A Comparative Study of Dual-tree Algorithms for Computing Spatial Distance Histograms

Chengcheng Mou,[†] Shaoping Chen,[†] and Yi-Cheng Tu

Abstract—The *2-body correlation function* (2-BCF) is a group of statistical measurements that found applications in many scientific domains. One type of 2-BCF named the Spatial Distance Histogram (SDH) is of vital importance in describing the physical features of natural systems. While a naïve way of computing SDH requires quadratical time, efficient algorithms based on resolving nodes in spatial trees have been developed. A key decision in the design of such algorithms is to choose a proper underlying data structure: our previous work utilizes quad-tree (oct-tree for 3-dimensional data) and in this paper we study a kd-tree-based solution. Although it is easy to see that both implementations have the same time complexity $O\left(N^{\frac{2d-1}{d}}\right)$, where d is the number of dimensions of the dataset, a thorough comparison of their *actual* running time under different scenarios is conducted. In particular, we present an analytical model to rigorously quantify the running time of dual-tree algorithms. Our analysis suggests that the kd-tree-based implementation outperforms the quad-/oct-tree solution under a wide range of data sizes and query parameters. Specifically, such performance advantage is shown as a speedup up to 1.23X over the quad-tree algorithm for 2D data, and 1.39X over the oct-tree for 3D data, respectively. Results of extensive experiments run on synthetic and real datasets confirm our findings.

Index Terms—scientific databases, query processing, spatial distance histogram, performance, quad-tree, oct-tree, kd-tree

1 Introduction

Recently, computational science fields have witnessed the momentum of data-intensive applications that severely challenge the design of *database management system* (DBMSs). Much efforts have been made in building systems and tools to meet the data management needs of such applications [1], [2], [3]. Generally, data-intensive scientific applications necessitate considerable storage space and I/O bandwidth, due to the large volume of data [4], [5], [6]. For instance, molecular simulations (MS) evaluate the movement patterns and interaction forces among molecular structures, each of which consists of millions of atoms. Other than the large volume of data, there is also the challenge of processing scientific queries that are often analytical in nature and bear high computational complexity [7], [8]. One remarkable example is the computation of *2-body correlation functions* (2-BCFs), which are statistical measurements that involve every pair of data points in the entire dataset. One type of 2-BCF called the *Spatial Distance Histogram* (SDH) is of vital importance in many computational sciences and thus the focus of this paper.

1.1 Problem Statement

The SDH problem can be formally stated as follows.

Given the coordinates of N points in a (2D or 3D) Cartesian coordinates system, draw a histogram that depicts the distribution of the point-to-point distances among the N points.

Generally, an SDH comes with a parameter l, which is the total number of buckets. Because the dataset is generated from a simulation system with a fixed dimension, the maximum distance (L_{max}) between any two points in the system is a constant. In this study, we deal with the standard SDH, whose buckets are of the same width. The width of buckets $p = L_{max}/l$, also named *histogram resolution*, is usually used as the parameter of the query. Specifically, with a given histogram resolution p, SDH asks for the number of point-to-point distances that fall into ranges $[0, p), [p, 2p), [2p, 3p), ..., [(l-1)p, lp)$, respectively. Obviously, for the same dataset, more computation is needed for an SDH with smaller p value.

1.2 Objective

In a dataset with N particles, SDH requires $O(N^2)$ computation time to carry out all point-to-point distance computations. Our previous work proposed more efficient algorithms [9]. Instead of computing every point-to-point distance, the main idea of such algorithms is to analyze the distances between two groups of points, as described in Section 3.1. These groups are represented by nodes in a space-partitioning tree structure, called *density map* (DMs), as discussed in Section 3.2. The reduction of running time is achieved by the fact that the brute-force distance computations are substituted by recursively calling the *Resolution Function* that takes two tree nodes as inputs (for which the algorithms are named *dual-tree* algorithms). The main objective of this paper is to provide analytical and empirical evaluations of different data structures for implementing the DM. So far our work only used a quad-tree (oct-tree for 3D data) for such purposes [9], and it is natural to look into other spatial data structures for the same purpose. In this paper, we study and evaluate an implementation based on

- [†]*These authors contributed equally to this work.*
- *Chengcheng Mou and Yi-Cheng Tu are with the Department of Computer Science and Engineering, University of South Florida, 4202 E. Fowler Ave., ENB 118, Tampa, FL, 33620. E-mails: chengcheng@mail.usf.edu, tuy@mail.usf.edu*
- *Shaoping Chen is with the Department of Mathematics, Wuhan University of Technology, 122 Luosi Road, Wuhan, Hubei 430070, P. R. China. Email: chensp@whut.edu.cn*

a region kd-tree whose details will be introduced in Section 3.2. Although algorithms based on both trees have the same time complexity $O\left(N^{\frac{2d-1}{d}}\right)$ where d is number of dimensions of dataset [10], a comparison of their actual execution time under different scenarios is thoroughly studied. Our main technique is to transform the analysis of the number of particle counts into a problem of quantifying the area of interesting geometric regions. Our analysis leads to rigorous results for differentiating the running time of these two dual-tree algorithms (quad-tree-based and kd-tree-based) under different cases. Our analysis suggests that the use of kd-tree brings significant performance advantage to the dual-tree algorithm under a wide range of data sizes and query parameters. In particular, the kd-tree yields a speedup up to 1.23X over the quad-tree in processing 2D data, and a speedup up to 1.39X over the oct-tree in processing 3D data. Results of extensive experiments confirm such findings.

1.3 Paper Organization

The remainder of this paper is organized as such: In Section 2, we review the works related to SDH problem and discuss the contributions of this work; In Section 3, we sketch the dual-tree algorithm; We introduce our modeling approach and present the main analytical results in Section 4; Based on the main results, we study and compare the performance of the two dual-tree algorithms in Section 5; We present extended analytical results about 3D data in Section 6; We report experimental results in Section 7, and conclude this paper in Section 8.

2 Related work and our contributions

2.1 Motivating Applications of SDH

The SDH is a fundamental tool in understanding the physical features of systems consisting of many particles. For that reason, SDH is routinely computed in analyzing data generated from a very important type of computer simulation - particle simulations. Such simulations treat individual components (e.g., atoms, stars, etc.) of large systems (e.g., molecules, galaxies, etc.) as classical entities that interact with each other following Newton's Law. These techniques are applicable in modeling of complex chemical and biological system that are beyond the scope of theoretical models, under such scenarios the simulation is called molecular simulations (MS). MS has been widely utilized in material sciences [11], astrophysics [12], biomedical sciences, and biophysics [13]. In a molecular system, the SDH is the discrete form of a continuous statistical distribution named radial distribution function (RDF), which describes how the atom density varies as a function of distance from a referenced point. RDF is an essential component in computing a series of critical quantities describing a system, such as internal pressure and energy [12], [14], [15].

Computation of SDH also finds its application in other domains. In computer vision and pattern recognition, the concept of *Color Correlogram*, which is a table indexed by color pairs, where a k-entry for $< i, j >$ specifies the probability of a pixel of color j at a distance k from a pixel of color i in the image, has been proposed. It is regarded as a robust feature for effective scene identification under changes in viewing angle, background scene, partial occlusion, and camera zoom [16], [17]. A single image generated from modern camera might contain millions of pixels. Therefore, it takes considerable amount of time to compute the color correlogram of these images.

In the data mining field, a feature vector represents an object. The multi-dimensional feature vector could be reduced to low-dimensional feature vector by using linear reduction techniques, such as Principal Components Analysis (PCA), Karhunen-Love Transform (KLT), the Discrete Fourier (DFT), Cosine Transform (DCT), etc. Then SDH of low-dimensional feature vector in Cartesian Coordinate System could therefore statistically conduct similarity search or classification of the specific objects [18], [19].

The significance of this work is not limited to SDH or the 2-BCF themselves: similar techniques presented in this paper can provide insights in computing the more general n-body correlation function (n-BCF) where $n > 2$ [20]. The n-BCFs are of interest in many forms: n-point function, n-tuple problem, nearest-neighbor classification, nonparametric outlier detection/denoising, kernel density/classify/regression [21] are examples of statistical measurements related to n-BCF, and their applications are found in various scientific fields [22], [23], [24].

2.2 Algorithms for Efficient SDH Computation

In our previous work [9], we proposed the use of quad/oct-tree to split the domain space into equally-sized cells for SDH processing. In [25], we presented a comprehensive analysis of quad/oct-tree-based dual-tree algorithm based on a geometry modeling approach; based on results of our rigorous mathematical proof, we showed the theoretical running time of our algorithms: $O\left(N^{\frac{2d-1}{d}}\right)$ where d is number of dimension of dataset. A solution for similar problems was proposed in [21], in which a data-driven spatial tree is used: each level of the tree is generated by partitioning the region into two subregions with equal number of data points along one dimension. Our region kd-tree method, on the other hand, partitions a region by cutting at the middle point of the one-dimensional segment represented by a node. In other words, under an uniform spatial distribution of data, their proposed data structure is equivalent to the region kd-tree we study in this paper. Our main contribution, however, lies in the quantitative analysis of the performance of the kd-tree-based solution in comparison with the original quad/oct-tree approach. In [21], a conjecture is presented with an asymptotical analysis of tree performance but no analytical details were shown. To the best of our knowledge, there is no rigorous analysis on performance of dual-tree problem by using kd-tree. Our work reported here takes advantage of the geometric modeling method we used to analyze the quad-tree approach as shown in [25]. On top of that, we develop new models to compare the two data structures of interest in this paper. Our recent work on this topic [26], [27], [28] focuses on approximate SDH processing and parallel computing. Such work, again, only considers the quad/oct-tree as the underlying data structure thus has little overlap with this paper.

A shorter version of this paper can be found in [29], in which we sketched our analytical model and presented the main results on the comparative study between two data structures under 2D data. In this paper, we extend the analysis to 3D data and comparison between kd-tree and oct-tree used in our previous work. The 3D analysis, although following the geometric modeling strategy, is significantly more complex and challenging. We also evaluate the 3D analytical results with extensive experiments. Furthermore, we present more complete proof of major theorems in the 2D analysis that was not published in [29] due to page limits.

3 Preliminaries

In this section, we elaborate on the dual-tree algorithm for computing SDH, in order to pave the way for future discussions related to the performance evaluation of the algorithm. In Table 1, we list the notations that are used throughout this paper. Note that symbols defined and referenced in a local context are not listed here.

TABLE 1: Symbols and notations

Symbol	Definition
p	width of histogram buckets
l	total number of histogram buckets
h	the histogram array with indexed elements $h_i (0 < i \leq l)$
N	total number of particles in data
i	an index symbol for any series
DM_i	the i-th level of density map
d	number of dimensions of data
δ	diagonal length of the cells

3.1 Overview of the Dual-tree Algorithm

The main idea of the dual-tree algorithm is to work on the distances between two **clusters** of points instead of those between two **individual** points to save time. From now on, we use 2D data to elaborate on technical details till we explicitly extend our discussions to 3D data in Section 6. The dual-tree algorithm starts by building the tree structures, and cache the total number of data points in each node. An entire level of the tree with such counts is called a *density map* (DM, see Fig. 1 for examples). The main body of the algorithm is a primitive named *ResolveTwoTrees* (referred to as *resolution function* hereafter) which takes a pair of tree nodes as input. Given a pair of nodes on the DM, if the both *minimum* and *maximum* distances between these two nodes fall completely into a histogram bucket, we say that this pair is *resolvable*. An important observation here is: for a pair of resolvable nodes, we only need to add the total number of distances between them to the corresponding bucket in the SDH. This is also the main reason why such algorithm is more efficient than the brute force approach. If the pair of nodes is unresolvable, the resolution function recursively visits next level of the tree to resolve all pairs of child nodes (cells, since they are the same, we may alternatively use them hereafter), so on and so forth. If a pair of nodes is still unresolvable at the leaf level, we have to compute all the point-to-point distances between the data points across that pair of nodes.

The pseudocode that summarizes the technical details of the algorithm can be found in Algorithm 1. The core process of the algorithm is the procedure *ResolveTwoTrees*, which tries to resolve two cells m_1 and m_2 on the same DM. In order to check whether m_1 and m_2 are resolvable, we firstly compute the *minimum* and *maximum* distances between any points from m_1 and m_2. Note this process only requires constant running time. When both minimum and maximum distances between the two cells fall into a same histogram bucket i, the value (i.e., distance counts) in bucket i will increment by $n_1 n_2$, where n_1 and n_2 are the number of points in the spatial region represented by m_1 and m_2, respectively. If m_1 and m_2 are not resolvable on density map DM_i, we move to next level of Density Map DM_{i+1}, and recursively call the same function to check each of four children in m_1 to each of four children in m_2. However, if two nodes are still not resolvable on the last level DM of the tree, we have to calculate the distances between all pairs of points from the two cells. In addition, if we have $n_1 = 0$ or $n_2 = 0$ (i.e., empty nodes), the procedure directly exits.

