

A Comparative Study of Dual-tree Algorithm Implementations for Computing 2-Body Statistics in Spatial Data

Chengcheng Mou

Dept. of Computer Science & Engineering
University of South Florida
Tampa, FL, USA 33620
Email: chengcheng@mail.usf.edu

Shaoping Chen

Department of Mathematics
Wuhan University of Technology
Wuhan, Hubei, P. R. China 430070
Email: chensp@whut.edu.cn

Yi-Cheng Tu

Dept. of Computer Science & Engineering
University of South Florida
Tampa, FL, USA 33620
Email: tuy@mail.usf.edu

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 quadratic 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 propose a kd-tree-based solution. Although it is easy to see that both implementations have the same time complexity $O(N^{\frac{2d-1}{d}})$, 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 all scenarios with different data sizes and query parameters. In particular, such performance advantage is shown as a speedup up to 1.23X over the quad-tree algorithm for 2D data. Results of extensive experiments run on synthetic and real datasets confirm our findings.

I. 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]–[3]. Generally, data-intensive scientific applications necessitate considerable storage space and I/O bandwidth, due to the large volume of data [4]–[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.

A. 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.

B. Motivation and Related Work

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 [9], astrophysics [10], biomedical sciences, and biophysics [11]. 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 [10], [12], [13].

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 $\langle i, j \rangle$ 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 [14], [15]. A single image generated from modern camera might contain millions of pixels. Therefore, it takes considerable 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 [16], [17].

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 $n > 2$ [18]. The n -BCFs are of interest in many forms: n -point function, n -tuple problem, nearest-neighbor classification, nonparametric outlier detection/denoising, and kernel density/classify/regression [19] are examples of statistical measurements related to n -BCF, and their applications range in various scientific fields [20]–[22].

C. 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 [23] proposed more efficient algorithms: instead of computing every point-to-point distance, the main idea is to analyze the distances between two groups of points, as described in Section II-A. These groups are represented by nodes in a space-partitioning tree structure, called *density map* (DMs), as discussed in Section II-B. 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 [23]. In this paper, we propose and evaluate an implementation based on a region kd-tree whose details will be introduced in Section II-B. In addition, the similar kd-tree implementation (data-driven) was used in [19] and [24]. Although algorithms based on both trees have the same time complexity $O(N^{\frac{2d-1}{2d}})$ where d is number of dimensions of dataset [25], a comparison of their actual execution time under different scenarios is thoroughly

studied. Thanks to the finer granularity of kd-tree in space partitioning, it is expected that the pruning ability of kd-tree gives more benefits on dual-tree algorithm. 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 all 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. Results of extensive experiments confirm such findings. We also evaluate that kd-tree yields a speedup up to 1.39X over the oct-tree in processing 3D data, since the paper limitation, we don't report them in this paper, but they can be found in [26].

D. Paper Organization

This paper is organized as followed: In Section II we sketch the dual-tree algorithm; We discuss our modeling approach and present the main analytical results in Section III; Based on the main results, we compare the performance of the two dual-tree algorithms in Section IV; We report experimental results in Section V, and conclude this paper in Section VI.

II. 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 I, 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 I: 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

A. 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. In this paper, we use 2D data to elaborate on technical details and show our theoretical and empirical work. All such work are formally extended to 3D data, for which the details are shown in a technical report [26]. 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*

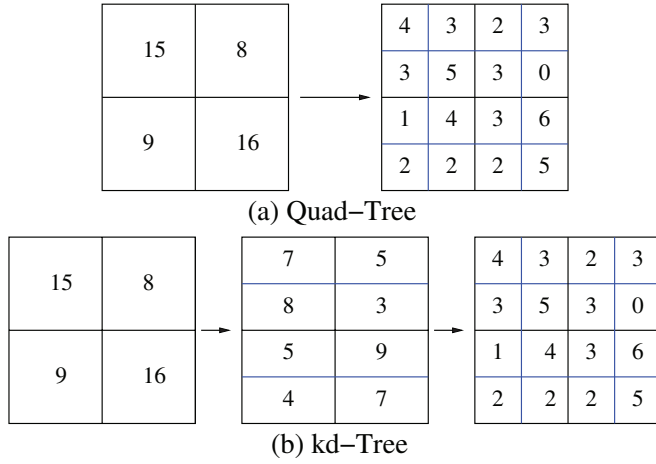


Fig. 1: A partial DM implemented by quad-tree and kd-tree. Each cell is marked by the total number of data points in it

