Distance Histogram Computation Based on Spatiotemporal Uniformity in Scientific Data

Anand Kumar [§], Vladimir Grupcev [§], Yongke Yuan [*], Yi-Cheng Tu [§], and Gang Shen [†]

[§]Department of Computer Science and Engineering, University of South Florida
4202 E. Fowler Ave., ENB 118, Tampa, FL 33620, USA

[*]School of Economics and Management, Beijing University of Technology
100 Pingleyuan, Chaoyang District, Beijing 100124, China

[†]Department of Statistics, North Dakota State University
201H Waldron Hall, NDSU Dept. 2770, P. O. Box 6050, Fargo, ND 58108, USA

[§]{akumar8, vgrupcev, ytu}@cse.usf.edu, [*]colin.yyuan@gmail.com,
[†]gang.shen@ndsu.edu

ABSTRACT

Large data generated by scientific applications imposes challenges in storage and efficient query processing. Many queries against scientific data are analytical in nature and require super-linear computation time using straightforward methods. Spatial distance histogram (SDH) is one of the basic queries to analyze the molecular simulation (MS) data, and it takes quadratic time to compute using brute-force approach. Often, an SDH query is executed continuously to analyze the simulation system over a period of time. This adds to the total time required to compute SDH. In this paper, we propose an approximate algorithm to compute SDH efficiently over consecutive time periods. In our approach, data is organized into a Quad-tree based data structure. The spatial locality of the particles (at given time) in each node of the tree is acquired to determine the particle distribution. Similarly, the temporal locality of particles (between consecutive time periods) in each node is also acquired. The spatial distribution and temporal locality are utilized to compute the approximate SDH at every time instant. The performance is boosted by storing and updating the spatial distribution information over time. The efficiency and accuracy of the proposed algorithm is supported by mathematical analysis and results of extensive experiments using biological data generated from real MS studies.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database applications—*Scientific databases*

[*]Work was done when Yuan was a visiting professor at the University of South Florida.

General Terms

Algorithms, Performance, Experimentation

Keywords

Scientific data, spatial distance histogram, quad-tree, density map, spatiotemporal locality

1. INTRODUCTION

With the improvement of computer simulation systems and experimental devices, the amount of data generated by scientific applications has posed a big challenge to the design of data management software. There have been attempts to manage such large data using traditional database management systems (DBMS) [11, 8, 19]. However, the traditional DBMSs are optimized towards business applications and hence require us to have another look at the system design. The access to scientific datasets is mainly through high-level analytical queries that are much more complex to compute than simple aggregates. Many scientific queries comprise of a small number of analytical routines that are frequently used, and these routines generally take super-linear time when computed by a brute-force approach. Therefore, there is a need to support efficient processing of such analytics in a scientific database system. In this paper, we present our work related to such an analytics that is extremely important in the analysis of *molecular simulation* (MS) data.

Molecular (or particle) simulations are computer based simulations of complex biological, chemical, or physical structures. The basic entities of such simulations are natural particles (e.g., atoms, molecules, etc.) that interact with each other following classical forces. The simulations are performed to study the behavior of the entities under the experimental settings based on some basic theoretical model. MS is widely used as a basic research tool in bio-physics, astronomical physics, biomedical sciences, material sciences, etc. The number of entities in simulations may be of large magnitude, often in the range of hundreds of thousands to millions. Quantities measured during the simulation are analyzed to test the theoretical model [9, 18]. Fig. 1 shows a snapshot of a collagen fiber, which consists of 890,000 atoms. Furthermore, the dataset contains many snapshots (called *frames*) of the system state captured at different time instants. Every frame contains all particles with their measurements such as spatial coordinates, velocity, forces, charge, and mass. A large number (e.g., tens of thousands) of such frames are stored during a typical simulation process.

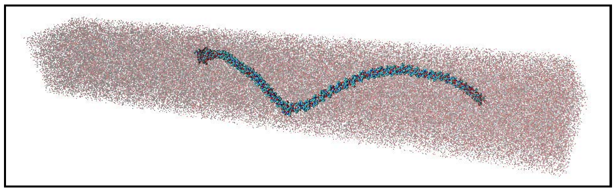


Figure 1: A snapshot of simulated collagen fiber structure

MS data analysis, on which scientific discovery heavily depends, involves computing complex quantities that show statistical properties of the data. Many queries involve counting more than one particle as a basic data unit – a function that involves counting all m-tuple subsets of the data is called a m-body correlation function. In this paper, we discuss one such analytical query called *spatial distance histogram* (SDH) [25], which is the histogram of distances between all pairs of particles in the system. Being a discrete approximation of the continuous probability distribution of distances named Radial Distribution Function (RDF), the SDH is a very important query in MS databases. It is one of the basic building blocks for a series of critical quantities required to describe the physical systems such as total pressure and energy as summarized in [9].

1.1 Problem Statement

The basic problem of SDH computation is defined as follows: Given the coordinates of N points and a user-specified distance w, we are required to compute the number of point-to-point distances that fall into a series of ranges (i.e., buckets) of width w: $[0, w), [w, 2w), \ldots, [(l-1)w, lw]$. In other words, the SDH computation gives an ordered list of non-negative integers $H = (h_1, h_2, \ldots, h_l)$, where each $h_i (0 < i \leq l)$ is the number of distances that fall into the bucket (distance range) $[(i-1)w, iw)$. As SDH is a basic tool in the validation and analysis of MS systems, its variations over a period of time during the simulation often become critical information. Therefore, we need an efficient technique to compute the SDH not only for a single frame, but also over a large number of consecutive frames.

2. COMPARISON TO RELATED WORK

The naive way of computing SDH involves calculating distance between every pair of particles in the system, and putting the distances into appropriate histogram buckets - this apparently requires quadratic time. Popular simulation data analysis softwares such as GROMACS [14] still follow the brute-force way to compute SDH. The current state-of-the-art techniques in SDH computation follow the strategy of treating clusters of particles as the basic processing units [25, 10]. Such clusters are often organized into nodes in space-partitioning trees such as *kd*-trees, as reported in [10]. The key idea in such methods is to process all the particles in a tree node as a whole, rather than building the histogram by processing particle-to-particle distances.

A Density-Map based SDH algorithm (DM-SDH) using a quad-tree data structure is presented in [25]. It is proved that DM-SDH runs at $\Theta(N^{\frac{3}{2}})$ for 2D data and $\Theta(N^{\frac{5}{3}})$ for 3D data. The central idea of DM-SDH will be presented later in more details. Even though the DM-SDH is an important cornerstones in tackling the problem of SDH processing, it is still not a *practically* efficient solution for the following reasons. First, the time complexity analysis [25] of DM-SDH is done taking the data size N as input under a given bucket width w of the SDH. Second, if we consider the running time as a function of w, or more conveniently, of the total number of buckets l, we can easily see that it increases dramatically with l. As a result, in many cases of reasonably small w values, the actual running time of the algorithm can be even longer than that of the brute-force algorithm [25]! Third, it focuses on computing SDH for a single frame while almost all MS data analysis tasks in the scientific community require SDH computation for many (if not all) consecutive frames. To accomplish such continuous SDH processing in F frames, we basically have to run the same algorithm F times, and F is normally on the order of tens of thousands.

An approximate SDH algorithm (ADM-SDH), whose running time is not dependent on the data size N (excluding the DM-tree construction time), is also introduced in [25]. However, its time is dependent on a guaranteed error bound and the bucket size w. Similar to DM-SDH, the ADM-SDH can only be applied to a single frame. Hence, the main objective of our work is to design an algorithm to remedy the problems of both DM-SDH and ADM-SDH algorithms while achieving higher efficiency and accuracy.

The SDH problem is often confused with the computation of the force/potential fields in the MS process itself. In the later, the physical properties of a particle in the simulation system are determined by the forces applied to it by all other particles in the system. Various approximate algorithms [3, 12] have been proposed to take advantage of the mathematical features of the formula that defines such forces. Although the problem is similar to SDH in definition and has time complexity of brute-force method, its algorithms provide little insights on how SDH can be efficiently computed. A detailed comparison between force field computation and SDH can be found in [6]. Note that the former is about computation done for simulation of a system while the latter is for system analysis.