Algorithm 1: The dual-tree algorithm for SDH

```
Data: all data points, DM, and bucket width $p$;
Result: an array of distance counts $h$
1  initialize all elements in $h$ to 0;
2  $DM_0 \leftarrow$ first DM with cell diagonal length $\delta \leq p$;
3  for every cell in $DM_0$ do
4      $n \leftarrow$ number of particles in the cell;
5      $h_1 = h_1 + \frac{1}{2}n(n-1)$;
6  end
7  for every pair of cells $m_i$ and $m_j$ in $DM_0$ do
8      ResolveTwoTrees $(m_i, m_j)$;
9  end
10 return $h$

11 ResolveTwoTrees $(m_1, m_2)$
12 $n_1 \leftarrow$ number of points in $m_1$
13 $n_2 \leftarrow$ number of points in $m_2$
14 if $n_1 = 0$ or $n_2 = 0$ then
15     return
16 end
17 if $m_1$ and $m_2$ are resolvable into a bucket $i$ then
18     $h_i \leftarrow h_i + n_1 n_2$;
19     return
20 end
21 if $m_1$ and $m_2$ are on the last density map then
22     for each particle A in $m_1$ do
23         for each particle B in $m_2$ do
24             $f \leftarrow$ distance between $A$ and $B$;
25             $i \leftarrow$ the bucket $f$ falls into;
26             $h_i \leftarrow h_i + 1$;
27         end
28     end
29 else
30     for each child node $m_1'$ of $m_1$ do
31         for each child node $m_2'$ of $m_2$ do
32             ResolveTwoTrees $(m_1', m_2')$
33         end
34     end
35 end
```

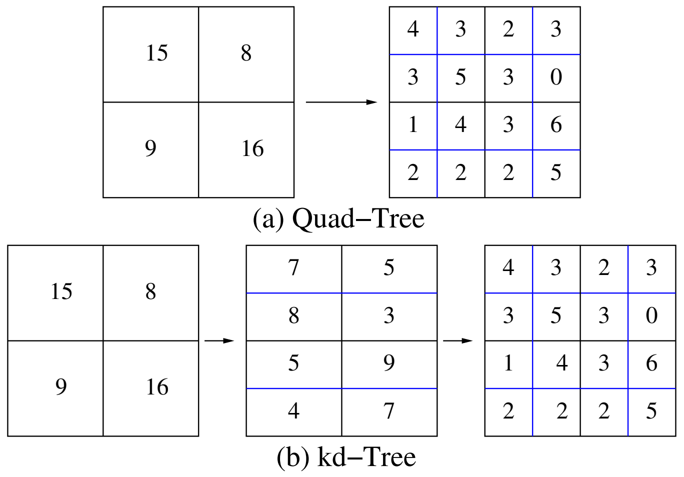


Fig. 1: A partial DM implemented by quad-tree and kd-tree. Each cell is marked by the total number of data points in it

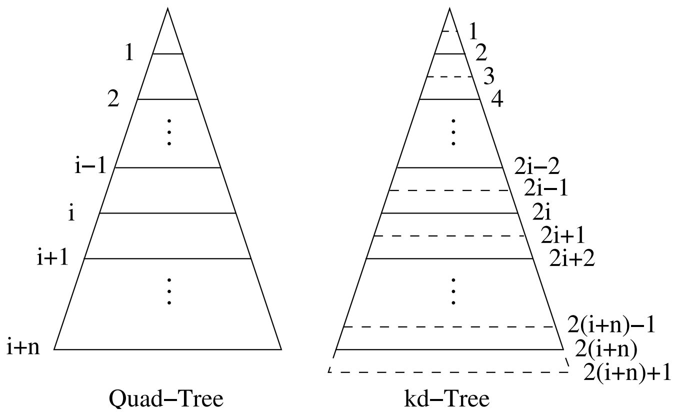


Fig. 2: Different levels on quad-tree and kd-tree. Dash line represents the intermediate level that only exists in kd-tree, and a solid line corresponds to a level that exists in both trees

3.2 Implementations Based on Different Trees

To implement Algorithm 1, one decision to make is what type of data structure we use to build the DM. Our previous work [9] uses a quad-tree: when the space is partitioned to lower-level nodes, the tree simultaneously bisects both x- and y-dimensions at each partition, generating four children for each internal node. In this paper, we consider the use of kd-tree, which alternatively bisects its x- or y-dimension at each partition, leading to a tree degree of two (Fig. 1). In both trees, the region containing all points in the dataset represents the root node. Given the same dataset, the kd-tree introduces an extra level of nodes in between any two neighboring levels of the quad-tree, as shown in Fig. 2. The immediate question is whether the kd-tree-based algorithm has better performance, and this paper presents an answer to this question via a rigorous analytical approach. A special note here is that both trees define a node by a prefixed region instead of being driven by data distribution. The main reason for this is: the resolving of two trees is a process that is only related to the dimensions of the two trees, the data in the trees are irrelevant.

Before we start performance analysis, it is essential to present two critical features of the dual-tree algorithm regarding the size of the tree structures. First, the height of the tree is determined by the data size N. Specifically, we keep partitioning the tree until the average number of data points in each node is smaller than a threshold b. Thus, the height of the tree can be expressed as

$$H = \left\lfloor \log_k \frac{N}{b} \right\rfloor + 1 \tag{1}$$

where k is the degree of the tree (i.e., 4 for quad-tree and 2 for kd-tree). The value b is set based on the following reasoning: the cost of computing all the point-to-point distances is b^2, and the cost of resolving two cells is a fixed value C; if we are to further partition the nodes into a new level, there will be k^2 resolution calls, therefore it makes sense to create this new level only if we have $b^2 > k^2C$, or $b > k\sqrt{C}$. Otherwise, we should not further partition the nodes and make the current level the leaf level. The important observation here is: given the same N, as C does not change, the kd-tree can build an extra level on the bottom as compared to the quad-tree.

Another important feature of the algorithm is the level of the tree where the algorithm starts calling the resolution function. Specifically, the algorithm starts at a tree level (i.e., a DM) where the size of the cells/nodes satisfies

$$a \leq \frac{p}{\sqrt{d}} \quad \text{i.e.} \quad \delta \leq p \tag{2}$$

where a is the side length (δ is the diagonal length) of the cells, p is the histogram bucket width, and d is the number of dimensions in the data. This is because, if the above is not true, none of the node pairs will resolve. In other words, the bucket width p determines the starting DM. Consequently, the algorithm may start at the identical or different levels on the quad-tree and kd-tree, depending on the value of p. The extra levels that only exist in the kd-tree give chances for the algorithm to start earlier (at such extra levels) in the tree (Fig. 2).

As we shall see later (Section 5), the above two features define four scenarios to consider in comparing the performance of the kd-tree-based algorithm to that of the quad-tree-based one. In these four cases, the relative performance of the algorithms are different. We will discuss the scenarios in a 3D system in Section 6.

4 Main analytical results

We first present our analysis on how fast the resolution function resolves the points when it recursively visits the tree in a depth-first manner. This turns out to be a key step in modeling the relative performance of the two algorithms.

4.1 The Geometric Modeling Approach

To quantify the number of points resolved, we transform the problem into a geometric modeling problem. In particular, we develop a model to quantify how the area of the region that can be resolved increases as more DMs (i.e., tree levels) are visited. Consequently, any points that fall into such regions are resolved.[1]

1. Note that such transformation is based on an implicit assumption that data is uniformly distributed in the simulation space, because we adopted space-oriented (bisecting each dimension) method. We will remove this assumption in our analysis as shown in Section 5.1.

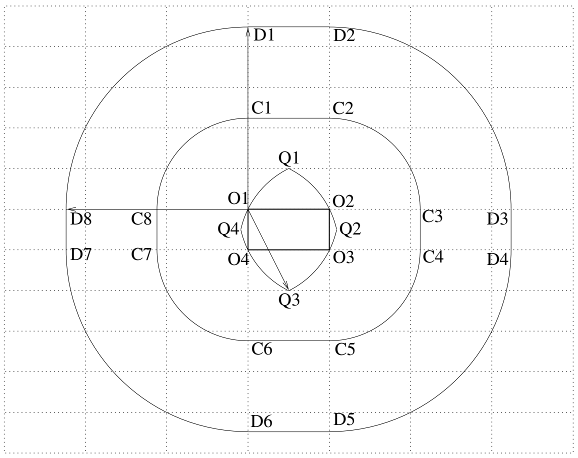


Fig. 3: Theoretical boundaries of bucket 1 and bucket 2 regions for cell A, with the bucket width $p = \sqrt{2}\delta$

Given any cell A on the DM where the algorithm starts (Fig. 3), we first define a theoretical region that contains all particles that can possibly resolve into the i-th bucket with any particle in A. We name this region as *bucket i region* for cell A, and denote it as A_i. Note that A can be either a square or a rectangle in the kd-tree implementation. In all illustrations of this paper, we only draw rectangular cells but our analysis will cover both cases. Going back to Fig. 3, cell A is marked with its four corner points O_1, O_2, O_3, and O_4, A_1 is therefore bounded by 4 arcs and 4 line segments connected by points C_1 through C_8. The arcs are of the same radius p. Here we consider the special case of Equation (2): the diagonal length of cell A is set to be $\delta = \frac{p}{\sqrt{2}}$. However, as we shall see later, the case of $\delta < \frac{p}{\sqrt{2}}$ will not change our analytical results.

The cells that are actually resolvable into bucket i with any subcells in A also from a region. We named such region as *coverable region* and denote it as A'_i. Since a coverable region contains rectangles or squares, its boundary (solid blue line in Fig. 4) shows a zigzag pattern. An essential part of our analysis is to study the area of coverable regions over all buckets and how the density map resolution affects it. We define the ratio of $\sum_i A'_i$ to $\sum_i A_i$ as the *covering factor*, which is a critical quantity to measure how much area are "covered" by the resolvable cells. Note that the boundary of A'_i approaches that of A_i (solid black line in Fig. 4) when the dual-tree algorithm visits more levels of the tree. As a result, the covering factor increases. Of special interest to our analysis is the *non-covering factor* which indicates the percentage of area that is not resolvable, i.e.,

$$\textit{non-covering factor} = 1 - \textit{covering factor} \tag{3}$$

Our previous work [25] has studied the resolution ratio of dual-tree algorithm running on top of the quad-tree. A very important feature of the non-covering factor in the quad-tree can be summarized in the following theorem.

Theorem 1. *Let DM_i be the first density map where the quad-tree algorithm starts running, and we define the non-covering factor α_m as a function of the levels of density maps visited m. In other words, α_m is the percentage of cell pairs that are not resolved upon visiting DM_{i+m}. We have*

$$\lim_{p\to 0} \frac{\alpha_{m+1}}{\alpha_m} = \frac{1}{2}$$

Basically, Theorem 1 says that half of the node pairs are resolved when one more level of the tree is visited. From this theorem we can easily derive a recurrence function that leads to the time complexity of the quad-tree-based algorithm dropping to $O\big(N^{\frac{2d-1}{d}}\big)$, where d is number of dimensions of dataset [10]. This theorem, by focusing on the non-covering factors on two consecutive levels, essentially shows how fast the data points could be resolved while the dual-tree algorithm visits the quad-tree structure.

For the same dataset, the kd-tree has extra levels that are not seen in the quad-tree, the data points could be resolved earlier in the kd-tree by the resolution function. Intuitively, if more data points are resolved by the resolution function call, fewer of them are left for distance computation. That is the *benefit* of calling the resolution function earlier (among the intermediate tree nodes). On the other hand, the time we spend on calling the resolution function on such levels is a pure *cost*. Just by looking, it is not clear how much net performance gain such "early resolution" in the kd-tree can generate. Therefore, it is essential to study the same quantity α_{m+1}/α_m in the kd-tree.

4.2 Non-Covering Factor Ratios in kd-tree

Rather than square cells in the quad-tree, the kd-tree introduces rectangular cells on the intermediate levels, the algorithm therefore alternatively visits the square and rectangular cells, resulting in more complicated scenarios in studying the resolution ratios on the kd-tree. Our main results on kd-tree can be seen in the following theorem.

Theorem 2. *Let DM_i be the first density map where the dual-tree algorithm starts running on a kd-tree, and α_m be the non-covering factor upon visiting the density map that lies m levels below DM_i, we have*

$$\lim_{p\to 0} \frac{\alpha_{m+1}}{\alpha_m} = \frac{3}{4} \tag{4}$$

when $i + m$ is even, and

$$\lim_{p\to 0} \frac{\alpha_{m+1}}{\alpha_m} = \frac{2}{3} \tag{5}$$

when $i + m$ is odd.

In the remainder of this section, we present a proof of Theorem 2. However, readers can jump to Section 5, in which we show how Theorem 2 leads to effective analysis of algorithm performance.