(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 *ResolveTwoTrees* 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

```

B. 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 [23] 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 propose 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,

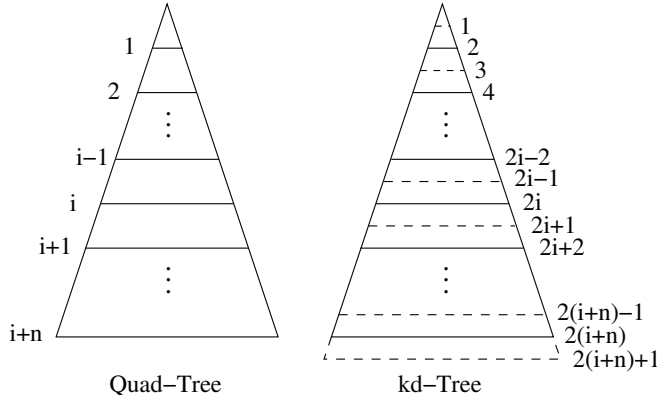


Fig. 2: Different levels on quad-tree and kd-tree. Dash line represents the intermediate level that only exists in kd-tree, and a solid line corresponds to a level that exists in both trees

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

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

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 $b^2 > k^2 C$, 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

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

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, as shown in Fig. 2.

As we shall see later (Section IV), 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.

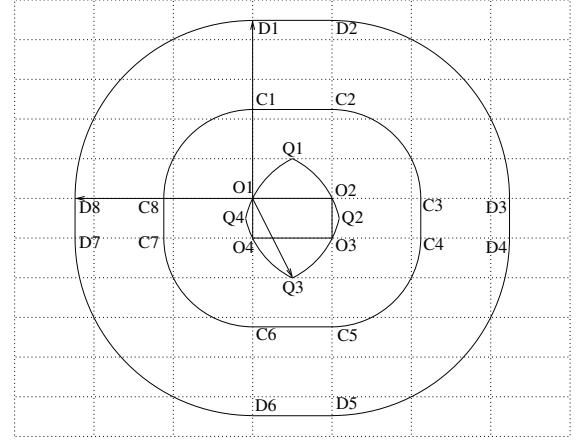


Fig. 3: Theoretical boundaries of bucket 1 and bucket 2 regions for cell A, with the bucket width $p = \sqrt{2}\delta$

III. 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.

A. The Geometric Modeling Approach

To identify the number of points are resolved, we transform the problem into a geometric modeling problem. In particular, we develop a geometric 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

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

¹Note 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 IV-A.

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 form 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. 5) 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. 5) 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.,

$$\text{non-covering factor} = 1 - \text{covering factor} \quad (3)$$

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

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

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 $O(N^{\frac{2d-1}{2d}})$, where d is number of dimensions of dataset [25]. 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.

B. 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 algo-

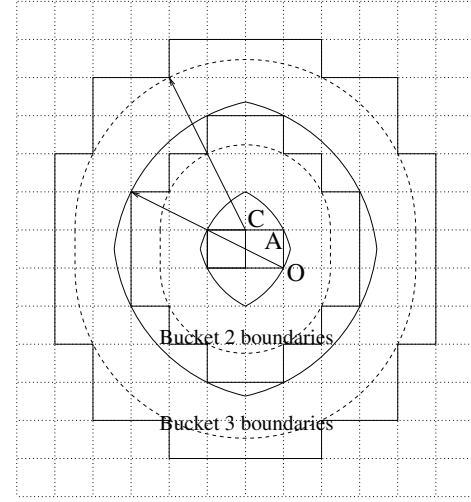


Fig. 4: Inner boundaries of the coverable region with $m = 1$

rithm 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

$$\lim_{p \rightarrow 0} \frac{\alpha_{m+1}}{\alpha_m} = \frac{3}{4} \quad (4)$$

when $i + m$ is even, and

$$\lim_{p \rightarrow 0} \frac{\alpha_{m+1}}{\alpha_m} = \frac{2}{3} \quad (5)$$

when $i + m$ is odd.

