^+_d)$ for all combinations of $2d$ indices as outer cuboids such that $R(j^-_1,\ldots,j^-_d,j^+_1,\ldots,j^+_d)$ contains at least $n - k_{\text{out}}$ points in P. Based on the above discussion,

$$\sum_{i=1,\ldots,d} j^-_1 + \sum_{i=1,\ldots,d} n-1-j^+_i \leq k_{\text{out}}\,.$$

Hence,

$$0 \leq j^-_i \leq k_{\text{out}} \text{ and } n-1-k_{\text{out}} \leq j^+_i \leq n-1,$$

for each $i = 1,...,d$,

This implies that the $(k_{\text{in}}, k_{\text{out}})$-CUBSHELL problem is reduced to $O(k^{2d}_{\text{out}})$ instances of the case of no outside outlier, yielding an $O(k^{2d}_{\text{out}} n)$-time algorithm by Lemma 4. More precisely, this can be done by iterating all possible combinations of $2d$ indices $(j^-_1,\ldots,j^-_d,j^+_1,\ldots,j^+_d)$ and solving the $(k_{\text{in}}, 0)$-CUBSHELL problem for point set

```
1: Algorithm MinWidthCuboidalShellOutliers(P, k_in, k_out)
2:     for each i = 1 to d do
3:         Specify 2k_out + 2 entries of X_i: X_i[0], ..., X_i[k_out] and X_i[n − 1 − k_out], ..., X_i[n − 1].
4:     end for
5:     A* ← any cuboidal shell with very large width.
6:     for each (j_1^−, ..., j_{d−1}^−, j_1^+, ..., j_{d−1}^+) with 0 ≤ j_i^− ≤ k_out and n − 1 − k_out ≤ j_i^+ ≤ n − 1 do
7:         for j_d^− ← 0 to k_out do
8:             Compute the index j* for (j_1^−, ..., j_d^−, j_1^+, ..., j_{d−1}^+).
9:             Construct the cuboid R = R(j_1^−, ..., j_d^−, j_1^+, ..., j_{d−1}^+, j*).
10:            Solve the (k_in, 0)-CubShell problem for P ∩ R and let R′ be the resulting cuboid.
11:            Let A be the cuboidal shell defined by outer cuboid R and inner cuboid R′.
12:            If the width of A is smaller then A*, then A* ← A.
13:        end for
14:    end for
15:    return A*
16: end Algorithm
```

Fig. 3. Algorithm for the (k_{in}, k_{out})-CubShell problem.

$P \cap R(j_1^-,\ldots,j_d^-,j_1^+,\ldots,j_d^+)$ by applying Lemma 4. Since $0 \le j_i^- \le k_{out}$ and $n-1-k_{out} \le j_i^+ < n-1$, the number of such combinations is bounded by $O(k_{out}^{2d})$, which results in time complexity $O(k_{out}^{2d}n)$ for any $k_{out} > 0$.

In the following we show that the number of combinations of $2d$ indices tested as candidates can indeed be reduced to $O(k_{out}^{2d-1})$, so that the (k_{in}, k_{out})-CubShell problem can be solved in $O(k_{out}^{2d-1}n)$ time, for any $k_{out} > 0$.

For any fixed $2d-1$ indices,

$$0 \le j_1^-,\ldots,j_d^- \le k_{out}$$

and

$$n-1-k_{out} \le j_1^+,\ldots,j_{d-1}^+ \le n-1,$$

let j^* be the smallest index such that the number of points in $P \cap R(j_1^-,\ldots,j_d^-,j_1^+,\ldots,j_{d-1}^+,j^*)$ is at least $n-k_{out}$. Such an index j^* is uniquely determined by the $2d-1$ indices $(j_1^-,\ldots,j_d^-,j_1^+,\ldots,j_{d-1}^+)$. Also, for any index $j' > j^*$ the number of points in $P \cap R(j_1^-,\ldots,j_d^-,j_1^+,\ldots,j_{d-1}^+,j^*)$ is at most the number of those in $P \cap R(j_1^-,\ldots,j_d^-,j_1^+,\ldots,j_{d-1}^+,j')$ since $X_d[j^*] \le X_d[j']$, and thus

