# Exact and Approximate Computation of a Histogram of Pairwise Distances between Astronomical Objects

Bin Fu, Eugene Fink, Garth Gibson and Jaime Carbonell

Computer Science, Carnegie Mellon University
Pittsburgh, Pennsylvania 15213, United States
{binf, e.fink, garth, jgc}@cs.cmu.edu

## ABSTRACT

We compare several alternative approaches to computing *correlation functions*, which is a cosmological application for analyzing the distribution of matter in the universe. This computation involves counting the pairs of galaxies within a given distance from each other and building a histogram that shows the dependency of the number of pairs on the distance.

The straightforward algorithm for counting the exact number of pairs has the $O(n^2)$ time complexity, which is unacceptably slow for most astronomical and cosmological datasets, which include billions of objects. We analyze the performance of several alternative algorithms, including the exact computation with an $O(n^{5/3})$ average running time, an approximate computation with linear running time, and another approximate algorithm with sub-linear running time, based on sampling the given dataset and computing the correlation functions for the samples. We compare the accuracy of the described algorithms and analyze the tradeoff between their accuracy and running time. We also propose a novel hybrid approximation algorithm, which outperforms each other technique.

## Categories and Subject Descriptors

J.2 [**Computer Applications**]: Physical Sciences and Engineering – *Astronomy*.

## General Terms

Algorithms, Performance, Experimentation.

## Keywords

Approximation, astrophysics, large-scale data, eScience, *kd*-tree, sampling, Hadoop.

## 1. INTRODUCTION

Today more and more massive scientific datasets are becoming available, which makes it possible to apply data-driven approaches to science. In astrophysics, digital sky surveys such as Sloan Digital Sky Survey [1] and Guide Star Catalog II [8], as well as cosmological simulations such as BHCosmo [4] and Coyote Universe [7], provide datasets with billions of celestial objects. This new trend poses challenges for analyzing these massive datasets.

We are addressing this challenge by developing scalable algorithms. First, we have developed a distributed algorithm that identifies clusters of objects in an astrophysics dataset [5]. In this paper, we address another astrophysics problem: building a histogram of pairwise distances between celestial objects, the *correlation function*. This histogram helps astronomers analyze cosmological models [10]. Since a naive solution takes quadratic time, a more efficient solution is necessary for large datasets.

Researchers have proposed several approaches to computing correlation functions. In particular, Gray and Moore used *kd*-trees [6], and Belussi and Faloutsos developed an approximation algorithm based on fractal dimensions [2]; however, our experiments have shown that the existing techniques are either impractically slow or give inaccurate approximations. We propose a sampling method and combine it with the *kd*-tree technique, which results in an efficient and accurate approximation of the correlation function, applicable to datasets with billions of objects.

We give a definition of the correlation function in Section 2, explain the experimental setup in Section 3, and describe existing techniques in Sections 4–6. We then present our sampling technique in Section 7, a hybrid technique in Section 8, and its distributed version in Section 9.

## 2. PROBLEM

We assume that every astronomical object is a point in three dimensions with known coordinates. We are given a set of $N$ astronomical objects, denoted $p_1$, $p_2$, …, $p_N$, and a strictly increasing series of $M + 1$ distances, denoted $d_0$, $d_1$, …, $d_M$, defining the bins of a histogram.

For each index $i$ from 1 to $M$, we need to determine the number of object pairs such that the distance between each pair is between $d_{i-1}$ and $d_i$:

$$cf(i) \text{ is the number of pairs } (p_u, p_v), \text{ where } u < v,$$
$$\text{such that } d_{i-1} \le dist(p_u, p_v) < d_i$$

where $dist(p_u, p_v)$ is the Euclidean distance between $p_u$ and $p_v$.

We assume that the given sequence of distances is a geometric progression. That is, we are given the distance $d_0$ and a constant $C > 1$, and we need to compute correlation functions for the distances $d_0$, $C \cdot d_0$, $C^2 \cdot d_0$, ..., $C^M \cdot d_0$.