The computation of SDH over multiple frames is related to persistent data structures [7]. These allow different versions of the computation results, maintained over time, that are updated for quick query processing. A multi-dimensional persistent tree (or MP-tree) [23] is an extension of such data structure for searching in spatio-temporal data. The location based services (e.g. GIS application) have motivated the need for efficient handling of spatio-temporal data in DBMSs. Building persistent index schemes on complex spatio-temporal data allows time efficient retrieval [17]. A detailed survey by Kaplan [15] presents applications in which persistent data structures play significant role in improving efficiency. These structures are basically designed to address the I/O bottleneck problem. Again, multi-frame SDH computation can hardly take advantage of techniques developed for persistent data structures, since heavy computation has to be performed at each instant, which overshadows the I/O time.

3. OVERVIEW AND CONTRIBUTIONS

In this paper, we present the design and evaluation of a **practical** algorithm for processing SDH of large-scale MS data, to address the issues mentioned in previous section. The main idea is to process a density map of the given data by taking advantage of the two types of uniformity that widely exist in MS data. This saves computational time while achieving high accuracy in query results.

The first type of uniformity is related to the **spatial distribution of data points** (e.g., atoms) in MS datasets. In short, we found that MS data often consists of spatial regions in the simulation space where the data points are uniformly distributed. It is well known that components of natural systems tend to spread out evenly in space due to the existence of chemical bonds and/or inter-particle forces [1, 2]. As a result, particles are often found to be uniformly distributed in localized regions in the simulation space.[1] For example, the solvent molecules represented by red dots in Fig. 1 are uniformly distributed since a body of water is essentially incompressible. Due to this uniformity, such a region can be treated as a single entity in computing SDH without introducing significant errors. This property is the key in controlling the running time of the algorithm such that it is not affected by the bucket width of SDH. More details of this technique are presented in section 5.1.

The second type of uniformity refers to the significant **temporal similarity among neighboring frames**. We argue such similarity is actually reflected in the final results of the SDH we obtain for the neighboring frames. Therefore, given two frames f_0 and f_1, if we know the SDH of f_0, the SDH of f_1 can be obtained by working on the regions that do not show similarity between the two frames while the regions that are similar can be ignored. To harness such similarities among frames, we design an **incremental algorithm** that quickly computes SDH of a frame from a base frame whose SDH is obtained using traditional single-frame algorithms. More details on this are presented in section 5.2.

We test a composite algorithm that combines the above ideas on real MS datasets. The results clearly show that the running time of the proposed algorithm is insensitive to the change of SDH bucket width w, and can be orders of magnitude shorter than that of the ADM-SDH. Furthermore, the accuracy of the algorithm also beats DM-SDH in almost all cases. We believe, we have developed the first practical fast algorithm for processing SDHs, and our success will also open up a new direction in tackling the more general and also difficult problem of multi-body (m-body) correlation function computation [21]. In summary, the major contributions of our work are:

[1] However, it will not make the data system-wise uniform. Otherwise, the SDH computation becomes a trivial task.

- Technique to identify spatial uniformity within a frame
- Technique to identify temporal locality between consecutive frames
- An approximate algorithm to compute the SDH of large number of data frames by utilizing the above properties
- Empirical and analytical evaluation of the algorithm and comparison with existing technique

4. BACKGROUND

In this section, we introduce the basic techniques and concepts applied in our work, as they serve as the foundations in this paper. A detailed presentation is available in [25]. A *density map* is a conceptual data structure that divides the simulation space into a grid of small cells (or regions) of equal size. The region is a cube in 3D and square in 2D.[2] Each cell of the grid is further divided into four equal sized cells to generate a density map of higher resolution. Therefore, we can organize different density maps of the same data using a region quad-tree [20], in which a cell is represented by a tree node and a density map is basically the collection of all nodes on **one level** of the tree. In every node of the tree, we also record the number of particles in it, and the cell location (i.e., coordinates of corner points). We call such a tree the Density-Map Tree (DM-tree).

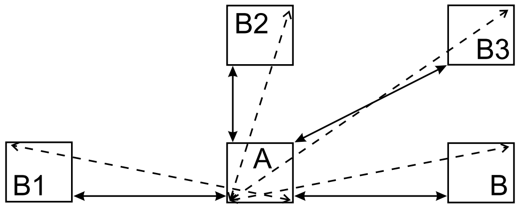


Figure 2: Computing minimum (i.e., length of solid lines) and maximum distance (i.e., length of dashed lines) range between two cells

The core of DM-SDH algorithm is a procedure called RESOLVETWOCELLS, which takes two cells (e.g., A and B in Fig. 2) from a density map as input. For any two cells, we can compute the minimum and maximum distance between them in constant time. A pair of cells is *resolvable* if the minimum and maximum distances between them fall into the same SDH bucket i. If two cells resolve into bucket i, we simply update the histogram by incrementing the distance count of bucket i by $n_A n_B$, where n_A and n_B are the number of particles in cell A and B, respectively. If the cells do not resolve, we take one of the following actions:

(1) Go to the next density map with higher resolution and resolve all children of A with those of B, or

(2) If it is the leaf-level density map, compute every distance between particles of A and B and update the histogram accordingly.

[2] We continue our discussions focusing on 2D data to elaborate and illustrate the proposed ideas. The extension of such ideas to 3D space would be straightforward.

The complete SDH is generated by executing the ResolveTwoCells routine for all cell pairs on a given density map DM_k (the density map in which the cell diagonal length is at most equal to the bucket width w). Therefore, the algorithm calls ResolveTwoCells recursively (i.e., action (1) above) till it reaches the lowest level of the tree (i.e., action (2) above). Using a geometric modeling approach, it has been proven that the time complexity of DM-SDH is $O\big(N^{\frac{2d-1}{d}}\big)$ [6], where d is the number of dimensions. The more exciting aspect is that DM-SDH can be extended to compute approximate SDH results with time complexity $I\big(\frac{1}{\epsilon}\big)^{2d-1}$, where ϵ is an error bound and I is the number of pairs of cells in DM_k.[3] The idea is to make recursive calls to ResolveTwoCells only for a fixed number (m) of levels.[4] For cell pairs that are still not resolvable after visiting the m levels, we use heuristics to greedily distribute distances into relevant SDH buckets. While the ADM-SDH is a fast algorithm with respect to data size N, its running time is very sensitive to the bucket width w. Specifically, *quantity I increases by a factor of 2^{2d} when $1/w$ doubles.* As the SDH is basically a discrete approximation of a continuous distribution of the distances in the simulation system, more information is lost when w increases. In practice, scientists prefer a w value that is small enough such that there are a few hundred buckets in the SDH computed. In this paper, we present a fast multi-frame SDH processing algorithm whose behavior is insensitive to both w and N. Our algorithm will utilize the same region quad-tree for data organization as in the DM-SDH and ADM-SDH algorithms.

5. SDH COMPUTATION BASED ON SPATIOTEMPORAL UNIFORMITY

The DM based algorithms depend heavily on resolving cells to achieve the desired accuracy. Only when we finish visiting m levels of the tree or reach the leaf nodes do we use heuristics to distribute the distances into relevant buckets. That is the main reason for the long running time. Our idea to remedy that problem is to greedily distribute distances between very large regions of the simulation space, even when no pairs of such regions are resolvable. In other words, we *use heuristics for distance distribution as early as possible.* However, the distribution of distances between two large regions may yield arbitrarily large errors. Therefore, the key challenge is to design a heuristic with high accuracy even under large regions.