$$R(j_1^-,\ldots,j_d^-,j_1^+,\ldots,j_{d-1}^+,j^*)$$

$$\subseteq R(j_1^-,\ldots,j_d^-,j_1^+,\ldots,j_{d-1}^+,j').$$

This implies that the optimal cuboidal shell for the (0, k_{in})-CubShell problem is not worse for points $P \cap R(j_1^-,\ldots,j_d^-,j_1^+,\ldots,j_{d-1}^+,j^*)$ than for points $P \cap R(j_1^-,\ldots,j_d^-,j_1^+,\ldots,j_{d-1}^+,j')$. Hence, we can ignore those combinations $(j_1^-,\ldots,j_d^-,j_1^+,\ldots,j_{d-1}^+,j')$ for all $j^* < j' \le n-1$. In this way, we are done by checking $O(k_{out}^{2d-1})$ combinations of $2d$ indices, and hence the same number of candidate outer cuboids.

Now, we describe our algorithm in a precise way. See Fig. 3 for a pseudocode of the algorithm.

First, we specify only $2k_{out}+2$ entries of the sorted array X_i for each $i = 1,\ldots,d$, namely, $X_i[0],\ldots,X_i[k_{out}]$, $X_i[n-1-k_{out}],\ldots,X_i[n-1]$. Note that we do not explicitly sort all points to identify the sorted arrays X_i, while this can be done in $O(n+k_{out}\log k_{out})$ time by selecting the k_{out}-th and $(n-1-k_{out})$-th value in X_i and sorting only $O(k_{out})$ values. We then consider all possible combinations of $2d-2$ indices $(j_1^-,\ldots,j_{d-1}^-,j_1^+,\ldots,j_{d-1}^+)$ by iterating $2d-2$ nested loops, which are compressed into line 6 in the pseudocode of Fig. 3. In the innermost loop, we increase j_d^- from 0 to k_{out} and for each j_d^- we compute the corresponding value of j^*. This innermost loop takes $O(k_{out})$ time to specify j^* for all $0 \le j_d^- \le k_{out}$ thanks to the monotonicity: as j_d^- increases, j^* increases or stays.

Thus, we can enumerate all $O(k_{out}^{2d-1})$ candidate combinations of $2d$ indices in $O(1)$ amortized time per each. For each of them, we constructed the corresponding outer cuboid $R(j_1^-,\ldots,j_d^-,j_1^+,\ldots,j_{d-1}^+,j^*)$ and apply Lemma 4.

The results in the following theorem.

Theorem 1. *Given n points and two integers* $k_{in}, k_{out} \ge 0$, *the* (k_{in}, k_{out})-CubShell *problem can be solved in* $O((k_{out}+1)^{2d-1}n)$ *time and* $O(n)$ *space. Therefore, the k-*CubShell *problem can be solved in* $O((k+1)^{2d}n)$ *time and* $O(n)$ *space for any* $k \ge 0$.

Proof. As discussed above, our algorithm consists of $2d-2$ nested loop for $2d-2$ indices running from 0 to k_{out} or from $n-1-k_{out}$ to $n-1$ and the innermost loop takes $O(k_{out}n)$ time by Lemma 4. Hence, the total time complexity is bounded by $O(k_{out}^{2d-1}n)$ for any $k_{out} > 0$ and $k_{in} \ge 0$. Together with Lemma 4, our algorithm solves the (k_{in}, k_{out})-CubShell problem in $O((k_{out}+1)^{2d-1}n)$ time

for any $k_{in}, k_{out} \geq 0$.

Recall that the k-CUBSHELL problem can be solved by solving $k + 1$ instances of (k_{in}, k_{out})-CUBSHELL problem with $k_{in} + k_{out} = k$. Therefore, the k-CUBSHELL problem can be solved in $O((k+1)^{2d}n)$ time for any $k \geq 0$.

It is not difficult to see that the space used in our algorithm is bounded by $O(n)$. □

IV. SECOND ALGORITHM

As discussed above, the offset cuboid problem is a central subproblem for our purpose, which needs to be solved a heavy number of times. Though an instance of the offset cuboid problem can be solved in optimal linear time, if we can do it more efficiently by paying some preprocessing cost, then we will be able to obtain more efficient algorithms for our main problem for non-constant k.