## 3. EXPERIMENTAL SETUP

We conduct experiments on a dataset of 4.5 million objects, with coordinate values between 0.0 and 40.0 (the unit in this dataset is


*AstroHPC'12*, June 19, 2012, Delft, The Netherlands.

Mpc/h [4]), which has been provided by our collaborators from McWilliams Center for Cosmology at Carnegie Mellon University. We compute the correlation functions with the following parameters unless indicated otherwise: $M$ = 257, $d_0$ = 0.001, and $C$ = 1.044.

We have implemented all the sequential algorithms (Sections 4–8) in Java 1.6 and tested them on a 2.66GHz Intel Core 2 Duo desktop with 2GB memory. We have conducted the distributed computing experiments (Section 9) in a dataintensive computing facility in Carnegie Mellon University, called the DiscCloud cluster, which consists of 64 compute nodes. Each node has eight 2.8GHz CPU cores in two quad-core processors, 16 GB of memory and four 1 TB SATA disks. The nodes are connected by a 10 GigE network.

## 4. NAIVE ALGORITHM

We first consider the straighforward algorithm shown in Figure 1, which iterates over all pairs of objects, thus taking $O(N^2)$ time. In Figure 2, we compare the theoretical estimation and the actual running time of the algorithm.

**Output**: Counters $cf_1, cf_2, \ldots, cf_M$

**for** $i = 1$ **to** $M$ **do** $cf_i = 0$
**for** $u = 1$ **to** $N - 1$ **do**
  **for** $v = u + 1$ **to** $N$ **do**
    find $i$ such that $d_{i-1} \le dist(p_u, p_v) < d_i$; $cf_i$ ++

**Figure 1. Naive algorithm.**

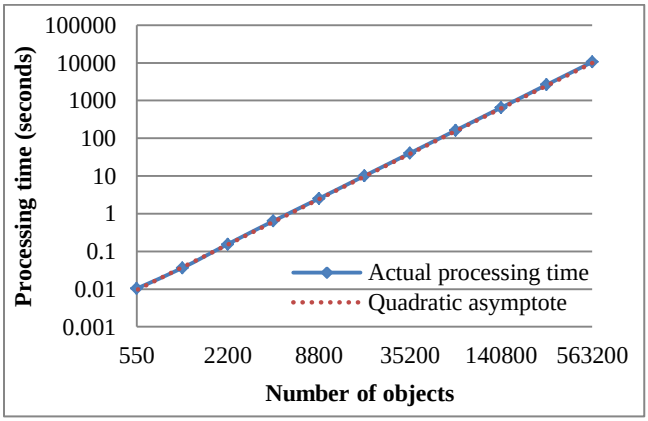


**Figure 2. Running time of the naive algorithm, which is quadratic on the number of objects. Note that both axes are on logarithmic scale.**

There are two approaches to determine in which histogram bin each object pair falls: a single logarithm operation (since the distance sequence is a geometric progression) and binary search. While the theoretical time complexity of the single logarithm operation is constant, its computation is slow in practice, and it is slower than binary search for a short sequence of distances. In Figure 3, we compare the running time of these two approaches. The results show that the logarithm computation is more efficient only when the length of distance sequence, $M$, is larger than 75.

In Figure 4, we show the exactly computed correlation function for the dataset of 4.5 million objects used in the experiments. The time to compute this function using the naive algorithm is 176 hours (7.3 days).