5.1 Utilizing Spatial Uniformity

Our first idea to address the aforementioned challenge is to take advantage of the spatial distribution of data points in the cells. As illustrated in Fig. 3: two cells have a distance range $[u, v]$ that overlaps with three SDH buckets (i.e., from bucket i to $i+2$). A critical observation here is: if we knew the probability distribution function (PDF) of the point-to-point distances between cells A and B, we can effectively distribute the actual number of distances $n_A n_B$ into the three overlapping SDH buckets. Specifically, the total number of $n_A n_B$ distances will be assigned to the buckets based on the

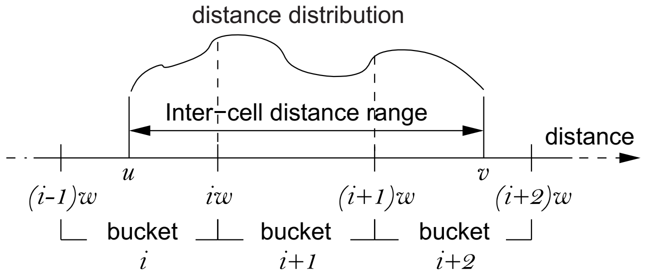


Figure 3: Distance range of non-resolvable cells overlaps with more than one bucket of the SDH

probability of a distance falling into each bucket according to the PDF. For the case in Fig. 3, the relevant SDH buckets and the number of distances assigned to them are as follows:

$$H[i], \quad n_A n_B \int_u^{iw} g(t)dt \tag{1}$$

$$H[i+1], \quad n_A n_B \int_{iw}^{(i+1)w} g(t)dt \tag{2}$$

$$H[i+2], \quad n_A n_B \int_{(i+1)w}^{v} g(t)dt \tag{3}$$

where g is the PDF. The biggest advantage of the above approach is that the errors generated in each distance count assignment operation can be very low, and the errors will not be affected by the bucket width w, as long as the PDF is an accurate description of the underlying distance distribution [5]. Therefore, the main task of the proposed approach is to derive the PDF.

Note that in the work presented in [25], the distances are proportionally distributed into the three buckets based on the overlaps between range $[u, v]$ and the individual buckets. Such a primitive heuristic, which is named Prop (short for "proportional"), implicitly assumes that the distance distribution is uniform within $[u, v]$. However, our experiments show that a typical distance distribution in MS data is far from being uniform. Hence, our proposed solution will naturally introduce less errors than the Prop heuristics adopted by ADM-SDH.

In general, the PDF of interest can be obtained by the spatial distribution of particles in the two relevant cells. The coordinates of any two particles - one from A and the other from B - can be modeled as two random vectors $\vec{v}_A$ and $\vec{v}_B$, respectively. The distance between these two particles can also be modeled as a random variable D, and we have

$$D = ||\vec{v}_A - \vec{v}_B||. \tag{4}$$

Given that, if we know the PDFs of both $\vec{v}_A$ and $\vec{v}_B$, the PDF of D can be derived via one of the following strategies: (1) generation of a closed-form via analyzing the PDFs of $\vec{v}_A$ and $\vec{v}_B$ as well as Eq. 4; or (2) Monte Carlo simulations using the PDFs of $\vec{v}_A$ and $\vec{v}_B$ as data generation functions. In practice, it is difficult to get a closed-form PDF for D even when the particle spatial distributions follow a simple form.[5] Therefore, we focus on the second technique of running Monte Carlo simulations in this paper.

[3] There is also a $O(N \log N)$ cost for building the Quad-tree.

[4] Given the user specified error bound ϵ, our analytical model can tell what value of m to choose [25]. We will study the heuristics for distance distribution in section 5.1.

[5] For example, our analytical study (Appendix F in [26]) under the assumption of uniform particle distribution shows that D^2 can only be approximated by a non-central chi-square distribution.

Monte Carlo simulations can help us obtain a discrete form of the PDF of the distance distribution, given the PDFs of the particle spatial distributions [18]. One important note here is that *the method works no matter what forms the spatial distributions follow*. However, to generate the particle spatial distributions, it is infeasible to test the MS dataset for all possible distributions. Instead, we focus on testing if the data follows the most popular distribution in MS - *spatial uniform distribution*.[6] As discussed in section 3, natural systems often contain localized uniform regions in simulation space. Given that, our proposed algorithm contains the following steps:

(1) Identifying **uniform regions** in which particles are uniformly distributed;

(2) Deriving the distance distribution PDFs between all pairs of uniform regions by Monte Carlo simulations;

(3) Assigning the actual distance counts in such regions following Eqs. 1– 3.

One technical detail skipped is that we also need to assign intra-cell distances (step(2) only handles inter-cell distances) to the first few buckets of the SDH. Details of such operations can be found in Appendix A.

To complete the above algorithm, all pairs of cells containing at least one non-uniform cell, will be processed using the Prop heuristic. Physical study of molecular systems have shown that it is normal to see a small number of large uniform regions covering most of the particles, leaving only a small fraction of particles in non-uniform regions [1, 2]. This is also verified by our experiments using real MS datasets (section 7). This translates into high efficiency of the proposed algorithm. Furthermore, the time complexity of steps (1) and (2) is unrelated to the bucket size w. The time needed for step (3) is inversely related to w.

In the remainder of this subsection, we present the main technical details of accomplishing the above steps.

5.1.1 Identification of uniform regions

The first problem related to this topic is: given a spatial region (represented as a quad-tree node), how do we test if it is a uniform region? We take advantage of the chi-square (χ^2) goodness-of-fit test to solve this problem. Here we show how the χ^2 test is formulated and implemented in our model. A brief introduction to this statistical tool and justification of its applicability to our problem can be found in Appendix III of technical report [16].

Definition 5.1. *Given a cell Q (i.e., a tree node) in the DM-tree, we say Q is uniform if its probability value p in the chi-square goodness-of-fit test against uniform distribution is greater than a predefined bound α.*

To obtain the p-value of a cell, we first need to compute two values: the χ^2 value and the degree of freedom (df) of that particular cell. Suppose cell Q resides in level k of the DM-tree (see Fig. 4). We go down the DM-tree from Q till we reach the leaf level, and define each leaf-level descendant of Q as a separate **category**. The intuition behind the test here is: *Q is uniform if each category contains roughly the same number of particles.* The number of such leaf-level descendants of cell Q is 4^{t-k}, where t is the leaf level number. Therefore, the df becomes $4^{t-k}-1$. The observed value, O_j, of a category j is the actual particle count in that leaf cell. The expected value, E_j, of a category is computed as follows:

$$E_j = \frac{Total\ Particle\ Count\ in\ Cell\ Q}{\#\ of\ leaf\ level\ descendants\ of\ Q} = \frac{n_Q}{4^{t-k}} \quad (5)$$

Having computed the observed and expected values of all categories related to Q, we obtain the χ^2 test score of cell Q through the following equation:

$$\chi^2 = \sum_{j=1}^{4^{t-k}} \frac{(O_j - E_j)^2}{E_j} \quad (6)$$

Next, we feed these two values, the χ^2 and the df, to the R statistical library [24], which computes the p-value. We then compare the p-value to a predefined probability bound α (e.g., 0.05). If $p > \alpha$, we mark the cell Q as uniform, otherwise we mark it as non-uniform. Note that the χ^2 test performs poorly when the particle counts in the cells drop bellow 5 [13]. But, we already had similar constraint in our algorithm while building the DM-tree, essentially making the cells in the leaf level contain more than 4 particles. Hence, we choose leaf level nodes as the categories in the test.

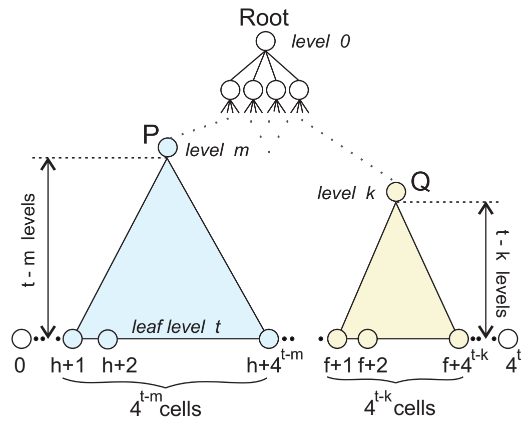


Figure 4: Sub-trees of nodes P and Q with their leaf nodes

To find all the uniform regions, we traverse the DM-tree starting from the root and perform the above χ^2 test for each node we visit. However, once a node is marked uniform, there is no need to visit its subtree. The pseudo code shown in Fig. 5 represents this idea – to find all uniform regions, we only need to call procedure MarkTree with the root node of the DM-tree as input.