4.2.1 Bucket Region

As shown in Fig. 3, the bucket 1 region for cell A is connected by C_1 through C_8; C_1C_2, C_3C_4, C_5C_6, and C_7C_8 are all line segments; C_2C_3, C_4C_5, C_6C_7, and C_8C_1 are all 90-degree arcs with radius p and centered at O_2, O_3, O_4, and O_1, respectively. Apparently, the area of this region is $\pi p^2 + 2p\delta + p\delta + \frac{\delta^2}{2}$. The bucket 2 region of A is similar to bucket 1 region but the radii of the four arcs are $2p -$

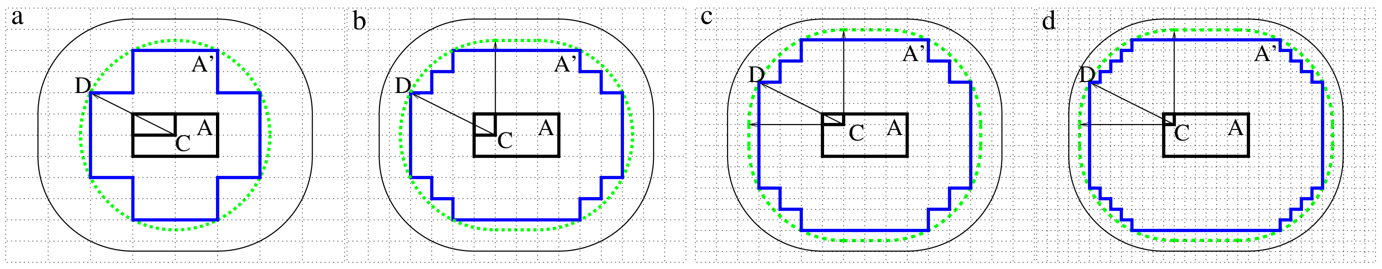


Fig. 4: Actual (*solid blue line*) and approximated (*dashed green line*) coverable region for bucket 1 under: a. $m = 2$, b. $m = 3$, c. $m = 4$, and d. $m = 5$. Outer solid black line represent the theoretical bucket 1 region. All arrowed line segments are drawn from the centers to the corresponding arcs with radius p

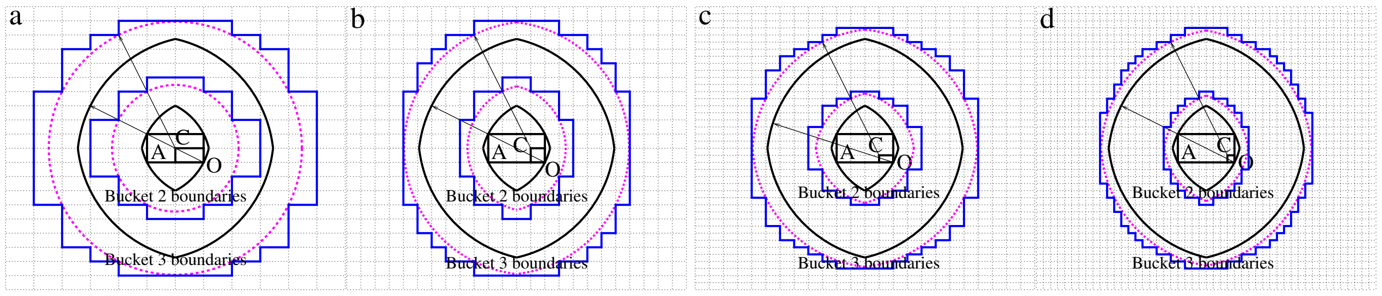


Fig. 5: Inner boundaries of the coverable regions of buckets 2 and 3 under a. $m = 2$, b. $m = 3$, c. $m = 4$, and d. $m = 5$. All arrowed line segments are of length $2p$

this region is connected by D_1 all the way around to D_8. However, if the points are too close to A, they will only be resolved into bucket 1, because their distances to any points in A will always be shorter than p. These points formed a region, which is connected by four arcs Q_1Q_2, Q_2Q_3, Q_3Q_4, and Q_4Q_1 with radius p and centered at opposite corners of A. The bucket 2 region should not take count of such inner region. This football-shaped inner region $Q_1Q_2Q_3Q_4$ has fourfold of the area of region $\widehat{Q_4Q_1}D$ (Fig. 6). To get area of $\widehat{Q_4Q_1}D$, we first calculate the area of sector $\widehat{Q_4Q_1}O_3$

$$\begin{aligned} S_{\widehat{Q_4Q_1}O_3} &= \frac{1}{2}p^2 \cdot \angle Q_4O_3Q_1 \\ &= \frac{1}{2}p^2 \cdot \left(\frac{\pi}{2} - \angle Q_4O_3F - \angle Q_1O_3A\right) \\ &= \frac{1}{2}p^2 \cdot \left(\frac{\pi}{2} - \arcsin\frac{\delta}{4p} - \arcsin\frac{\delta}{2p}\right) \end{aligned} \tag{6}$$

We then deduct the area of region $\triangle Q_4O_3B$ and $\triangle Q_1O_3C$.

$$\begin{cases} S_{\triangle Q_4O_3B} = \dfrac{\delta}{8}\sqrt{p^2 - \left(\frac{\delta}{4}\right)^2} \\ S_{\triangle Q_1O_3C} = \dfrac{\delta}{4}\sqrt{p^2 - \left(\frac{\delta}{2}\right)^2} \end{cases} \tag{7}$$

Note that, by doing that, we subtract the quadrilateral twice, and only once for each of two triangles. Thus, we have to put them back by adding the area of rectangle O_3BDC only once, then we get the area of $Q_1Q_2Q_3Q_4$, is given by Eq. (45) in Appendix A.

The shape of bucket i ($i > 2$) regions is the same as bucket 2 region except the radii of the arcs become ip. Recall that the algorithm starts from a DM where $p \geq diagonal$. For convenience of presentation, we set $p = diagonal$, i.e.,

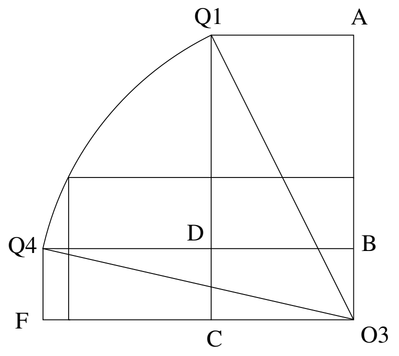


Fig. 6: A part of the football-shaped region shown in Fig. 3

$p = \frac{\sqrt{5}\delta}{2}$. As we will see later, $p > diagonal$ will not affect our analysis. We therefore have the general formula $g(i)$, is given by Eq. (46) in Appendix A, to measure the area of bucket i region.

4.2.2 Coverable Regions

Similar to bucket region, the coverable region consists of an outer region and an inner region.

4.2.2.1 The First Bucket: First, let us focus on bucket 1. In Fig. 4, we illustrate the coverable regions of four different density maps with m value ranging from 2 to 5. The solid blue line with zigzagged pattern indicates the coverable region of cell A, denotes as A'. This region contains all the cells that can be resolved into bucket 1 with any subcell in A. A key technique here is to use a smooth boundary (shown as dashed green line) to approximate the area of A'. As m increases, the boundaries of A' approach that of A. The covering factor of bucket 1 with cell A is then calculated as the ratio of the area of A' to that of A. The area of A' is given by Eq. (47) in Appendix A.

4.2.2.2 The Second Bucket and Beyond: First, we have to compute the area of the region A'_i by only considering the outer boundaries. This is the same as we did in Section 4.2.2.1 except the radii of arcs are ip. Such area for bucket A'_i, $S_{out}(i)$, is given by Eq. (48) in Appendix A.

Second, we have to consider the inner boundaries of the coverable region. Fig. 7 shows an example with $m = 1$ for buckets 2 and 3. Clearly, any cell that crossed by a segment of the theoretical inner boundary, as shown as thick solid line, will not be able to resolve into bucket i, because they are only resolvable to bucket $(i-1)$. In addition, there are more cells that are not resolvable to either bucket i or $(i-1)$. Again, we define a smooth boundary (dashed line in Fig. 7) to approximately separate the resolvable and non-resolvable regions. Such boundaries are drawn as follow: for each quadrant of cell A, we draw an arc (dashed line) with radius $(i-1)p$ and centered at the corner of the subcell of A. Consequently, any cell that crossed by this arc cannot resolve into bucket i, because they are too close to A. Such boundary also approximates the real inner boundaries (with a zigzagged pattern), and the area of region defined by such approximated boundaries is

$$\pi(ip)^2 + \delta ip - \pi[(i-1)p]^2 - \delta(i-1)p \tag{8}$$

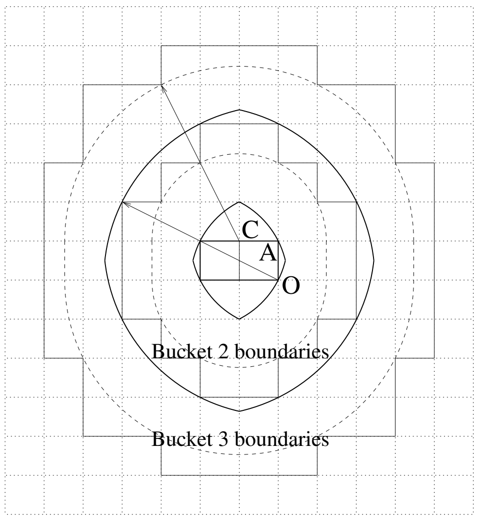


Fig. 7: Inner boundaries of the coverable region with $m = 1$

Fig. 5 illustrates more cases with m values from 2 to 5. For the cases of $m \geq 2$, we can use the same method as case of $m = 1$ to generate the real inner boundaries and approximated inner boundaries. Again, as m increases, point C approaches point O, and the approximated inner boundaries approach the theoretical inner boundaries. To compute the area of the regions formed by the approximated inner boundaries, we first need to derive angle $\angle DCB$ that encloses the shaded area shown in Fig. 8.

$$\angle DCB = \frac{\pi}{2} - \angle JCD - \angle KCB \tag{9}$$

When m is odd, the subcell is a square and we have $DJ = BK$. When m is even, the subcell is a rectangle and we have

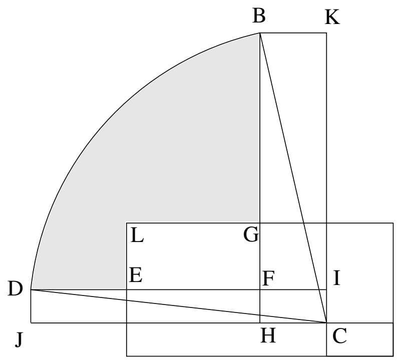


Fig. 8: The region bounded by four arcs in Fig. 7

$DJ = BK/2$. Consequently, we have two cases to calculate $\angle DCB$ when m increases.

$$\begin{cases} \beta = \frac{\pi}{2} - \arcsin\frac{\theta_m \frac{\delta}{2}}{p} - \arcsin\frac{\theta_m \delta}{p}, & m \text{ is even} \\ \beta = \frac{\pi}{2} - \arcsin\frac{\theta_{m-1} \frac{\delta}{2}}{p} - \arcsin\frac{\theta_{m+1} \delta}{p}, & m \text{ is odd} \end{cases} \tag{10}$$

where θ_m is a function of m:

$$\theta_m = \frac{1}{2} - \frac{1}{2^{m/2}}$$

With that, we can easily get the area of the Sector $\widehat{BDC}$

$$S_{\widehat{BDC}} = \frac{\angle DCB}{2\pi} \cdot \pi p^2 = \frac{\beta p^2}{2} \tag{11}$$

The area of the polygon $BFDC$ is

$$S_{BFDC} = S_{\triangle BHC} + S_{\triangle DIC} - S_{IFHC} \tag{12}$$

where $S_{\triangle BHC}$, $S_{\triangle DIC}$, and S_{IFHC} are defined as Eq. (13), Eq. (14), and Eq. (15), respectively.

$$S_{\triangle BHC} = \begin{cases} \sqrt{p^2 - (\theta_m \delta)^2} \cdot \theta_m \delta \cdot \frac{1}{2}, & m \text{ is even} \\ \sqrt{p^2 - (\theta_{m+1} \delta)^2} \cdot \theta_{m+1} \delta \cdot \frac{1}{2}, & m \text{ is odd} \end{cases} \tag{13}$$

$$S_{\triangle DIC} = \sqrt{p^2 - (\theta_m \delta)^2} \cdot \theta_m \delta \cdot \frac{1}{2} \tag{14}$$

$$S_{IFHC} = \begin{cases} (\theta_m)^2 \cdot \frac{\delta^2}{2}, & m \text{ is even} \\ \theta_{m-1} \cdot \theta_{m+1} \cdot \frac{\delta^2}{2}, & m \text{ is odd} \end{cases} \tag{15}$$

In addition, the area of the square $LEFG$ is

$$S_{LEFG} = \frac{\delta^2}{8} \tag{16}$$

Therefore, with the above four equations, we obtain the area of region bounded by four arcs (shaded region in Fig. 8) as

$$S_{shade} = (S_{sector} - S_{\triangle DIC} - S_{\triangle BHC} + S_{IFHC} - S_{LEFG})$$

For the i-th bucket, we can get the general equation to calculate S_{shade}, is given by Eq. (49) in Appendix A.

We denote the area of the coverable region A' for bucket i under different m values as $f(i, m)$

$$f(i, m) = S'_A = S_{out}(i) - 4 \cdot S_{shade}(i-1) - S_A \tag{17}$$

TABLE 2: Values of $\alpha(m+1)/\alpha(m)$ derived from fully expanded Eq. (18) as computed by MATLAB (Version 8.4). Precision is up to the 5th digit after decimal point

Density map levels	Total Number of histogram buckets									
	2	4	8	16	32	64	128	256	512	1024
$m = 1$	0.74197	0.64118	0.61973	0.61462	0.61336	0.61305	0.61297	0.61295	0.61295	0.61295
$m = 2$	0.67732	0.6691	0.66721	0.66679	0.66669	0.66667	0.66667	0.66667	0.66667	0.66667
$m = 3$	0.74807	0.74909	0.74968	0.7499	0.74997	0.74999	0.75	0.75	0.75	0.75
$m = 4$	0.67521	0.6688	0.66719	0.66679	0.6667	0.66667	0.66667	0.66667	0.66667	0.66667
$m = 5$	0.74448	0.74809	0.74941	0.74983	0.74995	0.74999	0.75	0.75	0.75	0.75
$m = 6$	0.67473	0.66891	0.66726	0.66682	0.66671	0.66668	0.66667	0.66667	0.66667	0.66667
$m = 7$	0.74276	0.74762	0.74929	0.7498	0.74994	0.74998	0.75	0.75	0.75	0.75
$m = 8$	0.67464	0.66903	0.66732	0.66685	0.66672	0.66668	0.66667	0.66667	0.66667	0.66667
$m = 9$	0.74193	0.74739	0.74923	0.74978	0.74994	0.74998	0.75	0.75	0.75	0.75
$m = 10$	0.67464	0.6691	0.66736	0.66686	0.66672	0.66668	0.66667	0.66667	0.66667	0.66667
$m = 11$	0.74151	0.74728	0.7492	0.74977	0.74994	0.74998	0.75	0.75	0.75	0.75
$m = 12$	0.67465	0.66915	0.66738	0.66687	0.66672	0.66668	0.66667	0.66667	0.66667	0.66667

The fully expanded formula for $f(i,m)$ can be found in Eq. (50) of Appendix A.