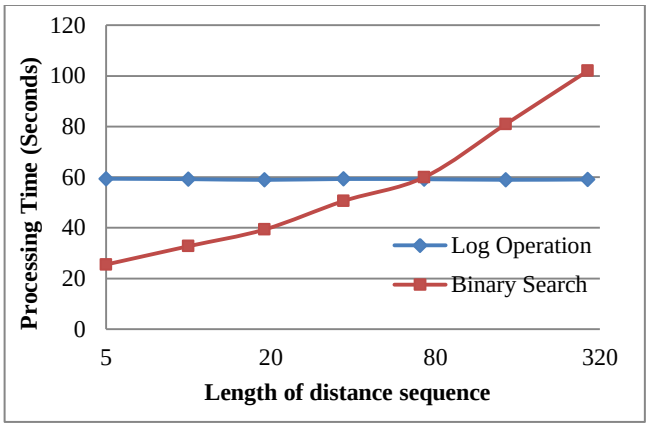


**Figure 3. Running time of the two approaches in the naive algorithm: logarithm operation and binary search. Note that the horizontal axis is on logarithmic scale. The use of logarithm operation is faster when the length of the distance sequence is over 75, that is, $M > 75$.**

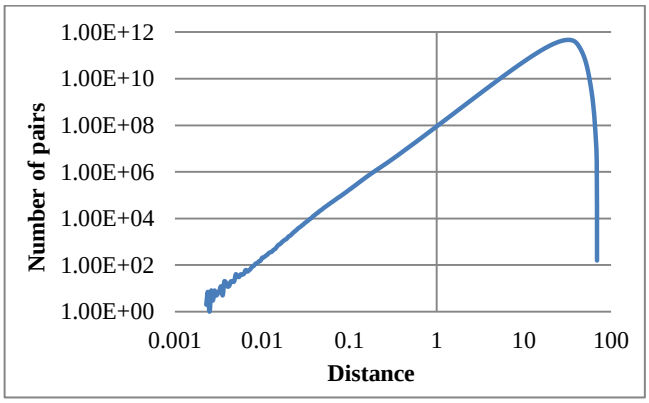


**Figure 4. The exact correlation function for the set of 4.5 million objects used in the experiments. Note that both axes are on logarithmic scale.**

## 5. *KD*-TREE IMPLEMENTATION

Gary and Moore used *kd*-tree to accelerate the computation of correlation functions [6], which is presented in Figure 5. Unlike the naive algorithm, their procedure processes one range of distances, $d_{i-1}$ to $d_i$, at a time. The *kd*-tree structure supports tree pruning, which speeds up the computation. The time complexity of processing a single range of distances is $O(N^{5/3})$. The overall processing time grows as we increase the length $M$ of the distance sequence, as shown in Figure 6.

To improve the efficiency, we have developed a new version of the *kd*-tree algorithm, called *multiple-range* algorithm, designed for processing multiple ranges in one pass. The pseudo-code of the multiple-range *kd*-tree algorithm is shown in Figure 7. We have conducted experiments with both the original single-range algorithm and the new multiple-range algorithm. The main results are as follows.

a. As shown in Figure 8, the single-range algorithm follows the $O(N^{5/3})$ asymptote. The multiple-range algorithm is in practice faster than the single-range *kd*-tree algorithm, but its asymptotic complexity is $O(N^2)$.

**Output**: Counters $cf_1, cf_2, \ldots, cf_M$

**for** $i = 1$ **to** $M$ **do** $cf_i = 0$
construct a *kd*-tree from the *root* node. Each node in the *kd*-tree has a *left* child and a *right* child. The bounding box of each node is calculated and stored. The number of objects (*num*) in each node is also calculated.
**for** $i = 1$ **to** $M$ **do**
  $cf_i$ = SINGLE-DISTANCE(*root*, *root*, $d_{i-1}$, $d_i$);