Here, we present a data structure for the *offset cuboid query* that solves the offset cuboid problem in poly-logarithmic time for any query (R, k_{in}) of cuboid R and integer k_{in}, given a fixed input set P of n points. As defined in the previous section, let $\delta(p, R)$ be the real number such that the offset of R by $\delta(p, R)$ contains p on its boundary. In order to answer the offset cuboid query in a desired time, we need to select the $(k_{in} + 1)$-th largest value among $\delta(p, R)$ for all $p \in P \cap R$ in poly-logarithmic time.

A. Offset Cuboid Queries

For the purpose, we construct a data structure of points in P that enables us to count the number of points of P lying in the interior of a query cuboid in $O(\log^{d-1} n)$ time. Such a data structure can be implemented using the *d-dimensional range tree* for orthogonal range counting with $O(n \log^{d-1} n)$ preprocessing time and $O(n \log^{d-1} n)$ storage [19]. In addition, we build d balanced binary search trees $\mathcal{T}_1,...,\mathcal{T}_d$ such that $\mathcal{T}_i$ for each $i = 1,...,d$ is constructed on the x_i-coordinates of n points in P and each node of $\mathcal{T}_i$ stores the corresponding point in P. These binary search trees $\mathcal{T}_i$ can be built in $O(n \log n)$ time with $O(n)$ storage. See the book [18] for more details about the balanced binary trees. The preparation of the above data structures is done only once as a preprocessing.

Now, we describe how to handle an offset cuboid query for given (R, k_{in}). Our goal is to output the largest offset R' of R that contains at most k_{in} points of P in its interior. Suppose that we are given a value $\delta \geq 0$, we are then able to determime in $O(\log^{d-1} n)$ time whether or not the offset of R by δ contains at most k_{in} points in its interior based on our data structure for range counting. This represents our decision algorithm for a fixed $\delta \geq 0$.

Let δ^* be the minimum value such that our decision algorithm returns a positive answer, and R^* be the inward offset of R by δ^*. Thus, the cuboidal shell determined by R and R^* represents the optimal solution for a given query outer cuboid R, and its width is δ^*. To reduce the search space for δ^*, we use the following observation:

LEMMA 5. *Let R^* be defined as above for a given query (R, k_{in}). Then, at least one point of P lies on the boundary of R^*.*

Proof. Suppose to the contrary that the boundary of R^* contains no point in P. Then, by growing R^* until it hits a point P, there must be a larger offset of R that contains at most k_{in} points in its interior, which is a contradiction Q.E.D.

Since a cuboid has $2d$ facets, at least one of the $2d$ facets of R^* should contain a point in P according to Lemma 5. Consider the case where the facet of R^* orthogonal to the x_1-axis with the smaller x_1-coordinate contains a point $p^* \in P$. Consider the two facets of R orthogonal to the x_1-axis and let x^- and x^+ be their x_1-coordinates with $x^- < x^+$. It is obvious that the x_1-coordinate of p^* lies in interval $[x^-, x^0]$, where $x^0 = \frac{x^- + x^+}{2}$.

Now, we are ready to describe our algorithm to determine p^* and δ^*. It starts with two standard search queries for x^- and x^0 on the balanced binary search tree $\mathcal{T}_1$, resulting in two paths from the root to a leaf in $\mathcal{T}_1$. The two search paths share a common component starting from the root and then split at some node v of $\mathcal{T}_1$. We traverse $\mathcal{T}_1$ again from the split node v. Based on the construction, the x_1-coordinate x_v corresponding to v lies in $[x^-, x^0]$. We then apply our decision algorithm for $\delta = x_v - x^0$. If the result is positive, we proceed to the left child of v; otherwise, we proceed to the right child of v. We apply our decision algorithm for the next node repeatedly until we reach a leaf of $\mathcal{T}_1$. The leaf node corresponds to the point p^*, in this case. Since the height of $\mathcal{T}_x$ is $O(\log n)$ and our decision algorithm takes $O(\log^{d-1} n)$ time, the procedure terminates in $O(\log^d n)$ time.

The other cases, where p^* lies on one of the other $2d - 1$ facets of R^* is handled symmetrically by traversing one of the d binary search trees $\mathcal{T}_1,...,\mathcal{T}_d$.

The following summarizes the above discussion.