We use the non-covering factor $\alpha(m)$ to study the percentage of unresolvable pairs of cell at each level.

$$\alpha(m) = 1 - c(m) = \frac{\sum_{i=1}^{l}[g(i) - f(i,m)]}{\sum_{i=1}^{l} g(i)} \tag{18}$$

To prove Theorem 2, we start by

$$\frac{\alpha(m+1)}{\alpha(m)} = \frac{\sum_{i=1}^{l} g(i) - \sum_{i=1}^{l} f(i,m+1)}{\sum_{i=1}^{l} g(i) - \sum_{i=1}^{l} f(i,m)} \tag{19}$$

Note that functions $g(i)$ and $f(i,m)$ are given by Eq. (46) and Eq. (17) already. By plugging those into Eq. (19), we can prove that when m is even, $\alpha(m+1)/\alpha(m)$ converges to $2/3$. Such proof can be found in Appendix B.

Now let us look at $\alpha(m+2)/\alpha(m+1)$. The m-th and $(m+2)$-th levels in the kd-tree correspond to two consecutive levels in the quad-tree. By Theorem 1, we have $\alpha(m+2)/\alpha(m)$ converges to $1/2$. Since we have already shown $\alpha(m+1)/\alpha(m)$ converges to $2/3$, we can easily get

$$\lim_{p\to 0} \frac{\alpha(m+2)}{\alpha(m+1)} = \frac{3}{4} \tag{20}$$

The above concludes the proof of Theorem 2.

Numerical results (Table 2) generated from computing expanded Eq. (18) show that non-covering factor ratios quickly converge to 2/3 and 3/4, even under large p values (corresponding to small total number of buckets). The only exception is the case of $m = 1$. The reason is: when we visit a high level of the tree, the coarse grid causes a relatively big gap between the approximated boundaries (zigzagged pattern) and real boundaries (Fig. 4a and Fig. 5a). When we move to lower levels, the approximated boundary is a better estimation of the real boundaries (Fig. 4d and Fig. 5d), and this leads to smaller modeling errors. As Table 2 shows, even when $m = 2$, the non-covering factor ratios converge perfectly. Note this discussion is not focused on the value of m, it is only a matter of the actual level of tree m corresponds to. Such a fact does not diminish the value of Theorem 2 because: (1) the case of $p \to 0$ implies the visited tree level is low even when $m = 1$ therefore the theorem covers such cases; (2) even if the algorithm starts on a high level with some modeling errors, the time spent on high levels is negligible therefore it does not impose significant effects on performance analysis (see Section 5).

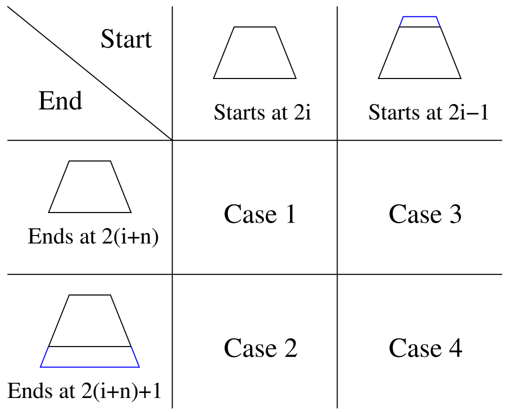


Fig. 9: Four cases in performance comparison listed from the perspective of the kd-tree-based algorithm. Note that level $2i$ corresponds to level i in the quad-tree according to Fig. 2, and a blue line represents a level that only exists in the kd-tree

5 Performance Comparison of Two Trees

Theorem 1 states that half of the node pairs are resolved when one more level of the quad-tree is visited. Theorem 2 states that a quarter of the node pairs will be resolved when the algorithm works on an even level (which has square cells and is also in the corresponding quad-tree), and a third will be resolved on the extra levels (with rectangular cell) that only show up in the kd-tree. From these two theorems we can easily derive a recurrence function that leads to the time complexity of the algorithm (see Section 6.1 in [10] for details). Although the time complexity of the algorithm is the same under both trees, it is not clear how the actual running time is affected by using a kd-tree. Intuitively, the appearance of the extra levels provides opportunities to resolve nodes earlier such that fewer node pairs are to be resolved in the following levels. On the other hand, there is extra cost to resolve pairs of nodes in such extra levels. Only when such cost is overshadowed by the saved time can we see a performance advantage from the kd-tree. Fortunately, with Theorem 2, we are able to quantitatively compare the actual running time of both algorithms under different cases (Fig. 9). Note that, in Algorithm 1, the time is only spent in two types of operation: Type I – resolution function calls; and Type II – computation of distances between data points in the unresolved leaf nodes.

5.1 Case 1

In this case, the algorithm ends at identical levels on both trees, they have the same number of unresolvable pairs of nodes at leaf level and thus the number of point-to-point distances to be computed. Therefore, we only need to compare the number of resolutions called by the algorithm.

In the quad-tree, if a pair of nodes is unresolvable at the current level, it will generate 16 pairs of nodes at the child level. In other words, for all the node pairs at the starting level, the algorithm leaves $16\alpha_0 I$ pairs unresolved, where α_0 is the non-covering factor, and I is the total number of node pairs at the starting level, respectively. At the next level, it leaves $16^2\alpha_0\alpha_1 I$ pairs unresolved. Thus, the total number of calls to the resolution function on quad-tree is

$$R = I(1 + 16\alpha_0 + 16^2\alpha_0\alpha_1 + \cdots + 16^n\alpha_0\alpha_1\cdots\alpha_{n-1}) \quad (21)$$

Based on Theorem 1, we have

$$R = I\left[1 + 16\alpha_0 + 16^2\alpha_0\left(\frac{1}{2}\right) + \cdots + 16^n\alpha_0\left(\frac{1}{2}\right)^{n-1}\right] \quad (22)$$

In the kd-tree, if a pair of nodes cannot be resolved at current level, it will generate 4 pairs of nodes at its child level. Similarly, we have total number of calls to the resolution function in the kd-tree as

$$R' = I(1 + 4\beta_0 + 4^2\beta_0\beta_1 + \cdots + 4^n\beta_0\beta_1\cdots\beta_{2n-1}) \quad (23)$$

where β_i is the non-covering factor, and I is the total number of node pairs at the starting level. With Theorem 2, we have

$$\begin{aligned}R' =I\Bigg[&1 + 4\beta_0 + 4^2\beta_0\left(\frac{3}{4}\right) + 4^3\beta_0\left(\frac{1}{2}\right)\\ &+ \cdots + 4^{2n-1}\beta_0\left(\frac{1}{2}\right)^{n-1} + 4^{2n}\beta_0\left(\frac{1}{2}\right)^{n-1}\left(\frac{3}{4}\right)\Bigg]\end{aligned} \quad (24)$$

Consider any level i of the quad-tree visited by the algorithm. Let us denote Δ_i as the ratio of number of calls to the resolution function of the two algorithms at that level (for kd-tree, this includes the calls at level $2i-1$ and $2i$). From Eq. (22) and Eq. (24), we have

$$\Delta_i = \frac{16^i\alpha_0(\frac{1}{2})^{i-1}}{4^{2i-1}\beta_0(\frac{1}{2})^{i-1} + 4^{2i}\beta_0(\frac{1}{2})^{i-1}(\frac{3}{4})} = \frac{\alpha_0}{\beta_0} \quad (25)$$

Since the algorithms start at identical levels of the tree in this case, we have $\alpha_0 = \beta_0$, which further gives $\Delta_i = 1$. This means the two algorithms make exactly the same number of calls to the resolution function.

Another factor that impacts the total calls to the resolution function is the existence of empty nodes, which are automatically ignored by the algorithm. Such empty nodes may appear earlier in the kd-tree due to the existence of the rectangular nodes, and such scenarios yield a net discount to the number of function calls made by the kd-tree. On such a level $2i-1$ in the kd-tree, let us define B as number of nodes at that level, ϵ as net discount to the number of function calls, and K the number of empty nodes, we have

$$\epsilon = \left[\binom{B}{2} - \binom{B-K}{2}\right]4^{2i-1}\beta_0\left(\frac{1}{2}\right)^{i-1} \quad (26)$$

If we model the spatial distribution of data points as a random process, the expected value of K can expressed as

$$E[K] = B \cdot Pr\{X\} \quad (27)$$

where X represents the event that a cell is empty. If the data is uniformly distributed in space, we have $Pr\{X\} = (1 - \frac{1}{B})^N$ for a dataset consisting of N points. Typically, only when we move to the lower levels of the tree (such that $B \to N$) can we see a non-negligible $Pr\{X\}$. However, under skewed data distribution (e.g., Zipf), $Pr\{X\}$ becomes significantly high even at higher levels of the tree, leading to a bigger discount ϵ.

5.2 Case 2

In this case, the dual-tree algorithm starts at identical levels on the quad-tree and kd-tree, but ends at different levels. In Case 1, we have already shown that the kd-tree beats the quad-tree on the number of Type I operations, so we just need to compare the difference of Type II operations.

In this case, the leaf nodes of the quad-tree are further divided into two child nodes (representing rectangular regions in space) in the kd-tree. As a result, more nodes can be resolved by the algorithm on the kd-tree, giving rise to fewer point-to-point distance computations. Suppose there are J unresolved distances left at leaf level $(i+n)$ of the quad-tree (which is identical to level $2(i+n)$ of kd-tree). Upon calling resolution function on the next level $2(i+n)+1$ of the kd-tree, there are $\frac{3}{4}J$ unresolved distances left. Then, we have a kd-/quad-tree speedup at this level as

$$Speedup = \frac{JC_1}{\frac{3}{4}JC_1 + PC_2} \quad (28)$$

where P is number of resolution function calls made at level $2(i+n)+1$ of kd-tree, C_1 and C_2 are the costs of distance computation and resolution function call, respectively. Since each resolution function call invokes 16 distance computation (Section 3.2), we have $16C_1 = C_2$. Consequently, the denominator of Eq. (28) becomes

$$\frac{3}{4}JC_1 + 16PC_1 \quad (29)$$

Let x be the average number of the points at the level $2(i+n)+1$ of kd-tree. Since the minimum average number of points at leaf level is set to 4, the average number of points at one level up will be no less than 8, thus we have $8 > x \geq 4$. Here each resolution function resolves x^2 distances, and we called resolution function P times. On the other hand, we have the $J/4$ of distances resolved by the resolution function at the bottom level of kd-tree. Therefore, we have the following relationship between J and P.

$$\frac{J}{4} = x^2P \Rightarrow \frac{J}{4x^2} = P \quad (30)$$

By plugging Eq. (29) and Eq. (30) into Eq. (28), we have

$$Speedup = \frac{1}{\frac{3}{4} + \frac{4}{x^2}} \tag{31}$$

Since $x \in [4, 8)$, we get $Speedup \in [1, 1.2308)$. Therefore, the kd-tree algorithm again has better performance, with a speedup up to 1.23X over the quad-tree algorithm.

5.3 Case 3

The algorithm starts at an odd level of kd-tree, which does not exist in the quad-tree, and ends at the same level for both trees. The latter is the same to Case 1, therefore the efficiency depends on how many times the algorithm calls the resolution function. Although the algorithm starts earlier in the kd-tree (level $2i - 1$), the number of nodes that are unresolvable at the next level (i.e., level $2i$) is exactly the same as the starting level i of the algorithm on the quad-tree. In other words, Eq. (22) remains the same and the only change to Eq. (24) is that the first term I becomes $I/4 + I\beta$ where $I/4$ is the number of node pairs at level $2i-1$, β is the non-covering factor at level $2i - 1$, and $I\beta$ is the number of function calls at level $2i$. Here β has an upper bound of $3/4$ (Theorem 2). Therefore, as compared to Case 1, the kd-tree beats the quad-tree by an even bigger margin. However, the extra margin is negligible because it only reflects the changes to the first item in Eq. (24), which is the one with the lowest order in the series. In other words, Case 3 is almost the same scenario as Case 1.

5.4 Case 4

This case combines the differences between the quad-tree and kd-tree as discussed in Cases 2 and 3: the kd-tree starts running at a higher (odd) level, and it ends at the extra leaf level that is not in the quad-tree. Since we have shown that both scenarios lead to performance advantages of the kd-tree, we conclude the kd-tree is the winner again. Furthermore, the performance gap between kd-tree and quad-tree can be modeled by Eq. (26) and Eq. (28).

6 Extension to 3D Data

In this section, we present the analysis on 3D datasets. In 3D systems, based on the same partitioning method as in 2D, the quad-tree (now named oct-tree) bisects its x-, y-, and z-dimension at each partition. Consequently, each internal node of an oct-tree has eight children (instead of four as in quad-tree). Given the same dataset, kd-tree introduces two extra levels of nodes in between any two neighboring levels of the oct-tree, in contrast to only one such extra level in 2D data (Fig. 9). Following the SDH start/stop condition adopted in Section 3.2, we have nine scenarios to consider in performance comparison (Fig. 10).