5.1.2 Monte Carlo simulations

The distribution of distances between a pair of cells, say A and B, can be determined based on their spatial distribution of particles, by running Monte Carlo simulations. Monte Carlo simulation is a way to model a phenomenon

[6] Particle spatial distribution is different from the distribution of distances between particles.

```
Procedure MarkTree(node $Q$, level $l$)

0  checkUniform($Q, l$)
1  if $Q$ is NOT uniform
2  then for each child $B_i$ of cell $Q$: $i := 1 \ldots 4$
3          MarkTree ($B_i$, $l+1$)
```

```
Procedure checkUniform(node $Q$, level $l$)

0  Go to leftmost leaf level ($t$) descendent of $Q$
1  for $k = 1$ to $4^{t-l}$
2      $\chi^2 := \chi^2 + \frac{(O_k - E_k)^2}{E_k}$
3  Get $pval(\chi^2)$ using $R$ library
4  if $pval >$ significance value $\alpha$
5  then mark $Q$ as uniform
6  else mark $Q$ as not uniform
```

Figure 5: Marking Uniform Regions

that has inherent uncertainty [18]. If the spatial distributions of particles in A and B are known to be uniform, the simulations can be done by sampling (say n_s) points independently at random from uniform distributions within the spatial ranges of A and B. Then, the distance distribution is computed from the points sampled in both cells. A *temporary* distance histogram can be built for this purpose. All n_s^2 distances are computed (brute-force method), and put into buckets of the temporary histogram accordingly (e.g., those overlapping with $[u, v]$ in Fig. 3). The final proportions of each bucket in the temporary histogram will fulfill our needs in step (3) of the algorithm.

Sufficient number of points are needed to get reasonably high accuracy of the SDH generated [5]. The cost of running such simulations can be high if we were to perform one simulation for each pair of uniform regions. This, fortunately, is not the case. First, let us emphasize that the simulations are not related to the number of particles (e.g., n_A and n_B) in the cells of interest - the purpose is to approximate the PDF of distance distribution. Second, and most importantly, the same simulation can be used for multiple pairs of cells in the same density map, as long as the two cells in such pairs have the same relative position in space. A simple example is shown in Fig. 2: cell pairs (A, B) and (A, B_1) will map to the same range $[u, v]$ and can definitely use the same PDF. A systematic analysis of such sharing is presented in following theorem.

Theorem 5.2. *The number of distinct Monte Carlo simulations performed for pairs of cells in a density map of M cells, is $O(M)$.*

Proof. See appendix B. □

Theorem 5.2 says that, for the possible $O(M^2)$ pairs of uniform regions on a density map, there are only a linear number of simulations that need to be run. Furthermore, as we will see later (Section 5.2), the same cells exist in all the frames of the dataset, therefore, a simulation run for one frame can be shared among all frames. Given the above facts, we can create a hash table or lookup table to store the simulation results to be shared among different operations when a PDF is required.

5.2 Utilizing Temporal Locality

Another inherent property of the MS is that the particles often exhibit temporal locality, and such temporal property can be utilized to compute the SDH of consecutiv frames even faster. The existence of temporal locality is mainly due to the physical properties of the particles in most of the simulation systems. More specifically, such properties can be observed at the following two levels:

(1) Particles often interact with each other in groups and move randomly in a very small subregion of the system;

(2) With particles moving in and out of a cell, the total number of particles in that cell does not change much

We discuss the algorithm in terms of only two frames f_0 and f_1, although the idea can be extended to an arbitrary number of frames. DM-trees T_0 and T_1 are built for the two frames f_0 and f_1, respectively. Since the DM-trees are built independently from the data they hold, the number of levels and cells, as well as the dimensions of corresponding cells in both DM-trees will be same. First, an existing algorithm (e.g., DM-SDH or ADM-SDH) is used to compute the SDH H_0 for the *base frame* f_0. Then we copy the SDH of frame f_0 to that of f_1, i.e., $H_1 = H_0$. The idea is to modify the initial value of H_1 to reach its correct form by *only processing cells that do not show temporal locality.*

Let DM_k^0 and DM_k^1 be the density maps, at level k, in their respective DM-trees T_0 and T_1. We augment each cell in DM_k^1 with the *ratio of particle count of that cell in DM_k^1 to the particle count of the same cell in DM_k^0*. A density map that has such ratios is called a *ratio density map* (RDM). The next step is to update the histogram H_1 according to the ratios in the RDM. Let r_A and r_B ($A \neq B$) be density ratios of any two cells A and B in the RDM. We have two scenarios:

Case 1: $r_A \times r_B = 1$. In this case, we do not make any changes to H_1. It indicates that the two cells A and B contributed the same (or similar) distance counts to the corresponding buckets in both histograms H_0 and H_1.

Case 2: $r_A \times r_B \neq 1$, indicates that some changes have to be made to H_1. Specifically, we follow the Prop heuristic, as in ADM-SDH, to proportionally update the buckets that overlap with the distance range $[u, v]$. For example, as shown in Fig. 3, consider the distance range $[u, v]$ overlapping three buckets $i, i+1$, and $i+2$. The buckets and their corresponding count updates are given in Eqs. 7– 9.

$$H_1[i], \quad (n_A^1 n_B^1 - n_A^0 n_B^0)\frac{iw-u}{v-u} \tag{7}$$

$$H_1[i+1], \quad (n_A^1 n_B^1 - n_A^0 n_B^0)\frac{w}{v-u} \tag{8}$$

$$H_1[i+2], \quad (n_A^1 n_B^1 - n_A^0 n_B^0)\frac{v-(i+1)w}{v-u} \tag{9}$$

where n_A^0 and n_B^0 are counts of particles in cells A and B, respectively, in density map DM_k^0 of frame f_0. Similarly,

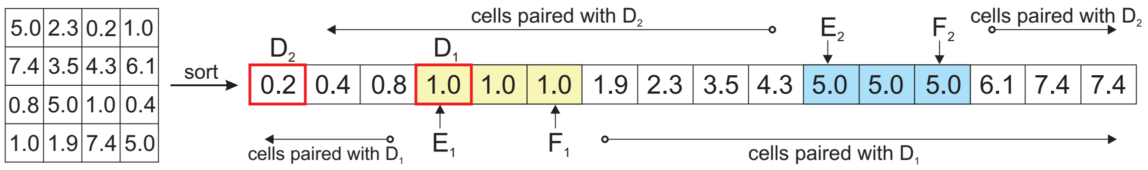


Figure 6: Grouping cells with equal density ratios by sorting the cell ratios in the RDM

n_A^1 and n_B^1 are counts of particles in corresponding cells of density map DM_k^1 in frame f_1. Note that we have $n_A^1 = r_A \cdot n_A^0$ and $n_B^1 = r_B \cdot n_B^0$. The total number of distances to be updated in the buckets is $n_A^1 \times n_B^1$ - $n_A^0 \times n_B^0$. This actually gives us the number of distances changed between cells A and B of density map DM_k, going from frame f_0 to frame f_1.

An efficient implementation of the above idea requires all pairs of cells that satisfy the **Case 1** condition to be skipped. In other words, our algorithm should **only process the Case 2 pairs, without even checking whether the product of two cells is** 1.0 (explained later). The histogram updates can be made efficiently if cells with equal or similar density ratios are grouped together. Our idea here is to store all the ratios in the RDM in a sorted array (Fig. 6). The advantage in sorting is that the sorted list can be used to efficiently find all pairs of cells with ratio product of 1.0. In other words, for any cell D with density ratio r_D, find the first cell E and the last cell F in the sorted list with ratios $1/r_D$, using binary search. Then, pair cell D with all other cells except the cells between E and F in the sorted list. Fig. 6 shows an example of a cell (D_1) with ratio 1.0 – we mark the first cell E_1 and the last cell F_1 with ratio of 1.0. Then we pair D_1 with rest of the cells in the list. Take another example of cell (D_2) with ratio 0.2 : we will effectively skip all the cells (E_2 to F_2) with ratio 5.0 (as $1/0.2 = 5.0$), and start pairing D_2 with those cells that do not have ratio 5.0 (to the left of E_2 and right of F_2). Again, there are also intra-cell distances to be processed (details in Appendix A).