LEMMA 6. *Given a cuboid R and an integer k_{in}, we can compute in $O(\log^d n)$ time the largest offset of R that contains at most k_{in} points in its interior, after preprocessing of time $O(n \log^{d-1} n)$ to build our data structures.*

B. Second Algorithm

Now, we return to the original (k_{in}, k_{out})-CUBSHELL problem. In the previous section, we showed that the

(k_{in}, k_{out})-CUBSHELL problem can be reduced to $O(k_{out}^{2d-1}n)$ instances of the (k_{in}, 0)-CUBSHELL problem (Fig. 3). By Lemma 6, a single such instance can be handled in poly-logarithmic time, after $O(n \log^{d-1} n)$-time overhead for preprocessing. More precisely, from the pseudocode in Fig. 3, line 10 can be implemented by Lemma 6 to yield the following result.

THEOREM 2. *Given a set P of n points in the plane and integers* $k_{in}, k_{out} \geq 0$, *the* (k_{in}, k_{out})-CUBSHELL *problem can be solved in* $O(n \log^{d-1} n + k_{out}^{2d-1} \log^{d} n)$ *time and* $O(n \log^{d-1} n)$ *space. Therefore, the k-*CUBSHELL *problem for any* $k \geq 0$ *can be solved in* $O(n \log^{d-1} n + k^{2d} \log^{d} n)$ *time and* $O(n \log^{d-1} n)$ *space*.

Proof. The description of this algorithm is almost identical to that shown in Fig. 3, except that we build the d-dimensional range tree on P for range counting queries and the d binary search trees $\mathcal{T}_1,...,\mathcal{T}_d$ as a preprocessing time. Also, we apply Lemma 6, rather than Lemma 4, to solve each instance of the (k_{in}, 0)-CUBSHELL problem occurring in the innermost loop of the algorithm.

The preprocessing step is completed in $O(n \log^{d-1} n)$ time as discussed above. Since we solve $O(k_{out}^{2d-1})$ instances of the (k_{in}, 0)-CUBSHELL problem in the innermost loop, the total time complexity is bounded by $O(n \log^{d-1} n + k_{out}^{2d-1} \log^{d} n)$, by Lemma 6. The space used is bounded by $O(n \log^{d-1} n)$.

The k-CUBSHELL problem for $k \geq 1$ is reduced to $k + 1$ instances of the (k_{in}, k_{out})-CUBSHELL problem with $k_{in} + k_{out} = k$. Therefore, $O(n \log^{d-1} n + k^{2d} \log^{d} n)$ time is sufficient.

V. CONCLUSION REMARKS

We have presented two efficient algorithms that solve the minimum-width cuboidal shell problem with outliers. No algorithm for the problem has been published prior to this work. Given n points in $\mathbb{R}^d$ for $d \geq 3$ and the number k of outliers, our algorithms solve the k-CUBSHELL problem in $O(k^{2d}n)$ or $O(n \log^{d-1} n + k^{2d} \log^{d} n)$ time. Since the number of outliers (k) is often very small relative to the number of data points (n), our algorithms are notably efficient in most practical cases. If k is a constant, then the former takes linear time, while the latter outperforms the former when $k = \Omega(\sqrt{\log n})$ Nonetheless, the first algorithm is worth its simplicity. The second algorithm makes use of standard data structures and algorithmic techniques, which can be easily found in an open source library.

The existing algorithms for $d = 2$ by Bae [15] and Ahn et al. [16] are indeed faster than the present algorithms. An interesting technique therein is the use of a *kernel* for the problem. A kernel for our problem can be defined to be a subset K of the input point set P such that the minimum-width cuboidal shell with outliers for points K is equivalent to that for points P. Bae [15] proved the existence of a kernel of size $O(k^4)$ for the problem where $d = 2$, and showed how to find such a kernel, resulting in a faster $O(nk^2 \log n + k^4 \log^2 k)$-time algorithm. However, the existence of such a kernel for $d \geq 3$ is unknown. A natural open question is whether there exists a kernel of small size for the cuboidal shell problem with $d \geq 3$. If one could prove the existence of a kernel of small size for $d \geq 3$ and show how to compute it, then an improved and faster algorithm for the k-CUBSHELL problem would be obtained.

ACKNOWLEDGEMENTS

This work was supported by Kyonggi University Research Grant 2019.