Our previous work [25] has shown Theorem 1 is also true for oct-tree. We could still follow the geometric modeling approach mentioned earlier to study the performance of the kd-tree-based algorithm for 3D data. However, the case of 3D is too complex to yield any closed-form formulae towards an analysis as rigorous as in 2D data. Fortunately,

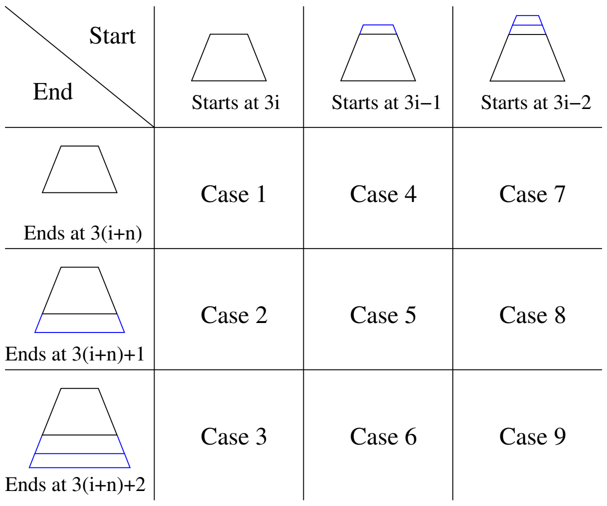


Fig. 10: Nine cases in running the kd-tree-based algorithm. Note that level $3i$ corresponds to level i in the oct-tree

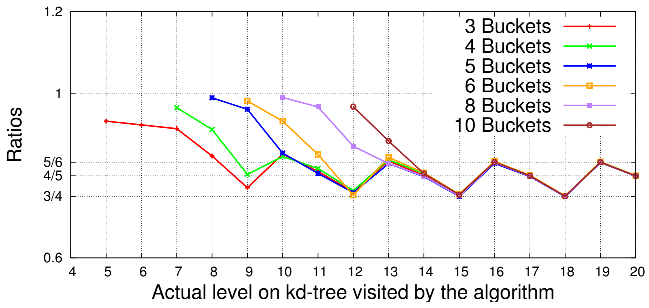


Fig. 11: Ratio of the non-covering factors of two neighboring levels visited by the kd-tree-based algorithm in processing a uniformly distributed 10-million-atom dataset. Each line represents one run under a particular p value. In each line, the ratio of non-covering factor converges very well to what Conjecture 1 states after the first 3 levels

via a large number of simulations, we found that the non-covering factor of kd-tree under 3D data has the following patterns.

Conjecture 1. *Let DM_m be a level of the kd-tree built for 3D data, and all nodes in DM_m are cubes (i.e., an identical DM exists in the corresponding oct-tree). Denote α_m as the non-covering factor of level m, we have*

$$\lim_{p\to 0} \frac{\alpha_{m+1}}{\alpha_m} = \frac{5}{6}, \lim_{p\to 0} \frac{\alpha_{m+2}}{\alpha_{m+1}} = \frac{4}{5}, \lim_{p\to 0} \frac{\alpha_{m+3}}{\alpha_{m+2}} = \frac{3}{4}$$

Conjecture 1 can be viewed as a 3D version of Theorem 2. It is easy to see that the product of the three constants in it is 1/2, which is consistent with Theorem 1 for the oct-tree and we conclude the time complexity is again the same for both trees under 3D. We have run simulations under many different sets of parameters and the results consistently support the conjecture. Results of one such experiment are shown in Fig. 11. Based on this, we will quantitatively compare the actual execution time of both algorithms under the cases shown in Fig. 10.

6.1 Start/Stop at The Same Level (Case 1)

The scenario is the same as what was discussed in Section 5.1 except there are two extra levels of DM in the kd-tree between those corresponding to any two neighboring levels in the oct-tree. In the oct-tree, if a pair of nodes is not resolvable at current level, it calls resolution function for 64 pairs of nodes at the children's level. So, after the algorithm worked on resolving all the nodes at current level, it leaves $64\alpha_0 I$ pairs unresolved. Consequently, after the algorithm worked on resolving all the nodes at level $i+1$, $64^2\alpha_0\alpha_1 I$ pairs remain unresolved, and so on. Thus, the total number of calls to the resolution function on oct-tree is

$$R = I[1 + 64\alpha_0 + 64^2\alpha_0\alpha_1 + \cdots + 64^n\alpha_0\alpha_1\cdots\alpha_{n-1}] \quad (32)$$

Again, based on the Theorem 1, we have

$$R = I\left[1 + 64\alpha_0 + 64^2\alpha_0\left(\frac{1}{2}\right) + \cdots + 64^n\alpha_0\left(\frac{1}{2}\right)^{n-1}\right] \quad (33)$$

In the kd-tree, if a pair of nodes cannot be resolved at current level, it will visit 4 pairs of nodes at its child level, therefore the total number of resolution function calls is

$$R' = I\left[1+4\beta_0+4^2\beta_0\beta_1+4^3\beta_0\beta_1\beta_2+\cdots+4^{3n}\beta_0\beta_1\cdots\beta_{3n-1}\right] \quad (34)$$

With Conjecture 1, we have

$$\begin{aligned} R' =& I\Bigg[1 + 4\beta_0 + 4^2\beta_0\left(\frac{4}{5}\right) + 4^3\beta_0\left(\frac{4}{5}\right)\left(\frac{3}{4}\right) + 4^4\beta_0\left(\frac{1}{2}\right) \\ &+ \cdots + 4^{3n}\beta_0\left(\frac{4}{5}\right)^n\left(\frac{3}{4}\right)^n\left(\frac{6}{5}\right)^{n-1}\Bigg] \end{aligned} \quad (35)$$

Similarly, let us denote Δ_i as the ratio of number of calls to the resolution function of oct-tree to kd-tree at level i of the oct-tree visited (for kd-tree, this includes the calls at level $3i-2$, $3i-1$, and $3i$). From Eq. (33) and Eq. (35), and also considering $\alpha_0 = \beta_0$ (since both algorithms start at identical DMs in each tree), we have

$$\Delta_i = \frac{64^i\alpha_0(\frac{1}{2})^{i-1}}{\beta_0(\frac{1}{2})^{i-1}\left[4^{3i-2} + 4^{3i-1}(\frac{5}{6}) + 4^{3i}(\frac{5}{6})(\frac{4}{5})\right]} = \frac{16}{15} \quad (36)$$

Therefore, the kd-tree-based algorithm makes fewer (15/16 to be specific) calls to the resolution function comparing to the quad-tree algorithm.

In addition, the appearance of empty nodes also impacts the total calls to the resolution function. In the 3D system, since the kd-tree has two extra levels, more empty nodes will appear in such intermediate levels.

6.2 Stop Further (Case 2 and 3)

This scenario is a counterpart of case 2 in 2D: we only need to compare cases that kd-tree has one and two extra level(s) resulting in differences in numbers of Type II opreations.

6.2.1 *Case 2*

In this case, the leaf nodes of oct-tree are further partitioned into two child nodes in the kd-tree. As a result, more nodes can be resolved, and less point-to-point distance computations are required for the kd-tree. Suppose there are J unresolved distances left at leaf level $(i+n)$ of the oct-tree (which is identical to level $3(i+n)$ of kd-tree). Upon calling resolution function on the next level $3(i+n)+1$ of the kd-tree, there are $\frac{5}{6}J$ unresolved distances left. Then, we have a kd-/oct-tree speedup at this level as

$$Speedup = \frac{JC_1}{\frac{5}{6}JC_1 + PC_2} \quad (37)$$

where P is number of resolution function calls made at level $3(i+n)+1$ of kd-tree, C_1 and C_2 are the costs of distance computation and resolution function call, respectively. In a 3D system, each of resolution function call requires $8\times 8 = 64$ distance computations. Then, we could substitute the C_2 with $64C_1$ to the denominator in Eq. (37),

$$\frac{5}{6}JC_1 + 64PC_1 \quad (38)$$

Similarly, let x be the average number of the points at the level $3(i+n)+1$ of kd-tree. Since our threshold b (average number of points at leaf level) is set to be equal or greater than 8, and the average number of points at one level in advance will not be less than 16, we have $16 > x \geq 8$. The number of distances resolved by the resolution function is x^2P, and there are $J/6$ distances resolved by the function at $3(i+n)+1$ level of kd-tree. Therefore, we have

$$\frac{J}{6} = x^2P \Rightarrow \frac{J}{6x^2} = P \quad (39)$$

Plugging Eq. (38) and Eq. (39) into Eq. (37), we have

$$Speedup = \frac{1}{\frac{5}{6} + \frac{64}{6x^2}} \quad (40)$$

Since $x \in [8, 16)$, we get $Speedup \in [1, 1.1429)$. Thus, the performance of kd-tree beats that of oct-tree.

6.2.2 *Case 3*

In this case, the leaf nodes of oct-tree are partitioned into four child nodes in the kd-tree. Similarly, in the kd-tree, more nodes can be resolved by the resolution function call, fewer distance computations are required. Suppose there are J unresolved distances left at leaf level $(i+n)$ of the oct-tree. After calling the resolution function at the next two levels $3(i+n)+1$ and $3(i+n)+2$ of the kd-tree, there are $(\frac{5}{6}\cdot\frac{4}{5})J$ distances left. Then, we have a kd-/oct-tree speedup as

$$Speedup = \frac{JC_1}{\frac{5}{6}\cdot\frac{4}{5}JC_1 + (P_1+P_2)C_2} \quad (41)$$

where P_1 and P_2 is number of resolution function calls made at level $3(i+n)+1$ and $3(i+n)+2$ of kd-tree, C_1 and C_2 are the costs of distance computation and resolution function call, respectively. Similarly, we have $C_2 = 64C_1$, then the denominator of Eq. (41) becomes

$$\frac{4}{6}JC_1 + 64C_1(P_1+P_2) \quad (42)$$

Let x_1 and x_2 be the average number of the points at the level $3(i+n)+1$ and $3(i+n)+2$, respectively. Similarly, by having the pre-defined threshold $b = 8$, we get $32 > x_1 \geq 16 > x_2 \geq 8$. In addition, the number of distances resolved by the function at the last two levels of kd-tree are $J/6$ and $1/5 \times 5J/6$, respectively. This leads to

$$\begin{cases} \frac{1}{6}J = x_1^2 P_1 & \text{at level } 3(i+n)+1 \\ \frac{1}{5} \times \frac{5}{6}J = x_2^2 P_2 & \text{at level } 3(i+n)+2 \end{cases} \quad (43)$$

By plugging Eq. (42) and Eq. (43) into Eq. (41), we have

$$Speedup = \frac{1}{\frac{2}{3} + \frac{64}{6x_1^2} + \frac{64}{6x_2^2}} \quad (44)$$

Since $x_1 \in [16, 32)$ and $x_2 \in [8, 16)$, we have $Speedup \in [1.1429, 1.3913)$. Thus, the kd-tree outperforms oct-tree with a speedup up to 1.39X.

6.3 Start Earlier (Case 4 and 7)

This scenario is similar to Case 3 in 2D analysis: the algorithm starts at one or two level(s) earlier on kd-tree, and stops at identical levels in both oct-tree and kd-tree. Thus, the difference lies on the number of resolution function calls.

6.3.1 Case 4

In this case, the algorithm starts one level earlier in the kd-tree (level $3i-1$). Similarly, Eq. (33) is unchanged, and the only change to Eq. (35) is that the first term I becomes $I/4 + I\beta$ where I/4 is the number of node pairs at level $3i-1$, β is non-covering factor at level $3i-1$, and $I\beta$ is number of function calls at level $3i$. Here β has an upper bound of 5/6 (Conjecture 1). Again, as such numbers are very small, it does not change the conclusion we made in Case 1.

6.3.2 Case 7

In this case, the dual-tree algorithm starts two levels earlier on the kd-tree (level $3i-2$). Again, Eq. (33) is unchanged, and the change to Eq. (35) is that the first term becomes $I/16 + I/4\beta' + I\beta''$, where I/16 is number of node pairs at level $3i-2$, I/4 is number of node pairs at level $3i-1$, β' is the non-covering factor at level $3i-2$, β'' is non-covering factor at level $3i-1$, $I/4\beta'$ is number of function calls at level $3i-1$, and $I\beta''$ is number of function calls at level $3i$. Here β' and β'' have upper bound of 3/4 and 5/6, respectively. This, again, does not change the results of Case 1.

6.4 Start Earlier, Stop Further (Case 5, 6, 8, and 9)

In this scenario, the dual-tree algorithm starts at one or two level(s) earlier and stops at one or two level(s) further. We can simply combine the aforementioned cases to carry out the performance analysis: Case 5 can be modeled by Eq. (36) and Eq. (37); Case 6 can be modeled by Eq. (36) and Eq. (41); Case 8 can be modeled by Eq. (36) and Eq. (37); and Case 9 can be modeled by Eq. (36) and Eq. (41).

7 Experimental Evaluation

We have implemented both algorithms with the C++ programming language and our experiments were run on a Mac OS X (El Capitan) server with an Intel i7-6700K Quad-Core 4.0GHz processor and 16GB of 1867MHz DDR3 memory. We used one real dataset, which was generated from a molecular dynamics study to simulate a bilayer membrane lipid system, and two synthetic datasets that represent different spatial distributions of data (i.e., *Uniform* and *Zipf* with order 1.0) in our experiments. All synthetic data was generated within a box with lateral length 25,000. All experiments were run under a series of histogram resolutions (i.e., 4-10 buckets) and different system sizes (i.e., 100,000 to 1,600,000 points). Note that the total number of buckets in the histogram (or bucket width p) determines which tree level the algorithm starts, and the data size determines which level the algorithm stops. Therefore, we set those two numbers in different ways to create all the cases discussed in Sections 5 and 6.

7.1 Results for 2D data

We first evaluate our analysis related to Case 1 of 2D data. Fig. 12a shows the recorded Δ_i values under different numbers of tree levels visited by the algorithm (i.e., m in Theorem 2). For the uniformly distributed data, Δ_i is close to 1 for most the levels. For smaller i, we observe smaller Δ_i values. This is due to the modeling errors caused by the coarse grid, as discussed at the end of Section 4. Note that such errors disappear at $m = 3$ in Fig. 12a. For the Zipf data, we see Δ_i values greater than 1 for larger i - this is due to the fact that empty nodes are found earlier in kd-tree. Such results confirm our analysis shown in Section 5.1.