In practice, a tolerance factor ϵ can be introduced to the **Case 1** condition such that the cells with ratio product within the range of $1.0 \pm \epsilon$ are skipped from the computations. While saving more time by allowing more cell pairs untouched, the factor ϵ can also introduce extra errors. However, our analysis in Section 6 shows that such errors are negligible.

5.3 Putting Both Ideas Together

The continuous histogram processing is sped up by utilizing both spatial uniformity and temporal locality properties. An overview of the technique is shown in the flow diagram of Fig. 7. The left branch (decision $A \equiv B$) is to compute the intra-cell distances. In the right branch we check the locality property of every pair of cells before checking for uniform distribution of the particles. Any pair that satisfies the locality property is skipped from further computations. The pairs that fail the locality property check are tested for the uniformity property. Based on the results of the check, subsequent steps are taken and the histogram buckets are updated.

The Monte Carlo simulation step introduced in our algorithm is expensive when computing SDH of a sequence of frames. As mentioned in section 5.1, the cost can actually spread over when we are processing a sequence of frames. It is an interesting fact that the tree building process is such that a cell in the DMs of same level in all frames is of same dimensions. Therefore, a simulation done once can be reused in all other frames. Given a pair of cells A and B and their respective distance range $[u, v]$, we compute the proportions of distances that map to each bucket covered by $[u, v]$ through Monte Carlo simulation. For each distinct $[u, v]$ range, we store such (and only such) proportions of distance distributions in a universal hash table.

For every pair of uniform cells that do not resolve and have distance range $[u, v]$, we look into the hash table to get the proportions to distribute the distances into buckets. If an entry is available in the hash table, we use it directly. Otherwise, a new simulation is done and proportions are calculated. This new simulation information is stored in the hash table. The hash table is universal and is used for computing the histogram of all the frames for a given bucket width.

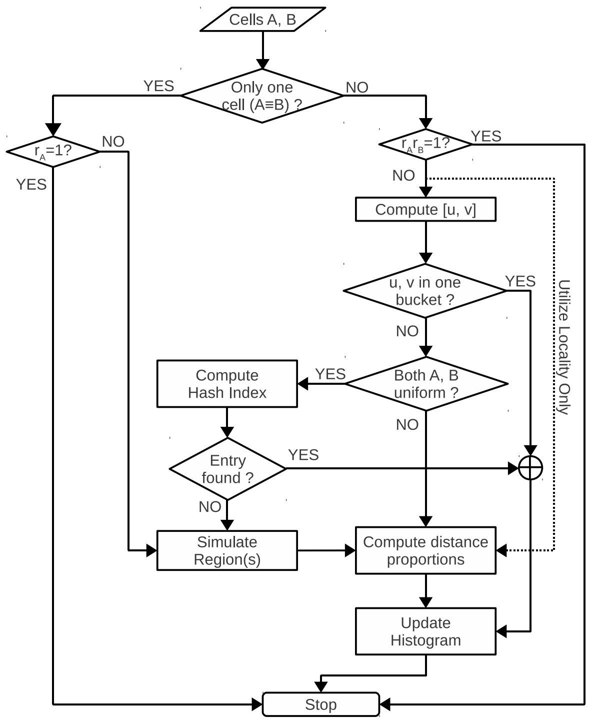


Figure 7: Steps in dealing with two cells of the composite algorithm for computing SDH

To simplify the implementation, one decision we made was to choose a level k in the DM-tree and process cells on that level only (instead of working on uniform regions on different levels). We need a level that balances both SDH computation time and the error – choosing a level close to the leaves may increase the time, while a level close to the root will introduce higher errors in the SDH. An important feature of our algorithm is that *the user can choose a level to run the algorithm according to her tolerance of the errors.* Such choices can be made beforehand by analysis as discussed in technical report [16]. Note that all the cells in the DM-tree that are uniform are marked before the continuous SDH processing begins.

The proposed technique is completely based on the general temporal and spatial uniformity of the data set. Such cell-wise uniformity is not only observed in MS, but also in many traditional spatiotemporal database applications [22]. Hence, it can be applied to very different data sets such as crowd of people and stars in astronomical studies.

6. PERFORMANCE ANALYSIS

The performance (running time and errors) of the proposed techniques is analyzed briefly in this section. More details are explained in technical report [16].

6.1 Time analysis

The running time of the algorithm utilizing only the spatial uniformity property is contributed by the following factors: (a) identifying uniform regions; This can be bound by $O(N \log N)$, as the count in each leaf node is used for at most $O(\log N)$ chi-square tests; (b) distributing distances into buckets; For this, all pairs of cells on a DM need to be processed - in a DM with M cells, the time is $O(M^2)$. (c) Monte Carlo simulations that require $O(MT_s)$ time according to Theorem 5.2. Here T_s is the time of each individual simulation that can be regarded as a constant, and such cost can be amortized into the large number of frames.

Theoretically, the cost of identifying uniform regions dominates, as the complexity is related to system size N. However, the $O(M^2)$ time for factor (b) can dwarf it if we choose a DM on the lower levels of the quad tree - M approaches N when the level gets lower.

6.2 Error analysis

Based on the sources, two types of errors are introduced by utilizing the spatial uniformity feature:

I. error (e_u) by pairs of cells that are both uniform, and

II. error (e_a) by those with at least one non-uniform cell.

Type I error is introduced by: (a) approximation error of Pearson's χ^2 test statistic [4], which is up to the order of $O_t^{\frac{df-1}{df}}$, where *df* is degree of freedom and O_t is the number of observations in χ^2 test. When *df* is sufficiently large, the error in marking a cell uniform is $e_u = 1/O_t$. (b) simulation error, which, according to the Law of Iterated Logarithm (LIL) [4], is up to the order of $(\frac{n_A n_B}{\log \log n_A n_B})^{-1/2}$, where n_A and n_B are the number of points simulated in cells A and B, respectively. Considering a scenario where n_A and n_B are of the order of 10^2, the simulation error is slightly smaller than the order of 10^{-2}.

It is easy to see that the Type II error is as much as the error achieved by the PROP heuristic. It is hard to get a tight error bound when the distribution of points in a cell is not uniform. We have done intensive qualitative research and found that the error (theoretical) can be loosely bounded by 10% [26]. Of course, this is not a tight bound, and our experimental results (section 7) show that the error is much smaller (less than 1%).

6.3 Error/Performance Tradeoff

Given the above analysis, we show our algorithm is *tunable* in that the user can choose a level of the DM-tree to get a desired error guarantee. Suppose p_u is the fraction of pairs of cells that are uniform on a given level, the total error ξ produced by our algorithm based on spatial uniformity is

$$\xi \leq e_u p_u + e_a(1 - p_u) \tag{10}$$

From the above equation, we can solve p_u to obtain a guideline on the level of the DM-tree we run the algorithm:

$$p_u \geq \frac{e_a - \xi}{e_a - e_u} \tag{11}$$

In other words, a user will choose to work on a DM where the fraction of uniform cells is at least $\sqrt{p_u}$, in order to get an error lower than ξ.

The performance analysis about temporal locality is done in similar way. The details are presented in [16], due to space limitations. In summary, with a negligible loss of accuracy, the application of spatial and temporal uniformity improves the performance significantly.

7. EXPERIMENTAL RESULTS

The proposed continuous SDH computation method was implemented in C++ programming language and tested on real MS data sets. The experiments were conducted on an Apple Xserve server with two Intel quad-core processors and 24 GB of physical memory. The Xserve was running OS X 10.6 Snow Leopard operating system. We tested the following algorithms to evaluate the performance of our approach.