Proof. Due to page limits, we will only sketch the idea to prove Theorem 2 in the following text. The complete proof can be found in a longer version of this paper [26].

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. The bucket 2 region of A is similar to bucket 1 region but the radii of the four arcs are $2p$ – 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 $Q_1Q_2Q_3Q_4$ is shown as in Figure 4. The shape of bucket i ($i > 2$) regions is the same as bucket 2 region except the radii of

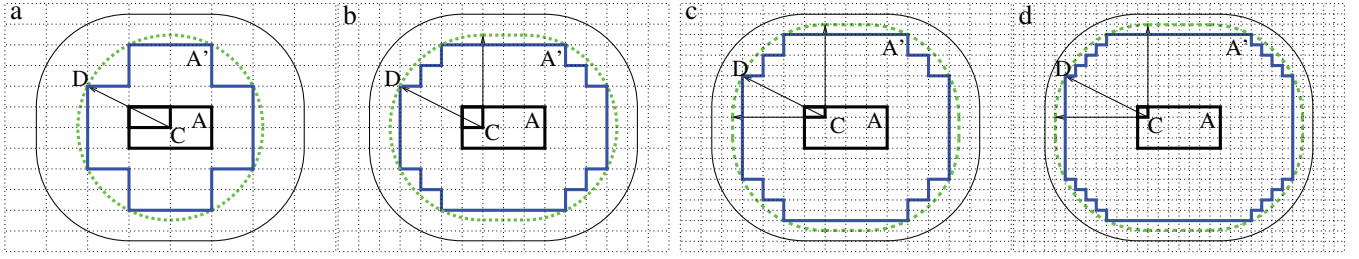


Fig. 5: Actual (solid blue line) and approximated (dashed green line) coverable region for bucket 1 under: a. $m = 2$, b. $m = 3$, c. $m = 4$, and d. $m = 5$. Outer solid black line represent the theoretical bucket 1 region. All arrowed line segments are drawn from the centers to the corresponding arcs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

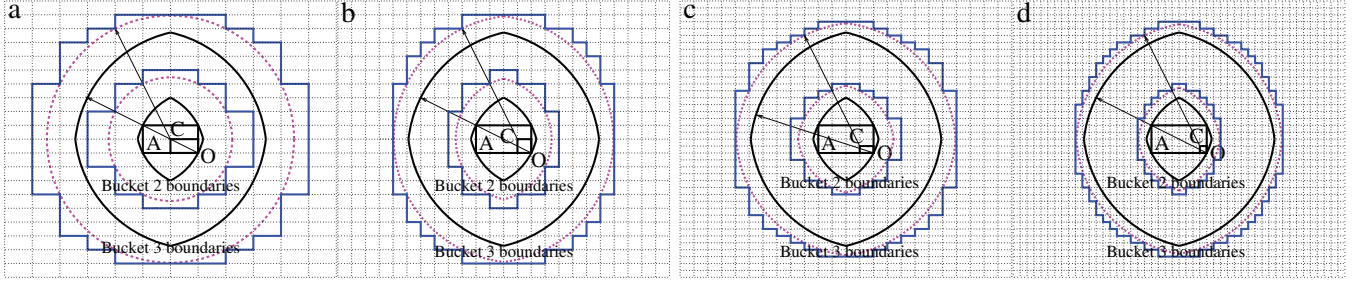


Fig. 6: Inner boundaries of the coverable regions of buckets 2 and 3 under a. $m = 2$, b. $m = 3$, c. $m = 4$, and d. $m = 5$. All arrowed line segments are of length $2p$

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., $p = \frac{\sqrt{5}\delta}{2}$. As we will see later, $p > diagonal$ will not affect our analysis. We then come up with $g(i)$ (can be found in Section 3 of [26]) to measure the area of bucket i region.

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

The First Bucket: First, let us focus on bucket 1. In Fig. 5, 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 calculated as the ratio of the area of A' to that of A .

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 for the first bucket except the radii of arcs are ip . Second, we have to consider the inner boundaries of the coverable region. Fig. 4 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. 4) 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). Fig. 6 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.

We denote the area of the coverable region A' for bucket i under different m values as $f(i, m)$ (The fully expanded formula for $f(i, m)$ can be found in Section 3 of [26]). We then use the non-covering factor $\alpha(m)$ (Eq. (6)) to study the percentage of unresolvable pairs of cell at each level.

$$\alpha(m) = 1 - c(m) = \frac{\sum_{i=1}^l [g(i) - f(i, m)]}{\sum_{i=1}^l g(i)} \quad (6)$$

To prove Theorem 2, we start by

$$\frac{\alpha(m+1)}{\alpha(m)} = \frac{\sum_{i=1}^l g(i) - \sum_{i=1}^l f(i, m+1)}{\sum_{i=1}^l g(i) - \sum_{i=1}^l f(i, m)} \quad (7)$$

By doing algebraic transformation [26], we can prove that when m is even, $\alpha(m+1)/\alpha(m)$ converges to $2/3$. 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