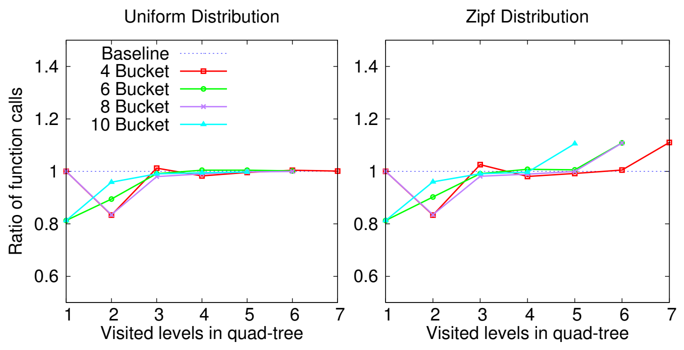
Related to Case 2, Fig. 12b shows the ratio of total number of distance computations (i.e., Type II operations) made by the two trees. Recall this is the case where the kd-tree has an extra level on the bottom. The curves converge to 4/3 in the uniformly distributed data, meaning the kd-tree saves 1/4 of the distance computations. For the skewed data, we see more fluctuations in the results, and the speedup is even higher than those in uniform data for most of the cases. This confirms the analysis shown in Eq. (30).

TABLE 3: Ranges of speedup (kd-tree over quad-tree) observed in all cases of 2D experiments shown in Fig. 13

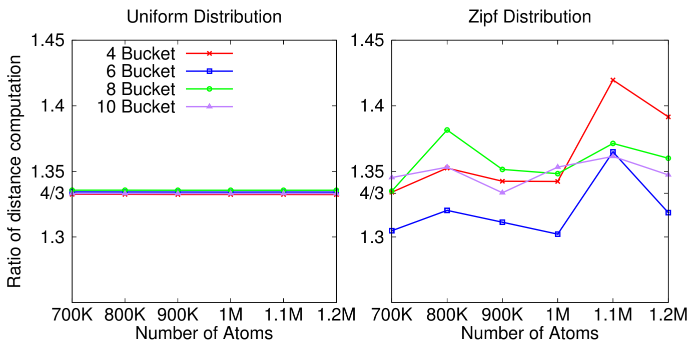
Scenario	Data Type		
	Uniform	Zipf	Real
Case 1	0.993 – 1.002	0.974 – 0.996	0.996 – 1.006
Case 2	1.052 – 1.204	1.159 – 1.230	1.084 – 1.219
Case 3	0.994 – 1.005	0.984 – 1.004	0.993 – 1.004
Case 4	1.042 – 1.212	1.154 – 1.228	1.095 – 1.229

Fig. 13 plots the actual running time of the two algorithms under different data sizes and data distributions. The ranges of speedup of kd-tree over quad-tree we observed in such experiments are presented in Table 3. Let us first discuss the results of Case 2 (Fig. 13c) and Case 4 (Fig. 13d): the kd-tree outperforms the quad-tree in all experimental runs, and the gap is significant with the highest speedup reaching 1.23X. This indicates that the reduced distance computations caused by the extra level on the bottom of the kd-tree plays a significant role in boosting performance, and the expected speedup of $[1X, 1.2308X]$ mentioned in Section 5.2 is an accurate estimation.

For Case 1 (Fig. 13a) and Case 3 (Fig. 13b), the performance of the two trees is very close. We also notice that there are cases where the kd-tree is slightly outperformed

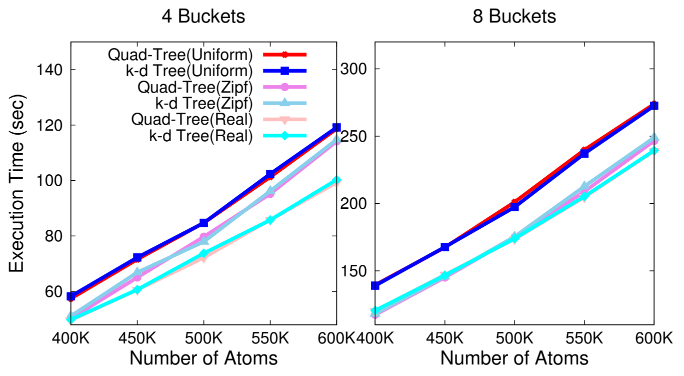


(a) Ratio of resolution function calls

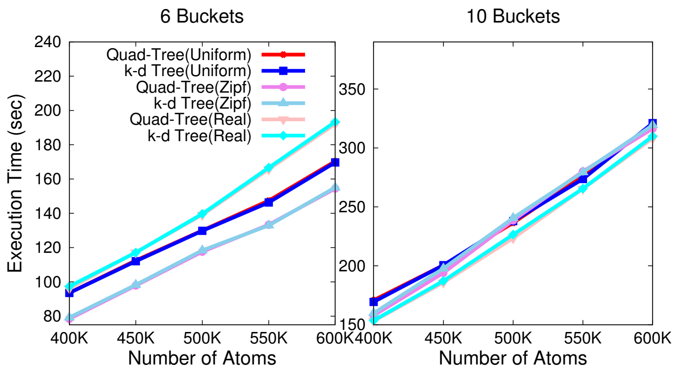


(b) Ratio of distance computations

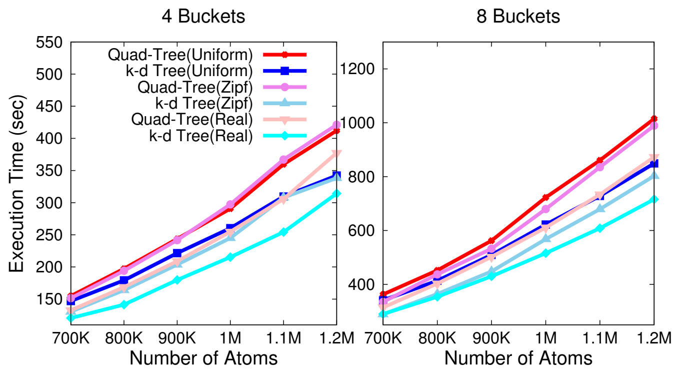
Fig. 12: Ratios of (a) Type I operations and (b) Type II operations made by quad-tree vs. that by the kd-tree under different histogram bucket numbers and data distribution patterns (i.e., uniform and Zipf)



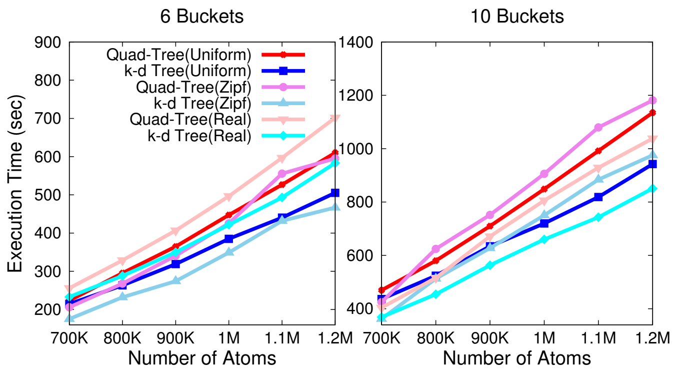
(a) Case 1



(b) Case 3

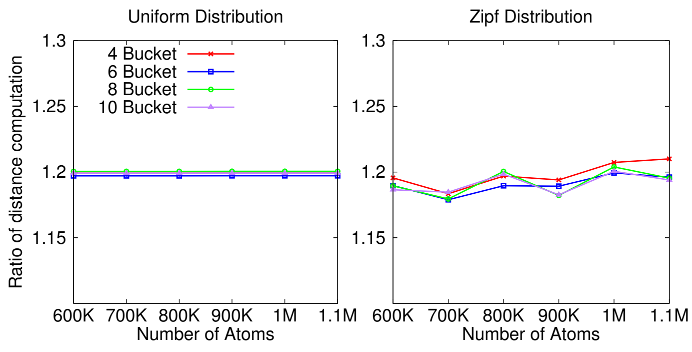


(c) Case 2

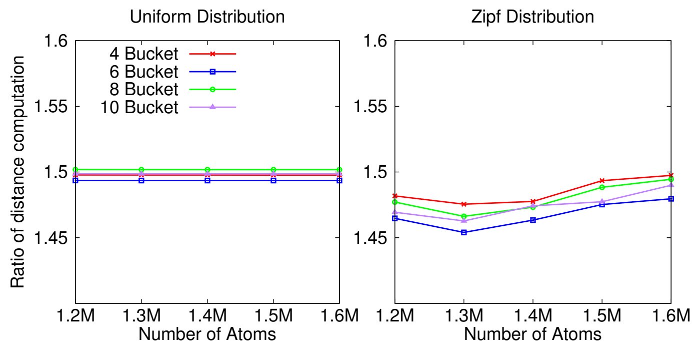


(d) Case 4

Fig. 13: Running time of the dual-tree algorithms in 2D systems under different data sizes and data distribution patterns



(a) Case 2: kd-tree has one extra level



(b) Case 3: kd-tree has two extra levels

Fig. 14: Ratios of Type II operations performed by oct-tree vs. that by the kd-tree under different data sizes, p values, and data distribution patterns

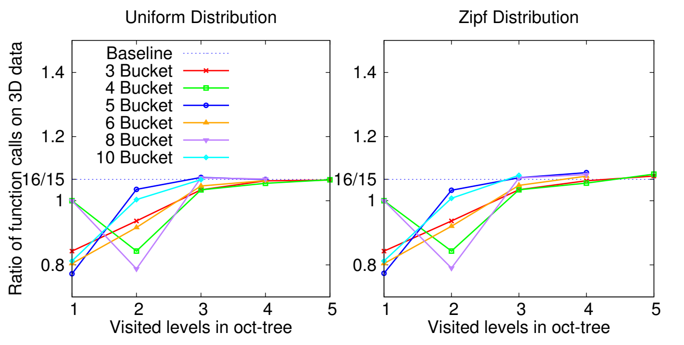


Fig. 15: Ratios of Type I operations performed by oct-tree vs. that by the kd-tree under different values of m, p, and data distribution patterns for a 10-million-point 3D dataset

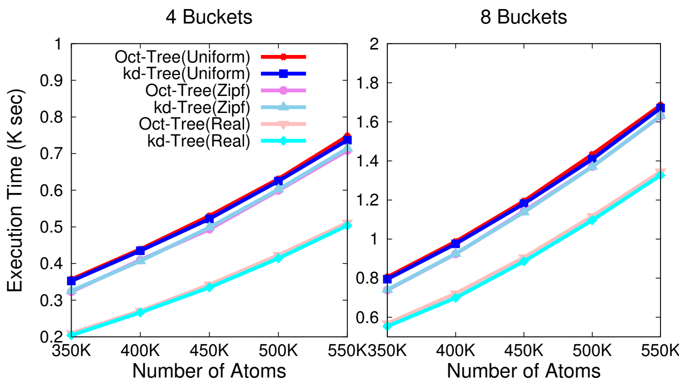
by the quad-tree (Table 3). This seems to be contradictory to our findings in Sections 5.1 and 5.3. Our explanation is: the data access pattern of the quad-tree naturally has better spatial locality which gives rise to higher cache hit rate. Specifically, when calling the resolution function, the OS could load all 4 sibling nodes (in consecutive memory addresses) at a time while there are only two children per node in the kd-tree. We collected the number of cache misses of two implementations by the *perf* tool under Linux, and found that the kd-tree has 1.5X-2X cache misses comparing with the quad-tree. The impact of such is seen more clearly for the Zipf data in Case 1, in which the quad-tree won in all cases (although with a small margin). This is because, the Zipf distribution rendered much less distance computations therefore more efficient resolution function call shows more positive effects on total performance.

7.2 Results for 3D Data

We first verify the key results for Case 1 studied in Section 6.1. Fig. 15 shows the Δ_i values recorded under different numbers of levels i visited by the algorithm for 3D data. For the uniform data, Δ_i approaches 16/15 (baseline) as expected from Eq. (36) when i is beyond 3. For smaller i values, we have unstable Δ_i values. This is similar to 2D system: coarse grid causes fluctuations on non-covering factors. For the Zipf data, the Δ_i values are greater than 16/15 for larger i, this is, much like the 2D cases, caused by the earlier appearance of empty nodes in the kd-tree.

For Case 2, Fig. 14a shows the ratio of the number of distance computations performed by the oct-tree vs. kd-tree. For the uniform data, the ratios are all very close to 6/5. This means the kd-tree saves 1/6 of the distance computations performed by the oct-tree, confirming our findings in Conjecture 1. Under Case 3 (Fig. 14b) such ratios are all close to 1.5, indicating the kd-tree saves 1/3 of the distance computations over oct-tree. This further validates Conjecture 1, as $1.5 = 6/5 \times 5/4$. For both cases, the results of the Zipf data show more fluctuations, and in most cases the ratio is smaller than the 1.2 and 1.5 found in uniform data. Our explanation is: skewed data is known to have distances resolved earlier as compared to uniform data [25]. At any level of the tree, although the average number of data points in the nodes is the same as in uniform data, we could see more nodes with fewer points due to the skewed spatial distribution. As a result, the advantage of adding extra levels in the kd-tree is less significant. Nevertheless, the kd-tree is still the obvious winner in performance.