A1: The ADM-SDH algorithm presented by Tu *et. al.* [25]. This method processes SDH frame by frame; and distributes the distances using PROP exclusively;

A2: The algorithm utilizing only temporal locality to compute SDH continuously over multiple frames;

A3: The algorithm utilizing only spatial uniformity to compute SDH frame by frame;

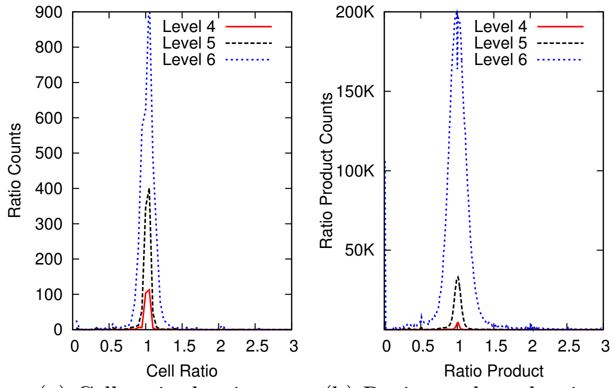
A4: The algorithm utilizing both temporal locality and spatial uniformity to compute SDH continuously.

The running times of the algorithms on different data sets are measured for comparison, along with the errors introduced due to approximation. The errors are computed by comparing the approximate SDH results with the correct

SDH of each frame. The error (in percentage) of each frame is calculated as

$$P_{error} = 100 \times \sum_i |H_i - H'_i| \Big/ \sum_i H_i$$

where H_i and H'_i are the correct and approximate distance counts, respectively, in bucket i of the histogram.



(a) Cell ratio density (b) Ratio product density

Figure 8: Temporal similarity between two consecutive frames chosen randomly from the dataset of $890K$ atoms

Data Sets: Two datasets from different simulation systems were used for experiments. The first dataset consists of frames captured from a collagen fiber simulation system. The simulated system is made of $890,000$ atoms and their positions are stored in a total number of $10,000$ frames. The second dataset is collected from a simulation of cross membrane protein system with about $8,000,000$ atoms and $10,000$ frames. We randomly selected a chunk of about 100 consecutive frames from the first data set and 11 frames from the second dataset for our experiments. The main bottleneck in testing the algorithms is computing the correct histogram of the frames, needed to compute the error. Obtaining correct histogram is basically running the naive or DM-SDH algorithm, which is computationally expensive. Therefore, we could only get the correct histograms of 11 frames from the 8 million data set (by naive approach in about 27 days!).

We observed significant temporal similarity among frames of the dataset. The success of utilizing the temporal similarity property depends on the total fraction of cells that exhibit such property. In fact, the running time of the technique is affected by the number of cell pairs (A, B) for which $r_A \times r_B = 1 \pm \epsilon$. Figs. 8(a) and 8(b) show the density of ratios and ratio products on each level of the DM-tree in two consecutive frames, chosen randomly from the data set of $890,000$ atoms. For all levels we tested, majority of the cells (cell pairs) show ratio (ratio product) that is close to 1.0. The number of cell pairs with ratio product of 1.0 increases as we descend down the tree. Similar trends are observed in the other data set also. This confirmed the great potential of using temporal similarity to save time in SDH processing.

Fig. 9(a) and 9(c) show the running time of all the algorithms for different bucket widths. It can be noted that the running time of $A1$ can be orders of magnitude longer than our proposed algorithms. The important observation to be made about algorithm $A1$ is that the running time increases dramatically with the decrease of w (note the logarithmic scale). Method $A2$ is similar to $A1$ but, utilizes temporal locality while working on only one level. When the bucket width is small, both methods work on lower tree levels, with small number of atoms in cells. The utilization of locality gives scope to save some running time in $A2$. Unlike the first two methods, the time spent by methods $A3$ and $A4$ does not change much with the change of bucket width w. Note that the average running times presented here have amortized all "start-up" costs including that for running Monte Carlo simulations, spatial uniformity test, and that for computing the SDH for the first frame.

The errors (in percentage) of each method are shown in Fig. 9(b) and 9(d) for different values of w. The errors rendered by $A3$ and $A4$ are always lower than those by method $A1$. However, the errors of $A2$ are slightly higher than $A1$ for small bucket widths. The number of distances to be distributed between two cells is very small, as the algorithm works close to leaf level. Therefore, by utilizing the temporal locality property the small errors are added on top of the PROP method applied for other cell pairs. Although method $A4$ is faster than $A3$, the price for that is an error rate that is slightly higher. However, it provides a good tradeoff as the improvement of performance is of larger magnitude than the loss of accuracy (note the differences between the lines of A3 and A4). The method $A3$ stands clear winner for producing the lowest errors. The distance distribution curve computed by Monte Carlo simulations diminishes the error that would have been introduced by heuristically distributing distances as in $A1$. The errors in method $A1$ stay low (still equal to or higher than other methods) for smaller bucket widths but goes higher under larger w values. The reason being, proportions for small buckets are almost similar in all the algorithms. Number of distances that are in the range of very small buckets are few and therefore their proportional distribution are not much different. Hence, the error is low. With the increase of bucket width, $A1$ would end up distributing the distances equally in all the buckets while our methods accurately compute the proportions of distances that should go into each bucket.

Number of simulations: Most of the initial time in computation of the first few frames is spent in performing the simulations to update the hash table entries. In our experiments on the data set of $890,000$ atoms, the number of simulations performed for each frame dropped quickly. In total, 100 frames were processed to compute SDH using algorithm $A3$. Fig. 11(a) shows the distribution of simulations performed over 100 frames. We can see that the first frame peaks at 120 simulations. In most of the other frames, no simulations are performed except for few frames for which less than 25 simulations are performed. This clearly states that the hash table utilized in $A3$ saves running time by reusing the simulations performed in previous frames.

The resolved pairs of cells eliminate direct computation of large number of distances, saving simulation time as well. Fig. 10 shows the number of such distance computations eliminated in SDH processing with different bucket sizes.

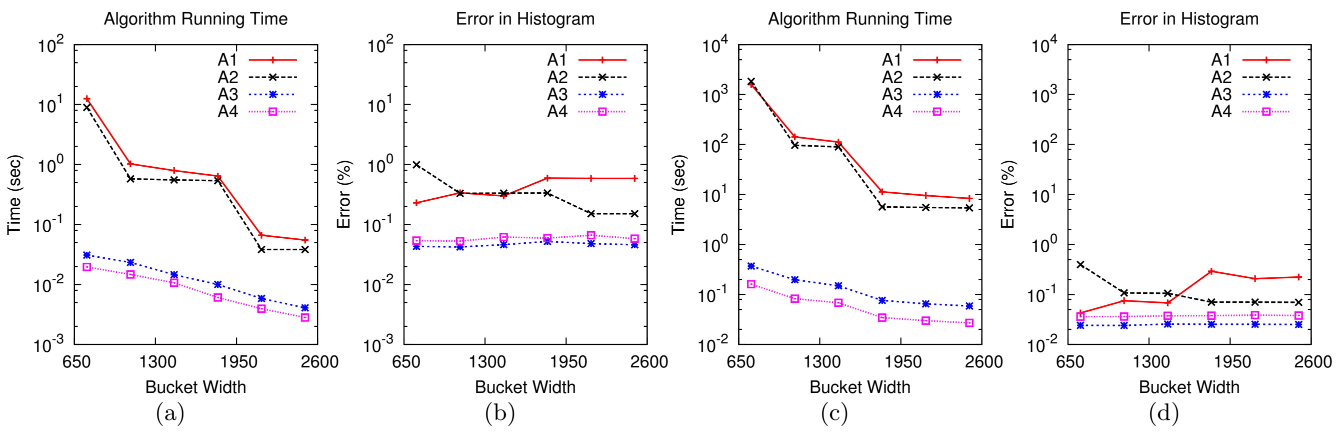


Figure 9: Comparison of running time and percentage error of the algorithms/methods (y-axis has *log* scale). (a)–(b) The results from $890,000$ atom data set. (c)–(d) Results from 8 million atom data set

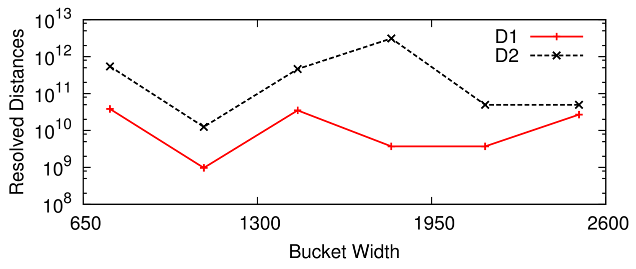


Figure 10: Distances resolved in SDH computation. Datasets: D1 - 890K atoms; D2 - 8 million atoms (y-axis has *log* scale)

Simulation size: The number of points used in every Monte Carlo simulation does not affect the SDH results, as long as sufficient number of points (i.e., 25) are generated. The error shown in Figure 11(b) does not change for different sizes of simulations performed. Thus, our analysis in section 6 only gives a loose error bound whereas the actual errors are much lower.