SINGLE-DISTANCE(*kd*-node $n_1$, *kd*-node $n_2$,
  double *small*, double *large*)
**if** ($n_1.num < 1$ || $n_2.num < 1$) **then return** 0
**if** ($n_1.num == 1$ && $n_2.num == 1$) **then**
  //assume $n_1$ includes object $p_1$ and $n_2$ includes object $p_2$
  **if** ($small \le dist(p_1, p_2) < large$) **then return** 1
  **else return** 0
calculate the maximum distance (*max*) and the minimum distance (*min*) between the bounding boxes of $n_1$ and $n_2$.
**if** ($min \ge large$ || $max < small$) **then return** 0
**if** ($min \ge small$ && $max < large$) **then**
  **return** $n_1.num \cdot n_2.num$
**if** ($n_1.num > n_2.num$) **then**
  **return** SINGLE-DISTANCE($n_1.left$, $n_2$, *small*, *large*) +
    SINGLE-DISTANCE($n_1.right$, $n_2$, *small*, *large*)
**else**
  **return** SINGLE-DISTANCE($n_1$, $n_2.left$, *small*, *large*) +
    SINGLE-DISTANCE($n_1$, $n_2.right$, *small*, *large*)
**end procedure**

**Figure 5. Single-range *kd*-tree algorithm.**

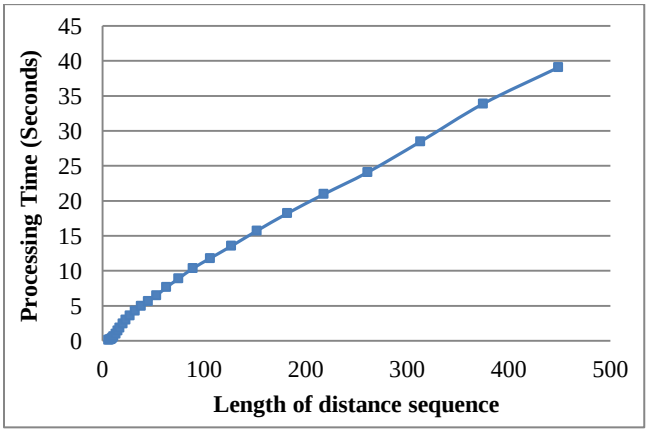


**Figure 6. Dependency of the running time on the length of distance sequence for the single-range *kd*-tree algorithm. We have run this experiment with a dataset of 75 thousand objects.**

b. Why is the time complexity of the multiple-range *kd*-tree $O(N^2)$? The reason is that $C$ in our experiments is relatively small, specifically, $C = 1.044$, which means the ranges used in constructing the histogram of distances are fine-grained, and thus the number of objects in each range is small. For a small number of objects, the tree pruning does not lead to the reduction of the asymptotic time complexity.

c. Since the distance sequence is relatively long ($M = 257$), the *kd*-tree algorithms run slower than the naive algorithm.

**Output**: Counters $cf_1, cf_2, \ldots, cf_M$

**for** $i = 1$ **to** $M$ **do** $cf_i = 0$
construct a *kd*-tree from the *root* node. Each node in the *kd*-tree has a *left* child and a *right* child. The bounding box of each node is calculated and stored. The number of objects (*num*) in each node is also calculated.
MULTIPLE-DISTANCES(*root*, *root*);

MULTIPLE-DISTANCES(*kd*-node $n_1$, *kd*-node $n_2$)
**if** ($n_1.num < 1$ || $n_2.num < 1$) **then return**
**if** ($n_1.num == 1$ && $n_2.num == 1$) **then**
  //assume $n_1$ includes object $p_1$ and $n_2$ includes object $p_2$
  find $i$ such that $d_{i-1} \le dist(p_1, p_2) < d_i$; $cf_i$ ++
  **return**
calculate the maximum distance (*max*) and the minimum distance (*min*) between the bounding boxes of $n_1$ and $n_2$.
**if** ($min \ge d_M$ || $max < d_0$) **then return**
**if** ([*min*, *max*] $\in [d_{i-1}, d_i)$ for an $i$) **then**
  $cf_i$ += $n_1.num \cdot n_2.num$;
  **return**
**if** ($n_1.num > n_2.num$) **then**
  MULTIPLE-DISTANCES($n_1.left$, $n_2$)
  MULTIPLE-DISTANCES($n_1.right$, $n_2$)
**else**
  MULTIPLE-DISTANCES($n_1$, $n_2.left$)
  MULTIPLE-DISTANCES($n_1$, $n_2.right$)
**end procedure**

**Figure 7. Multiple-range *kd*-tree algorithm.**

d. The processing time of the single-range *kd*-tree algorithm differs significantly for different distances, as shown in Figure 9. The results presented in Figure 9 suggest that the pruning is most effective for small and very large distances.