We also recorded the total running time of both algorithms under the nine different cases discussed in Section 6. In summary, the kd-tree outperforms oct-tree in every experimental run we conducted, and the speedup in all cases are within the range suggested by our analysis. A special note here is: Fig. 16, Figs. 17c and 17f represent different cases in which the oct-tree and kd-tree have identical leaf nodes. The three lines representing kd-tree results (under different input data types) are all slightly lower than their corresponding oct-tree lines under all data sizes, although such difference is small. For all other cases (i.e., cases 2, 3, 5, 6, 8, 9), the performance advantage of kd-tree over oct-tree is more significant thus can be clearly seen in the figures.

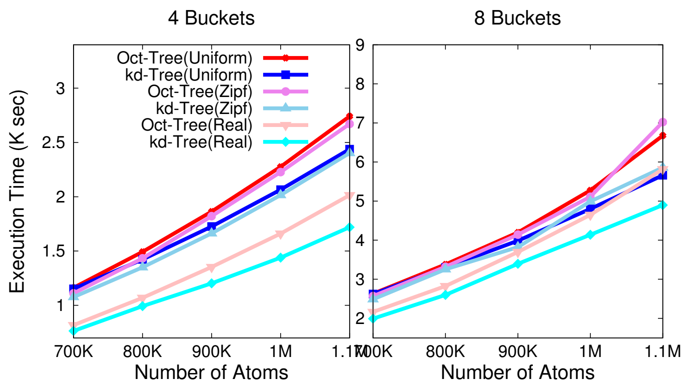


(a) Case 1

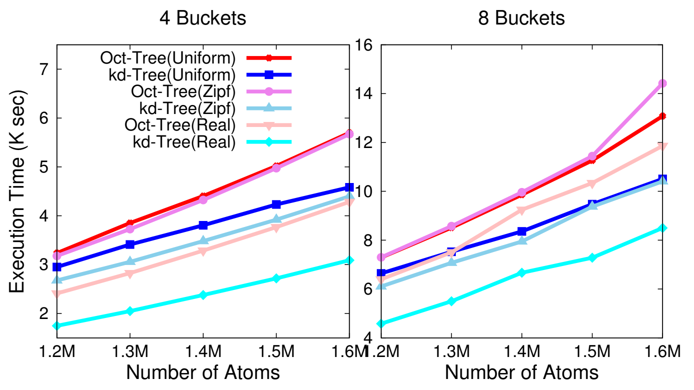
Fig. 16: Total running time of the dual-tree algorithm running on top of oct-tree and kd-tree under different data sizes and data distribution - Case 1

8 Conclusions

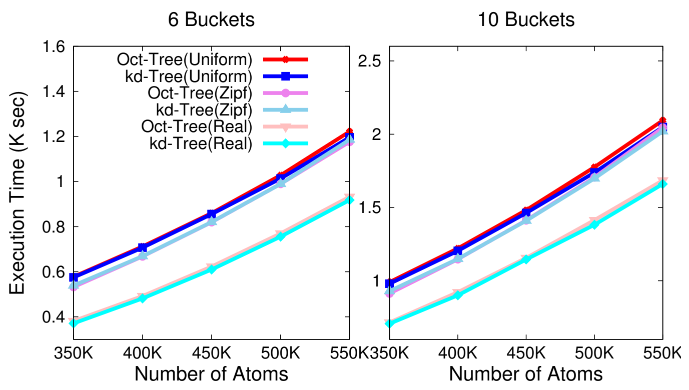
SDH is a type of 2-body statistics that found applications in many computing domains. Being the main building block of high-level analytics, SDH is of great importance in statistical learning and scientific discovery. In the past years, research on efficient processing of SDH has settled on a series of *dual-tree* algorithms that work on resolving distances between pairs of nodes of a spatial tree. Main implementations of the dual-tree algorithm are based on quad/oct-tree, which partitions data space along all dimensions, and the kd-tree, which does so along a single dimension. In this paper, we present quantitative analysis on the performance of dual-tree algorithms based on these two types of tree structures. Our analysis established on a geometric modeling framework suggests the kd-tree-based algorithm outperforms the quad-/oct-tree-based algorithm with different data sizes and histogram resolution. We also provide bounds for the speedup of kd-tree over quad-/oct-tree, and extensive experiments with both synthetic and real data inputs confirm our findings. We believe our results and methodology can also provide insights on analyzing similar algorithms for processing more general n-body statistics.



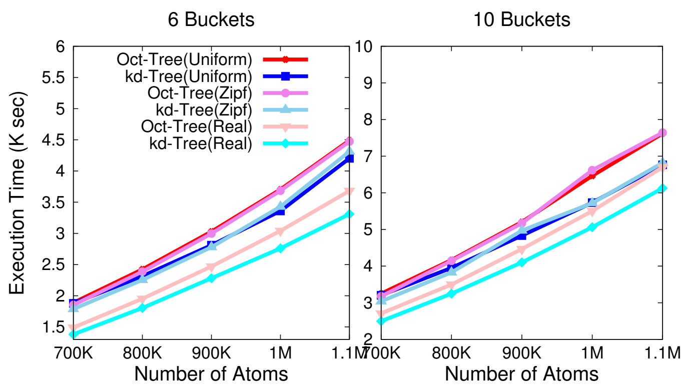
(a) Case 2



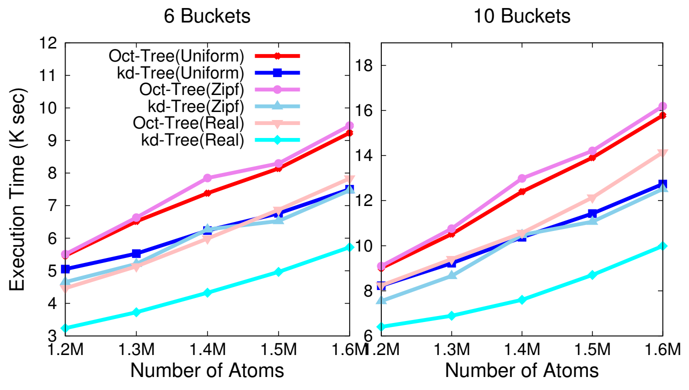
(b) Case 3



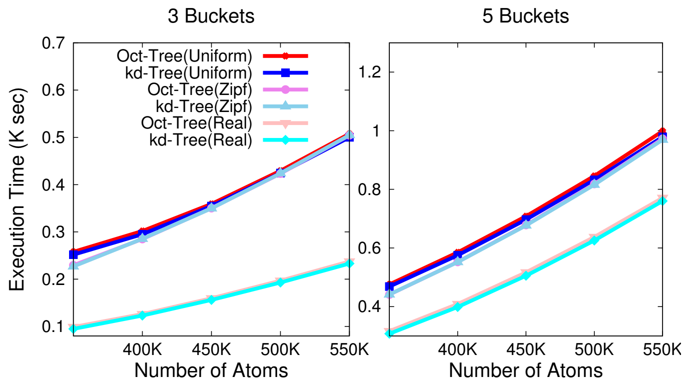
(c) Case 4



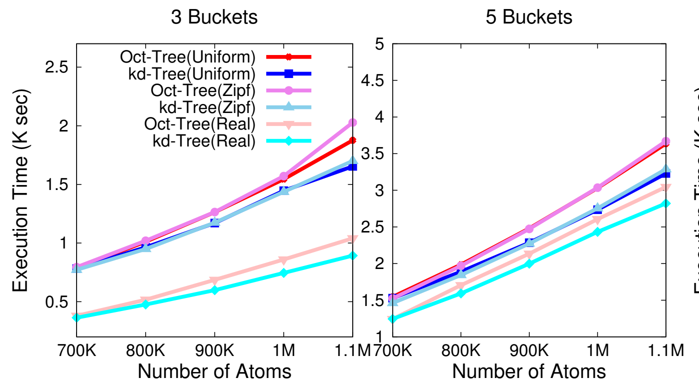
(d) Case 5



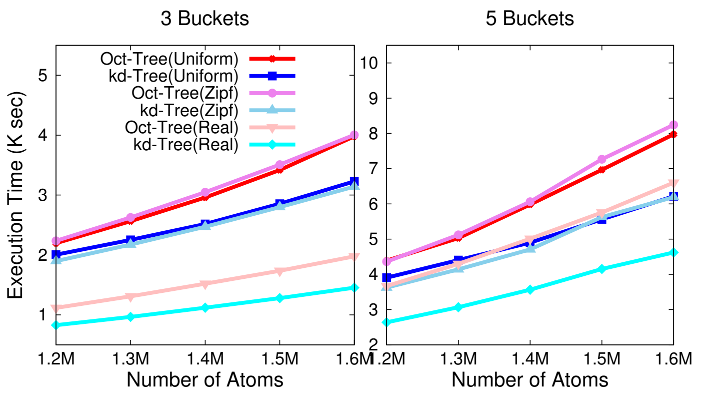
(e) Case 6



(f) Case 7



(g) Case 8



(h) Case 9

Fig. 17: Total running time of the dual-tree algorithm running on top of oct-tree and kd-tree under different data sizes and data distribution - other cases

ACKNOWLEDGMENT

This work is supported by an award (IIS-1253980) from the National Science Foundation (NSF) of U.S.A.. Equipments used in the experiments are partially supported by another grant (CNS-1513126) from the same agency.

REFERENCES

[1] J. Pfaltz and R. Orlandic (1999) A scalable dbms for large scientific simulations *1999 International Symposium on Database Applications in Non-Traditional Environments*, Kyoto, Japan, 28-30 Nov. 1999, pp. 271 – 275, IEEE

[2] X. Fei and S. Lu (2010) A collectional data model for scientific workflow composition. *2010 IEEE International Conference on ICWS*, Miami, FL, USA, 5-10 July 2010, pp. 567 – 574, IEEE

[3] D. E. Shaw, and et al. (2008) Anton, a special-purpose machine for molecular dynamics simulation. *Communications of the ACM*, vol. 51, pp. 91–97.

[4] D. Howe, and et al. (2008) Big data: The future of biocuration. *Nature*, vol. 455, pp. 47–50.

[5] B. A. Huberman (2012) Sociology of science: Big data deserve a bigger audience. *NATURE*, vol. 482, p. 308.

[6] D. Centola (2010) The spread of behavior in an online social network experiment *Science*, vol. 329, no. 5996, pp. 1194–1197.

[7] S. Lakshminarasimhan, and et al. (2011) Isabela-qa: Query-driven analytics with isabela-compressed extreme-scale scientific data. *2011 IEEE International Conference for High Performance Computing*, Seatle, WA, USA, 12-18 Nov. 2011, no. 1-11, IEEE

[8] M. Weidner, J. Dees, and P. Sanders (2013) Fast olap query execution in main memory on large data in a cluster. *2013 IEEE International Conference on Big Data*, Silicon Valley, CA, USA, 6-9 Oct. 2013, pp. 518 – 524, IEEE

[9] Y.-C. Tu, S. Chen, and S. Pandit (2009) Computing distance histograms efficiently in scientific databases. *IEEE 25th International Conference on Data Engineering (ICDE*, Shanghai, China, 29 March-2 April 2009, pp. 796–807, 2009, IEEE

[10] C.-C. Mou (2015) A comparative study of dual-tree algorithms for computing spatial distance histogram in scientific databases. *Master's thesis, University of South Florida, Tampa, Florida, USA,*

[11] S. Klasky, B. Ludaescher, and M. Parashar (2006) The center for plasma edge simulation workflow requirements *IEEE 22nd International Conference on Data Engineering Workshops*, Atlanta, GA, USA, 3-7 April 2006, p. 73, IEEE

[12] J.-L. Starck and F. Murtagh (2002) Astronomical image and data analysis *Springer*, New York City

[13] M. P. Allen and D. J. Tildesley (1987) Computer simulations of liquids. *Clarendon Press, Oxford*.

[14] A. Filipponi (1994) The radial distribution function probed by x–ray absorption spectroscopy. *Journal of Physics: Condensed Matter*, vol. 6, no. 8415–8427.

[15] V. Springel, and et al. (2005) Simulations of the formation, evolution and clustering of galaxies and quasars. *Nature*, vol. 435, pp. 629–636.

[16] J. Huang, S. R. Kumary, M. Mitra, W.-J. Zhu, and R. Zabih (1997) Image indexing using color correlograms. *1997 IEEE Conference on Computer Vision and Pattern Recognition*, San Juan, USA, 17-19 June 1997, pp. 762 – 768, IEEE

[17] G. Heidemann (2004) Combining spatial and colour information for content based image retrieval. *Computer Vision and Image Understanding*, vol. 94(1), pp. 234–270

[18] M. Ankerst, G. Kastenmüller, H.-P. Kriegel, and T. Seidl (1999) 3D shape histograms for similarity search and classification in spatial databases. *Proc. 6th Int. Symposium on Spatial Databases*, Hong Kong, China, 25 June 1999, pp. 207–226, Springer, Heidelberg

[19] G. V. and G. O. (1998) Multidimensional access methods," *ACM Computing Surveys*, vol. 30, no. 2 pp.170-231

[20] A. Moore, and et al. (2006) Fast algorithms and efficient statistics: N-point correlation functions. In:*Banday A.J., Zaroubi S., Bartelmann M. (eds) Mining the Sky. ESO ASTROPHYSICS SYMPOSIA (European Southern Observatory).* Springer, Berlin, Heidelberg.

[21] A. Gray and A. Moore (2000) N-body problems in statistical learning. In: *T. K. Leen and T. G. Dietterich (eds), Advances in Neural Information Processing Systems 13*, MIT Press, Cambridge, Massachusetts

[22] J. Tsang (2008) Evolving trajectories of the n-body problem *IEEE Congress on CEC*, Hong Kong, China, 1-6 June 2008, pp. 3726 – 3733.