One of the algorithms presented here can be chosen based on the performance/error tradeoff criteria. One way to quantify the performance/error tradeoff is the *product of time and error* - an algorithm with lower *time–error product* (TEP) is preferred. We calculated the TEPs of all tested algorithms and found that, among all settings and algorithms, $A4$ stands clear winner by producing the smallest TEPs under all bucket widths (details are discussed in technical report [16]). In summary, the computation of SDH based on spatial uniformity gives much better performance. The idea of utilizing the temporal locality can be combined with both $A1$ and $A3$ to make the computations faster.

8. CONCLUSIONS AND FUTURE WORK

An efficient approximate solution to the spatial distance histogram query is provided in this paper. SDH is one of the very important molecular simulation data analysis queries that is frequently applied to a collection of data frames. We used the point region Quad-tree efficiently to take advantage of the data locality and statistical data distribution properties. The algorithm presented is practically feasible to analyze data of large number of frames continuously. Its efficiency and accuracy are supported by mathematical analysis and extensive experimental results. As a follow-up work, we are investigating the implementation of our algorithms in modern parallel platforms such as Graphics Processing Units (GPU). An immediate work of interest is to extend our algorithm to spatial particle distributions other than the uniform pattern. Another important direction of research would be to study the feasibility of utilizing spatiotemporal uniformity properties for the computation of general m-body correlation functions in scientific databases.

Acknowledgments

The authors would like to thank Dr. Sagar Pandit in the Department of Physics at the University of South Florida for discussions on spatial distance histogram computation and, in general, multi-body correlation functions. The project described was supported by Award R01GM086707 from the National Institute of General Medical Sciences (NIGMS) at the National Institutes of Health (NIH), USA. The content is solely the responsibility of the authors and does not necessarily represent the official views of NIGMS or NIH.

9. REFERENCES

[1] M. Allen. *Introduction to Molecular Dynamics Simulation*, volume 23. John von Neumann Institute of Computing, NIC Seris, 2003.

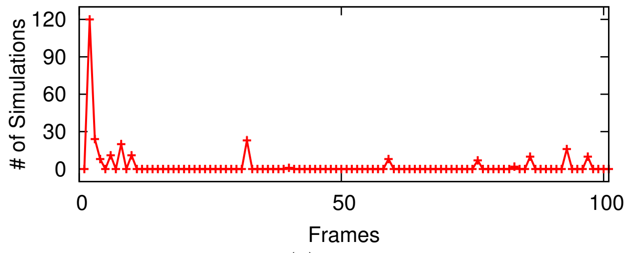
[2] Andrey Omeltchenko *et. al.* Scalable I/O of large-scale molecular dynamics simulations: A data-compression algorithm. *Computer physics communications*, 131(1–2):78–85, 2000.

[3] J. Barnes and P. Hut. A Hierarchical O(N log N) Force-Calculation Algo. *Nature*, 324(4):446–449, 1986.

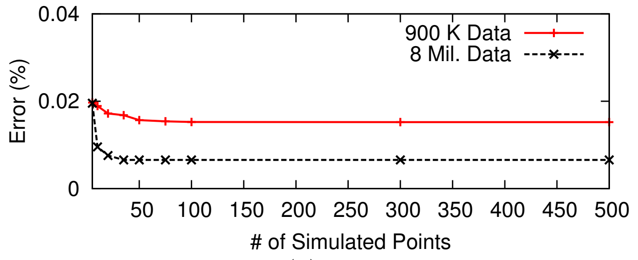
[4] R. N. Bhattacharya and N. H. Chan. Comparisons of chisquares, edgeworth expansions and bootstrap approximations to the distribution of the frequency chisquare. *Indian J. of Statistics*, 58(1):57–68, 1996.

[5] L. Breiman. *Probability (Classics in Applied Mathematics)*. Society for Industrial and Applied Mathematics, 1992.

[6] S. Chen, Y.-C. Tu, and Y. Xia. Performance analysis of a dual-tree algorithm for computing spatial distance



(a)



(b)

Figure 11: (a) Number of simulations performed per frame to process 100 frames together (b) Effect of simulation size on SDH error

histograms. *The VLDB Journal*, 20, 2011.

[7] P. Dietz and R. Raman. Persistence, amortization and randomization. In *Proc. of the ACM-SIAM symposium on Discrete algorithms*, pages 78–88, 1991.

[8] M. Eltabakh, M. Ouzzani, and W. Aref. BDBMS: A Database management system for biological data. In *CIDR*, pages 196–206, 2007.

[9] D. Frenkel and B. Smit. *Understanding Molecular Simulation: From Algorithms to Applications*, volume 1. Academic Press, Inc., 2nd edition, 2001.

[10] A. G. Gray and A. W. Moore. N-body problems in statistical learning. In *Advances in Neural Info. Processing Systems*, pages 521–527, 2001.

[11] J. Gray, D. T. Liu, M. Nieto-Santisteban, A. Szalay, D. J. DeWitt, and G. Heber. Scientific data management in the coming decade. *In SIGMOD Record*, 34:34–41, 2005.

[12] L. Greengard and V. Rokhlin. A Fast Algorithm for Particle Simulations . *Journal of Computational Physics*, 135(12):280–292, 1987.

[13] P. Greenwood and M. Nikulin. *A Guide to Chi-Squared Testing*. Wiley-Interscience, 1st edition, 1996.

[14] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl. GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation. *Journal of Chemical Theory and Computation*, 4(3):435–447, March 2008.

[15] H. Kaplan. Persistent data structures. In *Handbook on Data Structures and Applications*, pages 1–27. CRC Press, 2001.

[16] A. Kumar, V. Grupcev, Y. Yuan, Y.-C. Tu, and G. Shen. Distance Histogram Computation Based on Spatiotemporal Uniformity in Scientific Data. Technical Report CSE/11-053, URL: http://www.cse.usf.edu/˜ytu/pub/tech11-053.pdf, CSE Dept., Univ. of South Florida, USA., 2011.

[17] G. Lagogiannis, N. Lorentzos, S. Sioutas, and E. Theodoridis. A time efficient indexing scheme for complex spatiotemporal retrieval. *ACM SIGMOD Record*, 38:11–16, 2010.

[18] D. Landau and K. Binder. *A Guide to Monte Carlo Simulations in Statistical Physics*. Cambridge University Press, 2005.

[19] M. H. Ng *et. al.* In BioSimGrid: grid-enabled biomolecular simulation data storage and analysis. *Future Gen. Comput. Syst.*, 22:657–664, 2006.

[20] J. A. Orenstein. Multidimensional tries used for associative searching. *Information Processing Letters*, 14(4):150–157, 1982.

[21] I. Szapudi. *Introduction to Higher Order Spatial Statistics in Cosmology*, volume 665. Lecture Notes in Physics, Springer Verlag, 2009.

[22] Y. Tao, J. Sun, and D. Papadias. Analysis of predictive spatio-temporal queries. *ACM Trans. Database Syst.*, 28(4):295–336, 2003.