To summarize, we have confirmed that both the single-range and the multiple-range *kd*-tree algorithms have superlinear time complexity. Figure 8 further shows that they can be slower than the naive algorithm. They are therefore impractically slow for massive dataset with billions of objects. The *kd*-tree technique is effective for small and very large distances, but not in-between.

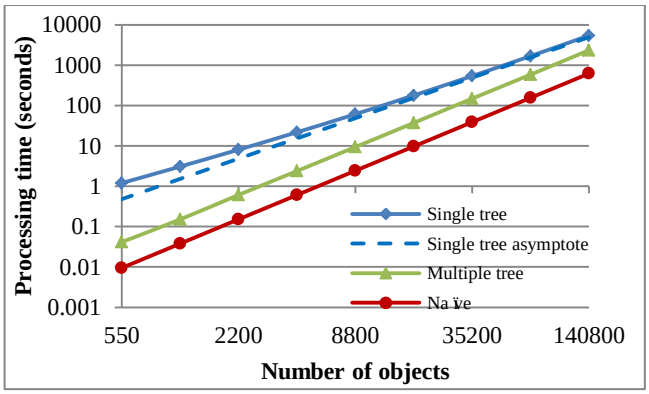


**Figure 8. Running time of the single-range *kd*-tree algorithm, multiple-range *kd*-tree algorithmm, and the naive algorithm (Figure 2). The single-range algorithm follows an $O(N^{5/3})$ asymptote. The multiple-range algorithm has quadratic complexity but it is faster than the single-tree algorithm in practice.**

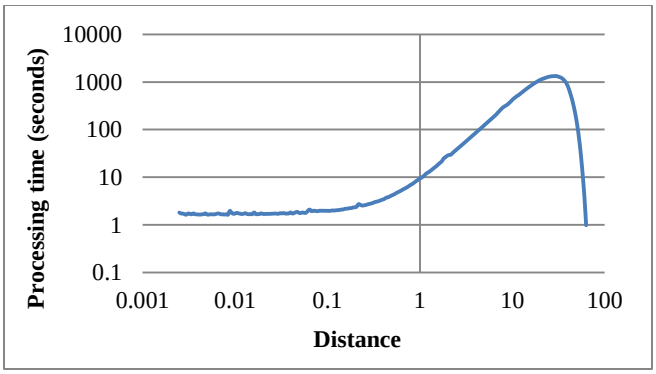


**Figure 9. The dependency of the running time on the distances for the single-range *kd*-tree algorithm. Note that the vertical axis is on logarithmic scale. This experiment is conducted on a dataset with 450 thousands objects. The results show that the processing of small and very large distance is much faster than the processing of medium distances.**

## 6. FRACTAL APPROXIMATION

The main advantage of the naive and *kd*-tree algorithms is that they provide exact results. On the downside, they are impractically slow for massive datasets. The processing of 4.5 million objects takes several days, and the time grows superlinearly with the number of objects, which means that processing a set with billions of objects would require very large time even on a supercomputer. We now consider the problem of developing fast approximate algorithms for computing correlation functions.

Belussi and Faloutsos used an approximate technique for computing fractal dimensions of point sets [2], which can be readily adapted to computating approximate correlation function in two main steps, as shown in Figure 10. First, we divide the space into a grid of cubic cells, iterate over all objects, and assign each object to the corresponding cell. Second, we count the number of objects in each cell.