[23] K. Tsoi, C. Ho, H. Yeung, and P. Leong, (2005) An arithmetic library and its application to the n-body problem. *12th Annual IEEE Symposium on FCCM*, Napa, CA, USA, 20-23 April 2004, pp. 68–78, IEEE

[24] L. Perrone and D. Nicol, (2000) Using n-body algorithms for interference computation in wireless cellular simulations. *8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, San Francisco, CA, USA, 29 Aug.-1 Sept. 2000, pp. 49 – 56, IEEE

[25] S. Chen, Y.-C. Tu, and Y. Xia (2011) Performance analysis of a dual-tree algorithm for computing spatial distance histograms. *VLDB Journal*, vol. 20, no. 4, pp. 471–494

[26] A. Kumar, V. Grupcev, Y. Yuan, Y.-C. Tu, and G. Shen (2012) Distance histogram computation based on spatiotemporal uniformity in scientific data. In *Proceedings of 15th International Conference on Extending Database Technology (EDBT)*, Berlin, Germany, March 27 - 30, 2012, pp. 288–299.

[27] V. Grupcev, Y. Yuan, Y. Tu, J. Huang, S. Chen, S. Pandit, and M. Weng (2013) Approximate algorithms for computing spatial distance histograms with accuracy guarantees. *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 9, pp. 1982–1996, . [Online]. Available: https://doi.org/10.1109/TKDE.2012.149

[28] A. Kumar, V. Grupcev, Y. Yuan, J. Huang, Y. Tu, and G. Shen (2014) Computing spatial distance histograms for large scientific data sets on-the-fly. *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 10, pp. 2410–2424, [Online]. Available: https://doi.org/10.1109/TKDE.2014.2298015

[29] C. Mou, S. Chen, and Y. Tu (2016) A comparative study of dual-tree algorithm implementations for computing 2-body statistics in spatial data. In *2016 IEEE International Conference on Big Data, BigData 2016, Washington DC, USA, December 5-8, 2016*, pp. 2676–2685, IEEE, [Online]. Available: https://doi.org/10.1109/BigData.2016.7840911

Appendix A
Equations Not Shown in Section 4

$$\begin{aligned} S_{Q_1Q_2Q_3Q_4} =& 4(S_{\widehat{Q_4Q_1}O_3} - S_{\triangle Q_4O_3B} - S_{\triangle Q_1O_3C} + S_{O_3BDC}) \\ =& 2\left(\frac{\pi}{2} - \arcsin\frac{\delta}{4p} - \arcsin\frac{\delta}{2p}\right)p^2 - \frac{\delta}{2}\sqrt{p^2 - \left(\frac{\delta}{4}\right)^2} - \delta\sqrt{p^2 - \left(\frac{\delta}{2}\right)^2} + \frac{\delta^2}{2} \end{aligned} \tag{45}$$

$$g(i) = \begin{cases} \left(\frac{5}{4}\pi + \frac{3\sqrt{5}+1}{2}\right)\delta^2 & i = 1 \\ \Big\{\frac{5}{4}\pi i^2 + \frac{3\sqrt{5}}{2}i - \Big[\Big(\frac{5\pi}{4} - \frac{5}{2}\arcsin\frac{\sqrt{5}}{10(i-1)} - \frac{5}{2}\arcsin\frac{\sqrt{5}}{5(i-1)}\Big)(i-1)^2 & \\ \quad -\frac{1}{2}\sqrt{\frac{5}{4}(i-1)^2 - \frac{1}{16}} - \sqrt{\frac{5}{4}(i-1)^2 - \frac{1}{4}}\Big]\Big\}\delta^2 & i > 2 \end{cases} \tag{46}$$

$$S_{A'} = \begin{cases} 4p^2 \arccos\frac{\delta}{4p} - \frac{\delta\sqrt{p^2 - \left(\frac{\delta}{4}\right)^2}}{2}, & m = 1 \\ \pi p^2 + 2p\left(1 - \frac{2}{2^{\frac{m}{2}}}\right)\delta + 2p\left(1 - \frac{2}{2^{\frac{m}{2}}}\right)\frac{\delta}{2} + \left(1 - \frac{2}{2^{\frac{m}{2}}}\right)\delta \cdot \left(1 - \frac{2}{2^{\frac{m}{2}}}\right)\frac{\delta}{2}, & m \text{ is even } (\geq 2) \\ \pi p^2 + 2p\left(1 - \frac{1}{2^{\frac{m-1}{2}}}\right)\delta + 2p\left(1 - \frac{2}{2^{\frac{m-1}{2}}}\right)\frac{\delta}{2} + \left(1 - \frac{1}{2^{\frac{m-1}{2}}}\right)\delta \cdot \left(1 - \frac{2}{2^{\frac{m-1}{2}}}\right)\frac{\delta}{2}, & m \text{ is odd } (\geq 2) \end{cases} \tag{47}$$

$$S_{out}(i) = \begin{cases} \left[\frac{5}{4}\pi i^2 + \frac{3\sqrt{5}}{2}\left(1 - \frac{2}{2^{\frac{m}{2}}}\right)i + \frac{1}{2}\left(1 - \frac{2}{2^{\frac{m}{2}}}\right)^2\right]\delta^2, & m \text{ is even}, m \geq 2 \\ \left[\frac{5}{4}\pi i^2 + \sqrt{5}\left(1 - \frac{1}{2^{\frac{m-1}{2}}}\right)i + \frac{\sqrt{5}}{2}\left(1 - \frac{2}{2^{\frac{m-1}{2}}}\right)i + \frac{1}{2}\left(1 - \frac{1}{2^{\frac{m-1}{2}}}\right)\left(1 - \frac{2}{2^{\frac{m-1}{2}}}\right)\right]\delta^2, & m \text{ is odd}, m \geq 2 \end{cases} \tag{48}$$

$$S_{shade}(i) = \begin{cases} \left(\frac{5\beta_{even}}{8}i^2 - \frac{\theta_m}{4}\sqrt{\frac{5}{4}i^2 - \frac{\theta_m^2}{4}} - \frac{\theta_m}{2}\sqrt{\frac{5}{4}i^2 - \theta_m^2} + \frac{\theta_m^2}{2} - \frac{1}{8}\right)\delta^2, & m \text{ is even} \\ \left(\frac{5\beta_{odd}}{8}i^2 - \frac{\theta_{m-1}}{4}\sqrt{\frac{5}{4}i^2 - \frac{\theta_{m-1}^2}{4}} - \frac{\theta_{m+1}}{2}\sqrt{\frac{5}{4}i^2 - \theta_{m+1}^2} + \frac{\theta_{m-1}\theta_{m+1}}{2} - \frac{1}{8}\right)\delta^2, & m \text{ is odd} \end{cases} \tag{49}$$

The area of coverable region A' maps to bucket i and density map m

$$f(i,m) = \begin{cases} \left(2\sqrt{5}\arccos\frac{\sqrt{5}}{10} - \frac{1}{2}\sqrt{\frac{5}{4} - \frac{1}{16}}\right)\delta^2, & i = 1, m = 1 \\ \left(\frac{5}{4}\pi + 3\sqrt{5}\theta_m + 2\theta_m^2\right)\delta^2, & i = 1, m \geq 2, \text{ and } m \text{ is even} \\ \left(\frac{5}{4}\pi + 2\sqrt{5}\theta_{m+1} + \sqrt{5}\theta_{m-1} + 2\theta_{m+1}\theta_{m-1}\right)\delta^2, & i = 1, m \geq 2, \text{ and } m \text{ is odd} \\ \left[\frac{5}{4}\pi i^2 + \frac{\sqrt{5}}{4}i - \frac{5}{4}\pi(i-1)^2 + \frac{\sqrt{5}}{4}(i-1)\right]\delta^2, & i > 1, m = 1 \\ \Big(\frac{5}{4}\pi i^2 + 3\sqrt{5}\theta_m i - \frac{5\beta_{even}}{2}(i-1)^2 + \theta_m\sqrt{\frac{5}{4}(i-1)^2 - \frac{\theta_m^2}{4}} & \\ \quad +2\theta_m\sqrt{\frac{5}{4}(i-1)^2 - \theta_m^2}\Big)\delta^2, & m \geq 2, \text{ and } m \text{ is even} \\ \Big(\frac{5}{4}\pi i^2 + 2\sqrt{5}\theta_{m+1}i + \sqrt{5}\theta_{m-1}i - \frac{5\beta_{odd}}{2}(i-1)^2 & \\ \quad +\theta_{m-1}\sqrt{\frac{5}{4}(i-1)^2 - \frac{\theta_{m-1}^2}{4}} + 2\theta_{m+1}\sqrt{\frac{5}{4}(i-1)^2 - \theta_{m+1}^2}\Big)\delta^2, & m \geq 2, \text{ and } m \text{ is odd} \end{cases} \tag{50}$$

Appendix B
Proof of Eq. (19) converging to 2/3

To prove the $\alpha(m+1)/\alpha(m)$ converges to 2/3, it is equivalent to prove the following equation.

$$\sum_{i=1}^{l}[3f(i,m+1)-2f(i,m)]=\sum_{i=1}^{l}g(i) \tag{51}$$

The left hand side of the Eq. (51) could be expressed as

$$\begin{aligned} LHS =& \sum_{i=1}^{l}[3f(i,m+1)-2f(i,m)] \\ =&\delta^2\sum_{i=2}^{l}\Bigg\{3\Bigg[\frac{5}{4}\pi i^2+3\sqrt{5}\theta_m i-\frac{5\beta_{even}}{2}(i-1)^2+\theta_m\sqrt{\frac{5}{4}(i-1)^2-\frac{\theta_m^2}{4}}+2\theta_m\sqrt{\frac{5}{4}(i-1)^2-\theta_m^2}\Bigg] \\ &-2\Bigg[\frac{5}{4}\pi i^2+2\sqrt{5}\theta_{m+2}i+\sqrt{5}\theta_m i-\frac{5\beta_{odd}}{2}(i-1)^2+\theta_m\sqrt{\frac{5}{4}(i-1)^2-\frac{\theta_m^2}{4}}+2\theta_{m+2}\sqrt{\frac{5}{4}(i-1)^2-\theta_{m+2}^2}\Bigg]\Bigg\} \end{aligned} \tag{52}$$

The right hand side of the Eq. (51) could be expressed as

$$\begin{aligned} RHS =& \sum_{i=1}^{l}g(i) \\ =&\delta^2\sum_{i=2}^{l}\Bigg\{\frac{5}{4}\pi i^2+\frac{3\sqrt{5}}{2}i-\Bigg[\bigg(\frac{5}{4}\pi-\frac{5}{2}\arcsin\frac{\sqrt{5}}{10(i-1)}-\frac{5}{2}\arcsin\frac{\sqrt{5}}{5(i-1)}\bigg)(i-1)^2 \\ &-\frac{1}{2}\sqrt{\frac{5}{4}(i-1)^2-\frac{1}{16}}-\sqrt{\frac{5}{4}(i-1)^2-\frac{1}{4}}\Bigg]\Bigg\} \end{aligned} \tag{53}$$

We could use the difference between LHS and RHS to prove Eq. (51)

$$\begin{aligned} LHS-RHS =&\delta^2\sum_{i=2}^{l}\Bigg\{\bigg(-3\sqrt{5}\theta_m+6\sqrt{5}\theta_{m+2}-\frac{3\sqrt{5}}{2}\bigg)\cdot i+\bigg(-\frac{3\sqrt{5}}{2}\theta_m+3\sqrt{5}\theta_{m+2}-\frac{3\sqrt{5}}{4}\bigg)(i-1) \\ &+\bigg[5\beta_{odd}-\frac{15}{2}\beta_{even}+\frac{5}{4}\pi-\frac{5}{2}\bigg(\arcsin\frac{\sqrt{5}}{10(i-1)}+\arcsin\frac{\sqrt{5}}{5(i-1)}\bigg)\bigg](i-1)^2\Bigg\} \end{aligned} \tag{54}$$

Since the m is level of the density map, when m getting larger, the approximated boundary will approach to the theoretical boundary. Therefore when m approaches to infinity, the θ approaches to $\frac{1}{2}$, thus, we can replace all the θ by $\frac{1}{2}$ into the above equation, and when $l\to\infty$, obtains,

$$\sum_{i=2}^{l}\bigg(-3\sqrt{5}\cdot\frac{1}{2}+6\sqrt{5}\cdot\frac{1}{2}-\frac{3\sqrt{5}}{2}\bigg)i=0$$

$$\sum_{i=2}^{l}\bigg(-\frac{3\sqrt{5}}{2}\cdot\frac{1}{2}+3\sqrt{5}\cdot\frac{1}{2}-\frac{3\sqrt{5}}{4}\bigg)(i-1)=0$$

$$\beta_{even}=\frac{\pi}{2}-\arcsin\frac{\theta_m\sqrt{5}}{5i}-\arcsin\frac{2\sqrt{5}\theta_{m+2}}{5i}\to\frac{\pi}{2}$$

$$\beta_{odd}=\frac{\pi}{2}-\arcsin\frac{\theta_m\sqrt{5}}{5i}-\arcsin\frac{2\sqrt{5}\theta_{m+2}}{5i}\to\frac{\pi}{2}$$

$$\frac{5}{2}\bigg(\arcsin\frac{\sqrt{5}}{10(i-1)}+\arcsin\frac{\sqrt{5}}{5(i-1)}\bigg)\to 0$$

$$\sum_{i=2}^{l}\bigg[5\beta_{odd}-\frac{15}{2}\beta_{even}+\frac{5}{4}\pi-\frac{5}{2}\bigg(\arcsin\frac{\sqrt{5}}{10(i-1)}+\arcsin\frac{\sqrt{5}}{5(i-1)}\bigg)\bigg](i-1)^2\to 0$$

Therefore, we have $LHS=RHS$, and Eq. (51) is proved.