$$\lim_{p \rightarrow 0} \frac{\alpha(m+2)}{\alpha(m+1)} = \frac{3}{4} \quad (8)$$

The above concludes the proof of Theorem 2. \square





Start \ End		
	Starts at $2i$	Starts at $2i-1$
	Case 1	Case 3
Ends at $2(i+n)$		
	Case 2	Case 4
Ends at $2(i+n)+1$		

Fig. 7: 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

IV. 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 [25]. 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. With Theorem 2, we are able to quantitatively compare the actual running time of both algorithms under different cases (Fig. 7). 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. For rest of this section, we derived cost equations to compare the performance of two operations while the Algorithm 1 is traveling on quad-tree and kd-tree.

A. 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

$$R = I(1 + 16\alpha_0 + 16^2\alpha_0\alpha_1 + \dots + 16^n\alpha_0\alpha_1 \dots \alpha_{n-1}) \quad (9)$$

Based on Theorem 1, we have

$$R = I \left[1 + 16\alpha_0 + 16^2\alpha_0 \left(\frac{1}{2}\right) + \dots + 16^n\alpha_0 \left(\frac{1}{2}\right)^{n-1} \right] \quad (10)$$

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

$$R' = I(1 + 4\beta_0 + 4^2\beta_0\beta_1 + \dots + 4^n\beta_0\beta_1 \dots \beta_{2n-1}) \quad (11)$$

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

$$R' = I \left[1 + 4\beta_0 + 4^2\beta_0 \left(\frac{3}{4}\right) + 4^3\beta_0 \left(\frac{1}{2}\right) + \dots + 4^{2n-1}\beta_0 \left(\frac{1}{2}\right)^{n-1} + 4^{2n}\beta_0 \left(\frac{1}{2}\right)^{n-1} \left(\frac{3}{4}\right) \right] \quad (12)$$

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. (10) and Eq. (12), we have

$$\Delta_i = \frac{16^i\alpha_0\left(\frac{1}{2}\right)^{i-1}}{4^{2i-1}\beta_0\left(\frac{1}{2}\right)^{i-1} + 4^{2i}\beta_0\left(\frac{1}{2}\right)^{i-1}\left(\frac{3}{4}\right)} = \frac{\alpha_0}{\beta_0} \quad (13)$$

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 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

$$\epsilon = \left[\binom{B}{2} - \binom{B-K}{2} \right] 4^{2i-1}\beta_0 \left(\frac{1}{2}\right)^{i-1} \quad (14)$$

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

$$E[K] = B \cdot Pr\{X\} \quad (15)$$

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 $B \rightarrow N$) do we see a non-negligible $Pr\{X\}$. However, under skewed data distribution (e.g., Zipf), $Pr\{X\}$ becomes critically large even at higher levels of the tree, leading to a bigger discount ϵ .

B. 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

$$Speedup = \frac{JC_1}{\frac{3}{4}JC_1 + PC_2} \quad (16)$$