**Output**: Counters $cf_1$, $cf_2$, …, $cf_M$

create array $s_i$, $i = 0$ **to** $M$
$s_0 = 0$
**for** $i = 1$ **to** $M$ **do**
  divide the space into a grid of cubic cells with side $d_i$
  we use $f_k$ to denote the number of objects falling in the $k^{\text{th}}$ cell
  $s_i = \lg(\sum_k (f_k)^2)$
  $D = (s_i - s_{i-1}) / (d_i - d_{i-1})$
  $cf_i = N\,(N-1)\,(\pi / 6)^{D/3}\,(2d_i)^D / 2$
**for** $i = M$ **to** 2 **do**
  $cf_i = cf_i - cf_{i-1}$

**Figure 10. Fractal algorithm.**

The underlying assumption is that the object density among nearby regions is similar, which leads to two approximation steps. First, when calculating how many objects are within distance $r$ of an object $p_i$, we do not consider a ball centered at $p_i$ with radius $r$, but instead use a cube centered at $p_i$ with side length $r \cdot (4\pi/3)^{1/3}$. Since the volume of the ball and the cube are the same, we expect that the numbers of objects within them is approximately the same. Second, rather than going over each object and counting the number of objects in their corresponding cubes, we use a set of global cubes (the respective *cells*) to represent all the cubes around objects, which greatly reduce the time complexity.

If we represent the grid by a hash table, the time complexity of the fractal algorithm is linear on the number of objects. We show the empirical running time in Figure 11. If we process each of the $M$ ranges separately, the overall time complexity is $O(M \cdot N)$. In Figure 12, we show the dependency of the running time on the number of ranges. We may further reduce the processing time to $O(N)$ by sorting the cells in $Z$-order [9].

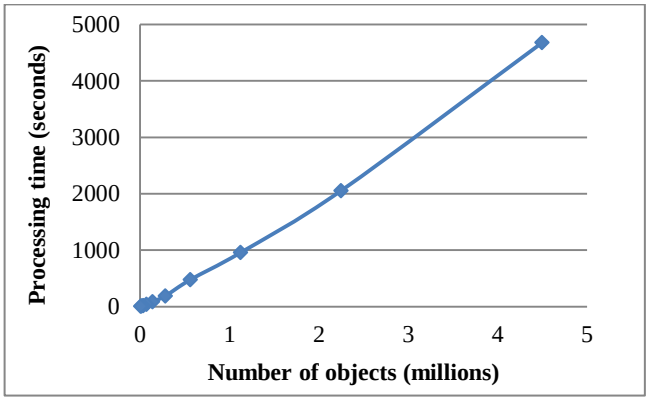


**Figure 11. Running time of the fractal approximation, which is linear on the number of objects.**

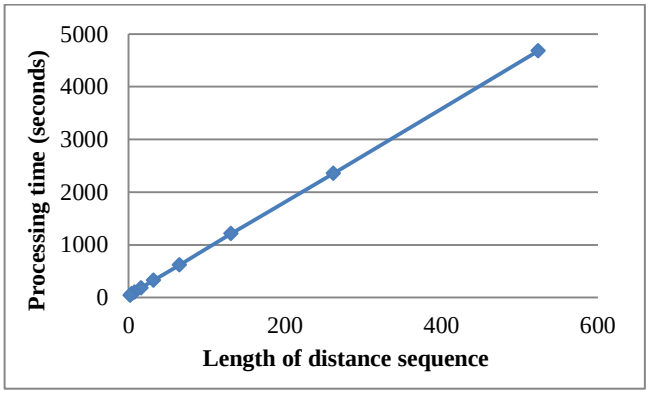


**Figure 12. Dependency of the running time on the length $M$ of the distance sequence.**

While this procedure is very fast, the resulting approximation is inaccurate. Belussi and Faloutsos reported that the error of the fractal procedure for estimating $\sum_{k=0}^{i} cf_k$ is in the 10–15% range [2]. Since the accuracy of the correlation-function histogram depends on the accuracy of computing specific values of $cf_k$, the resulting error is even greater, specifically around 40%.