[23] Teruhiko Teraoka *et. al.* The MP-tree: A data structure for spatio-temporal data. In *Proceedings of the Phoenix Conference on Computers and Communications*, pages 326–333, 1995.

[24] The R Development Core Team. *R Reference Manual: Base Package*, volume 1. Network Theory Ltd., 2003.

[25] Y.-C. Tu, S. Chen, and S. Pandit. Computing distance histograms efficiently in scientific databases. In *ICDE*, pages 796–807, 2009.

[26] Y. Yuan, V. Grupcev, Y.-C. Tu, S. Chen, S. Pandit, and M. Weng. On Fast Algorithms for Computing Spatial Distance Histograms (SDH). Technical Report CSE/11-052, URL: http://www.cse.usf.edu/˜ytu/pub/tech11-052.pdf, CSE Dept., Univ. of South Florida, USA., 2011.

APPENDIX

A. HANDLING INTRA-CELL DISTANCES

Given a cell A with diagonal length of q, the distances between two particles within A fall into the range $[0, q]$. We also assume this range overlaps over buckets 0 to i.

Spatial uniformity based algorithm: We run a simulation by sampling from a single uniform region with dimensions equal to those of cell A. This will generate a distance distribution that spans over buckets 0 to i. We follow the same idea shown in Eqs. 1-3 to assign the distance counts of cell A into the buckets:

$$\begin{array}{ll} H[0], & \left(\frac{n_A^1(n_A^1-1)}{2}\right)\int_0^w g(t)dt \\ H[1], & \left(\frac{n_A^1(n_A^1-1)}{2}\right)\int_w^{2w} g(t)dt \\ \dots & \dots \\ H[i], & \left(\frac{n_A^1(n_A^1-1)}{2}\right)\int_{(i-1)w}^{q} g(t)dt \end{array}$$

Temporal locality based algorithm: Each individual cell of the RDM with $r_i \neq 1$ is used to update the counts.

Specifically,

$$H_1[0,\ldots,i-1], \quad \left(\frac{n_A^1(n_A^1-1)}{2}-\frac{n_A^0(n_A^0-1)}{2}\right)\frac{w}{q}$$

$$H_1[i], \quad \left(\frac{n_A^1(n_A^1-1)}{2}-\frac{n_A^0(n_A^0-1)}{2}\right)\frac{(i-1)w}{q}$$

B. TOTAL SIMULATIONS PERFORMED

The density map is organized as a grid of $M = n \times n$ cells. We represent the position of each cell by an ordered pair (x, y), where x and y are the horizontal and vertical displacements respectively, of the cell from the top-left corner of the density map. A cell C with displacements i, j is represented by $C(i, j)$. The width or side of each cell is denoted by t (see Fig. 12). We discuss the number of Monte Carlo simulations performed in a density map through a special feature called L-shape (Definition. B.1). The number of simulations performed is directly related to the number of distinct L-shapes found in the density map.

Definition B.1. *L-shape of two cells A and B, $L(A, B)$, is a subset of the density map that includes the two end cells $A(x_A, y_A)$ and $B(x_B, y_B)$ and all the cells with positions*

$$(x_A+1, y_A), (x_A+2, y_A), \ldots, (x_B, y_A) \text{ and}$$
$$(x_B, y_A+1), (x_B, y_A+2), \ldots, (x_B, y_B-1)$$

or the positions

$$(x_A, y_A+1), (x_A, y_A+2), \ldots, (x_A, y_B) \text{ and}$$
$$(x_A+1, y_B), (x_A+2, y_B), \ldots, (x_B-1, y_B)$$

Without loss of generality we assume $x_A < x_B$ and $y_A < y_B$ in rest of the discussion. It is to be noted that both cells, A and B, have only one neighbor cell in the $L(A, B)$-shape.

Definition B.2. *The size of an $L(A, B)$ shape, denoted as $d(L(A, B))$, is the ordered pair (a, b) where a is the horizontal distance (counted in number of cells) and b is the vertical distance between the cells A and B.*

Definition B.3. *Equivalent L-shapes: Let $L(A, B)$ and $L(C, D)$ be two L-shapes with sizes $d(L(A, B)) = (a, b)$ and $d(L(C, D)) = (c, d)$. Then $L(A, B)$ is equivalent to $L(C, D)$ (i.e,. $L(A, B) \equiv L(C, D)$) iff ($a = c$ and $b = d$) or ($a = d$ and $b = c$).*

Lemma B.4. *$L(A, B) \equiv L(C, D)$ iff the minimum and maximum distances between A, B and between C, D are equal. In other words, $L(A, B) \equiv L(C, D)$ iff $dist_{min,max}(A, B) = dist_{min,max}(C, D)$.*

Proof. Consider two L-shapes, $L(A, B)$ and $L(C, D)$ with sizes $d(L(A, B)) = (a, b)$ and $d(L(C, D)) = (c, d)$.

If $L(A, B) \equiv L(C, D)$ then, by the definition B.3, $d(L(A, B)) = d(L(C, D))$. Thus, $a = c$ and $b = d$ or $a = d$ and $b = c$.

Fig. 12 shows maximum distance between cells A and B.

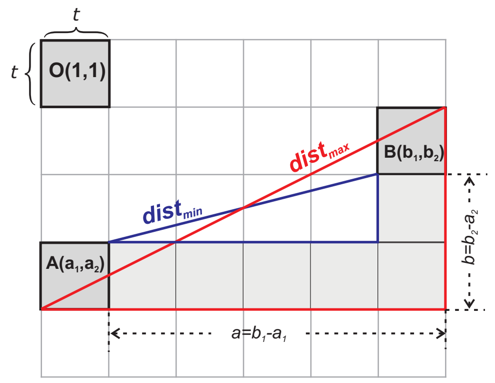


Figure 12: Illustration of L-shape $L(A, B)$ of size $d(L(A, B)) = (a, b)$ in a density map

$$\begin{aligned} dist_{max}(A,B) &= \sqrt{((a+1)*t)^2 + ((b+1)*t)^2} \\ &= \sqrt{((c+1)*t)^2 + ((d+1)*t)^2} \\ &= dist_{max}(C,D) \end{aligned}$$

Similarly for the minimum distance between cells A and B,

$$\begin{aligned} dist_{min}(A,B) &= \sqrt{((a-1)*t)^2 + ((b-1)*t)^2} \\ &= \sqrt{((c-1)*t)^2 + ((d-1)*t)^2} \\ &= dist_{min}(C,D). \end{aligned}$$

Let two pairs of cell (A, B) and (C, D) have same minimum and maximum distance between them i.e.,

$$dist_{min,max}(A,B) = dist_{min,max}(C,D)$$

Thus,

$$\sqrt{((a-1)*t)^2 + ((b-1)*t)^2} = \sqrt{((c-1)*t)^2 + ((d-1)*t)^2}$$

The equation holds only if ($a = c$ and $b = d$) or ($a = d$ and $b = c$). Thus, $d(L(A, B)) \equiv d(L(C, D))$. By definition, if two L-shapes have same size, they are equivalent. □

Theorem B.5. *The number of distinct L-shapes (regardless of position) in a density map with $M = n^2$ cells is $\frac{n(n+1)}{2} - 1$.*

Proof. The form of each L-shape $L(A, B)$ is defined by its size $d(L(A, B)) = (a, b)$, where $0 \le a \le n-1$ and $0 \le b \le n-1$. But, since the L shapes with size (a, b) are equivalent to the L-shapes with size (b, a) we need only to count the L-shapes with size (a, b) where $b \ge a$ and $b \ne 0$. The number of such L-shapes for given values of $a = 1, 2, \ldots n-1$ are $n-1, n-2 \ldots, 1$ respectively. For $a = 0$ there are $n-1$ L-shapes. Obviously, the total number of all distinct L-shapes of size (a, b) is $\frac{n*(n+1)}{2} - 1$. □

As the number of distinct Monte Carlo simulations performed in an RDM is equal to the number of distinct L-shapes, the total number of simulation performed to compute SDH is bound by $O(M)$.