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 II-B), we have $16C_1 = C_2$. Consequently, the denominator of Eq. (16) becomes

$$\frac{3}{4}JC_1 + 16PC_1 \quad (17)$$

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 .

$$\frac{J}{4} = x^2 P \Rightarrow \frac{J}{4x^2} = P \quad (18)$$

By plugging Eq. (17) and Eq. (18) into Eq. (16), we have

$$Speedup = \frac{1}{\frac{3}{4} + \frac{4}{x^2}} \quad (19)$$

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.

C. 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. (10) remains the same and the only change to Eq. (12) 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. (12), which is the one with the lowest order in the series. In other words, Case 3 is almost the same scenario as Case 1.

D. 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. (14) and Eq. (16).

V. 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 size (i.e., 100,000 to 1,600,000 points).

We first evaluate our analysis related to Case 1 of 2D data. Fig. 8a 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 III. Note that such errors disappear at $m = 3$ in Fig. 8a. For the Zipf data, we see Δ_i values greater than 1 for larger i - this is due to the fact that

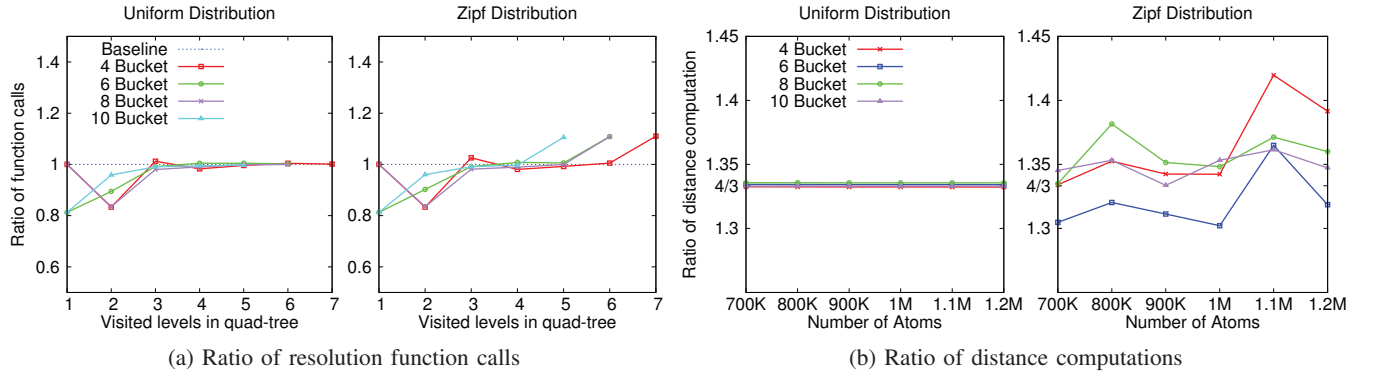


Fig. 8: Ratios of (a) Type I operations and (b) Type II operations made by quad-tree vs. that by the kd-tree under different histogram bucket numbers and data distribution patterns (i.e., uniform and Zipf)

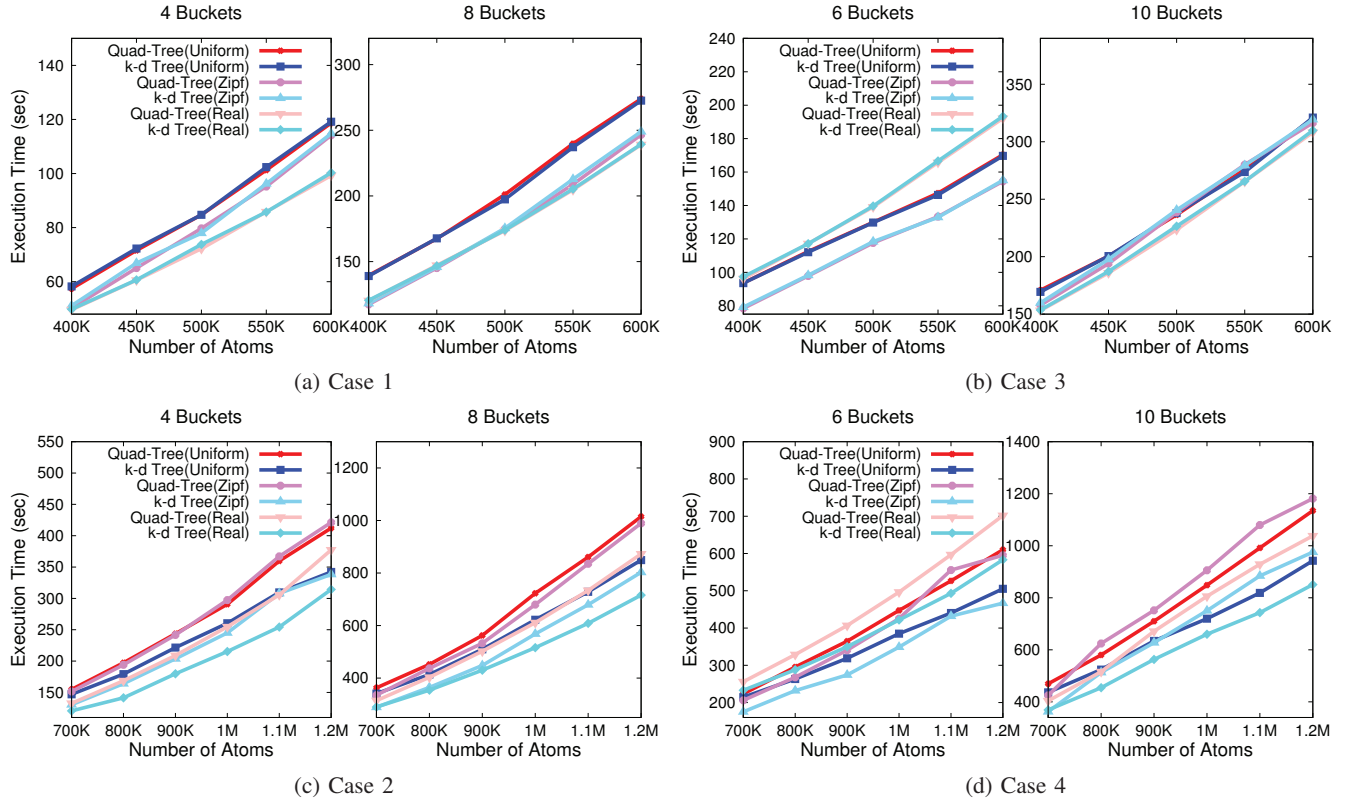


Fig. 9: Running time of the dual-tree algorithm in 2D systems under different data sizes and data distribution patterns

empty nodes are found earlier in kd-tree. Such results confirm our analysis shown in Section IV-A.

Related to Case 2, Fig. 8b shows the ratio of total number of distance computations 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 (e.g., Zipf), we observe more fluctuations in the results, and the speedup is even higher than those in uniform data for most of the cases. This result confirms the analysis shown in Eq. (18).

Fig. 9 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 II. Recall that the number of buckets in the histogram (or the value of 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 the four cases discussed in Section IV. In Case 1 (Fig. 9a) and Case 3 (Fig. 9b), the performance of the two trees is almost identical, confirming our findings in Sections IV-A and IV-C. In Case 2 (Fig. 9c) and

TABLE II: Ranges of speedup (kd-tree over quad-tree) observed in all cases of 2D experiments shown in Fig. 9

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