REFERENCES

1. A. Aggarwal, H. Imai, N. Katoh, and S. Suri, "Finding k points with minimum diameter and related problems," *Journal of Algorithms*, vol. 12, no. 1, pp. 38-56, 1991.
2. M. Segal and K. Kedem, "Enclosing k points in the smallest axis parallel rectangle," *Information Processing Letters*, vol. 65, no. 2, pp. 95-99, 1998.
3. R. Atanassov, P. Bose, M. Couture, A. Maheshwari, P. Morin, M. Paquette, M. Smid, and S. Wuhrer, "Algorithms for optimal outlier removal," *Journal of Discrete Algorithms*, vol. 7, no. 2, pp. 239-248, 2009.
4. H.-K. Ahn, S. W. Bae, E. D. Demaine, M. L. Demaine, S.-S. Kim, M. Korman, I. Reinbacher, and W. Son, "Covering points by disjoint boxes with outliers," *Computational Geometry*, vol. 44, no. 3, pp. 178-190, 2011.
5. T. M. Chan, "Geometric applications of a randomized optimization technique," *Discrete & Computational Geometry*, vol. 22, no. 4, pp. 547-567, 1999.
6. J. Matousek, "On geometric optimization with few violated constraints," *Discrete & Computational Geometry*, vol. 14, pp. 365-384, 1995.
7. T. M. Chan, "Low-dimensional linear programming with violations," *SIAM Journal on Computing*, vol. 34, no. 4, pp. 879-893, 2005.
8. H. Ebara, N. Fukuyama, H. Nakano, and Y. Nakanishi, "Roundness algorithms using the Voronoi diagrams," in *Proceedings of the 1st Canadian Conference on Computational Geometry (CCCG)*, Montreal, Canada, 1989.
9. U. Roy and X. Zhang, "Establishment of a pair of concentric circles with the minimum radial separation for assessing roundness error," *Computer-Aided Design*, vol. 24, no. 3, pp. 161-168, 1992.
10. A. D. Wainstein, "A non-monotonous placement problem in the plane," in *Proceedings of the 9th All-Union Symposium Abstracts: Software Systems for Solving Optimal Planning Problems*, Minsk, BSSR, 1986, pp. 70-71.
11. P. K. Agarwal, M. Sharir, and S. Toledo, "Applications of parametric searching in geometric optimization," *Journal of*

Algorithms, vol. 17, no. 3, pp. 292-318, 1994.
12. P. K. Agarwal and M. Sharir, "Efficient randomized algorithms for some geometric optimization problems," *Discrete & Computational Geometry*, vol. 16, no. 4, pp. 317-337, 1996.
13. M. Abellanas, F. Hurtado, C. Icking, L. Ma, B. Palop, and P. A. Ramos, "Best fitting rectangles," in *Proceedings of the European Workshop on Computational Geometry (EuroCG)*, Bonn, Germany, 2003, pp. 147-150.
14. O. N. Gluchshenko, H. W. Hamacher, and A. Tamir, "An optimal O(nlogn) algorithm for finding an enclosing planar rectilinear annulus of minimum width," *Operations Research Letters*, vol. 37, no. 3, pp. 168-170, 2009.
15. S. W. Bae, "Computing a minimum-width square or rectangular annulus with outliers," *Computational Geometry*, vol. 76, pp. 33-45, 2019.
16. H.-K. Ahn, T. Ahn, S. W. Bae, J. Choi, M. Kim, E. Oh, C.-S. Shin, and S. D. Yoon, "Minimum-width annulus with outliers: circular, square, and rectangular cases," *Information Processing Letters*, vol. 145, pp. 16-23, 2019.
17. J. Mukherjee, P. R. S. Mahapatra, A. Karmakar, and S. Das, "Minimum-width rectangular annulus," *Theoretical Computer Science*, vol. 508, pp. 74-80, 2013.
18. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA: MIT Press, 2009.
19. M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*. Heidelberg, Germany: Springer, 2008.

Sang Won Bae 0000-0002-8802-4247

Sang Won Bae got his Ph.D. in 2008 at Department of Computer Science, Korea Advanced Institute of Science and Technology, Daejeon, Korea. At present, he is working as an associate professor at Division of Computer Science and Engineering, Kyonggi University, Suwon, Korea. Research interests include algorithms design and analysis in computational geometry, discrete and combinatorial geometry, graph theory, and their algorithmic applications to other disciplines.