Case 4 (Fig. 9d), however, the performance gap between the two trees becomes much larger. 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 IV-B is an accurate estimation. As a result, the kd-tree is still the obvious winner in performance

We also recorded the total running time of Algorithm 1 under the oct-tree-based and kd-tree-based implementations for 3D data, due to space limit, we put them in Appendix C of [26]. In summary, the kd-tree outperforms oct-tree in all experimental runs we conducted, and the speedup in all cases are within the range suggested by our analysis.

VI. 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 under all scenarios 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.

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, "A scalable dbms for large scientific simulations," *IEEE DANTE Proceedings*, pp. 271 – 275, 1999.
- [2] X. Fei and S. Lu, "A collectional data model for scientific workflow composition," *IEEE International Conference on ICWS*, pp. 567 – 574, 2010.
- [3] D. E. Shaw, M. M. Deneroff, R. O. Dror, J. S. Kuskin, and et al., "Anton, a special-purpose machine for molecular dynamics simulation," *Communications of the ACM*, vol. 51, pp. 91–97, 2008.
- [4] D. Howe, M. Costanzo, P. Fey, T. Gojobori, and et al., "Big data: The future of biocuration," *Nature*, vol. 455, pp. 47–50, 2008.
- [5] B. A. Huberman, "Sociology of science: Big data deserve a bigger audience," *NATURE*, vol. 482, p. 308, 2012.
- [6] D. Centola, "The spread of behavior in an online social network experiment," *Science*, vol. 329, no. 5996, pp. 1194–1197, 2010.
- [7] S. Lakshminarasimhan, J. Jenkins, I. a. Z. G. Arkatkar, H. Kolla, S.-H. Ku, S. Ethier, J. Chen, C. Chang, S. Klasky, R. Latham, R. Ross, and N. Samatova, "Isabela-qa: Query-driven analytics with isabela-compressed extreme-scale scientific data," *IEEE International Conference for High Performance Computing*, no. 1-11, 2011.
- [8] M. Weidner, J. Dees, and P. Sanders, "Fast olap query execution in main memory on large data in a cluster," *IEEE International Conference on Big Data*, pp. 518 – 524, 2013.
- [9] S. Klasky, B. Ludaescher, and M. Parashar, "The center for plasma edge simulation workflow requirements," *IEEE 22nd International Conference on Data Engineering Workshops*, p. 73, 2006.
- [10] J.-L. Starck and F. Murtagh, "Astronomical image and data analysis," *Springer*, 2002.
- [11] M. P. Allen and D. J. Tildesley, "Computer simulations of liquids. clarendon press," *Clarendon Press, Oxford*, 1987.
- [12] A. Filipponi, "The radial distribution function probed by x-ray absorption spectroscopy," *Phys.: Condens. Matter*, vol. 6, no. 8415–8427, 1994.
- [13] V. Springel, S. D. M. White, A. Jenkins, C. S. Frenk, N. Yoshida, L. Gao, J. Navarro, R. Thacker, D. Croton, J. Helly, J. A. Peacock, S. Cole, P. Thomas, H. Couchman, A. Evrard, J. Colberg, and F. Pearce, "Simulations of the formation, evolution and clustering of galaxies and quasars," *Nature*, vol. 435, pp. 629–636, June 2005.
- [14] J. Huang, S. R. Kumary, M. Mitra, W.-J. Zhu, and R. Zabih, "Image indexing using color correlograms," *IEEE Conf on Comp Vision and Pattern Recognition*, pp. 762 – 768, Jun 1997.
- [15] G. Heidemann, "Combining spatial and colour information for content based image retrieval," *Computer Vision and Image Understanding*, vol. 94(1), no. S1077314203001899, pp. 234–270, 2004.
- [16] M. Ankerst, G. Kastenmüller, H.-P. Kriegel, and T. Seidl, "3d shape histograms for similarity search and classification in spatial databases," *Proc. 6th Int. Symposium on Spatial Databases*, pp. 207–226, 1999.
- [17] G. V. and G. O., "Multidimensional access methods," *ACM Computing Surveys*, vol. 30, no. 2 (1998) 170–231, 1998.
- [18] A. Moore, A. Connolly, C. Genovese, A. Gray, L. Grone, N. K. II, R. Nichol, J. Schneider, A. Szalay, I. Szapudi, and L. Wasserman, "Fast algorithms and efficient statistics: N-point correlation functions," *Springer, Heidelberg*, pp. 71–82, 2006.
- [19] A. Gray and A. Moore, "N-body problems in statistical learning," in *Advances in Neural Information Processing Systems* (T. K. Leen and T. G. Dietterich, eds.), 2001.
- [20] J. Tsang, "Evolving trajectories of the n-body problem," *IEEE Congress on CEC*, pp. 3726 – 3733, 2008.
- [21] K. Tsoi, C. Ho, H. Yeung, and P. Leong, "An arithmetic library and its application to the n-body problem," *12th Annual IEEE Symposium on FCCM*, pp. 68–78, 2005.
- [22] L. Perrone and D. Nicol, "Using n-body algorithms for interference computation in wireless cellular simulations," *8th International Symposium on Telecommunication Systems*, pp. 49 – 56, 2008.
- [23] Y.-C. Tu, S. Chen, and S. Pandit, "Computing distance histograms efficiently in scientific databases," *ICDE*, pp. 796–807, 2009.
- [24] A. Moore, A. Connolly, C. Genovese, A. Gray, L. Grone, N. I. Kanidoris, R. Nichol, J. Schneider, A. Szalay, I. Szapudi, and L. Wasserman, "N-point correlation functions," *ESO Astrophysics Symposia, Springer*, vol. 2001, p. 7182, 2006.
- [25] S. Chen, Y.-C. Tu, and Y. Xia, "Performance analysis of a dual-tree algorithm for computing spatial distance histograms," *VLDB Journal*, vol. 20, no. 4, pp. 471–494, 2011.
- [26] C.-C. Mou, S. Chen, and Y.-C. Tu, "A comparative study of dual-tree algorithms for computing spatial distance histograms," *Technique Report, Dept. of Computer Science and Engineering, USF, Tampa, Florida, USA*, vol. 2016, no. 172, 2016.