## 7. SAMPLING

We next consider the application of sampling to approximate computation of correlation functions. Specifically, we randomly select $S$ objects from the original set and apply the naive algorithm to this smaller sample. To convert the resulting counters to the estimates of the counters for the original set, we multiple them by $(N/S)^2$ (Figure 13).

To get a more accurate estimate, we repeat the described procedure $T$ times, and then compute the mean $\mu(cf_i)$ and the standard deviation $\sigma(cf_i)$ for each $i$. According to the central limit

theorem, when $T$ is at least 30, these estimates follow the normal distribution, which allows determining confidence intervals.

**Output**: Mean and standard deviation of $cf_1, cf_2, \ldots, cf_M$

$T = 30$ // $T$ is the number of sampling iteration
create two dimensional array $a_{uv}$, $u = 1$ **to** $M$, $v = 1$ **to** $T$
**for** $u = 1$ **to** $M$ **do**
  **for** $v = 1$ **to** $T$ **do**
    $a_{uv} = 0$
**for** $t = 1$ **to** $T$ **do**
  randomly select $S$ objects $r_1, r_2, \ldots, r_S$ from $p_1, p_2, \ldots, p_N$
  **for** $u = 1$ **to** $S - 1$ **do**
    **for** $v = u + 1$ **to** $S$ **do**
      find $i$ such that $d_{i-1} \le dist(r_u, r_v) < d_i$; $a_{it}$ ++
  **for** $i = 1$ **to** $M$ **do** $a_{it} = a_{it} \cdot (N/S)^2$
**for** $i = 1$ **to** $M$ **do**
  $\mu(cf_i) = (a_{i1} + a_{i2} + \ldots + a_{iT}) / T$
  $\sigma(cf_i)^2 = ((a_{i1} - \mu(cf_i))^2 + (a_{i2} - \mu(cf_i))^2 + \ldots + (a_{iT} - \mu(cf_i))^2) / (T(T-1))$

**Figure 13. Sampling algorithm.**

In Figure 14, we show the relative error of the sampling algorithm, which is the absolute error of the approximation divided by the exact value. We have found that even a small sample provides a relatively accurate approximation for many distances. For example, if $S$ = 10,000, the sampling algorithm produces estimates with less than 1% error for distances from 2 to 50.

On the downside, the sampling error is much higher for small and very large distances. Generally, the error of estimating $cf_i$ (Figure 14) is in inverse proportion to the value of $cf_i$.

The overall running time of the sampling algorithm is the sum of (1) the time to retrieve samples from the original dataset and (2) the time to conduct the subsequent computation on the samples. We use a straightforward method to select samples from the original dataset, which makes a liner pass through the whole dataset, thus taking $O(N)$ time. For the dataset of 4.5 million objects, this sample selection takes 2.3 seconds. The time of applying the naive correlation function algorithm to the selected samples is $O(T \cdot S^2)$.

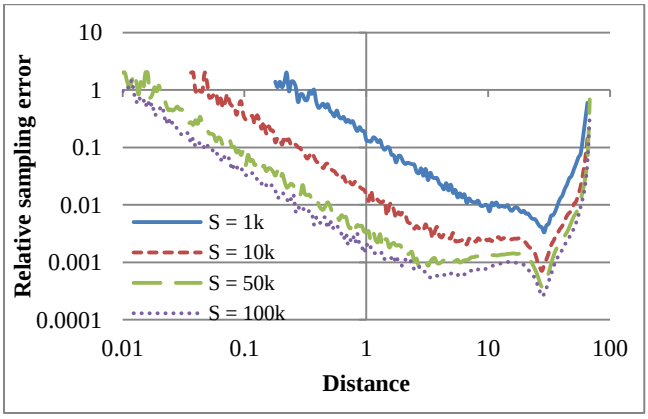


**Figure 14. The relative error of the sampling technique. The four lines represent experiments with samples of size 1,000, 10,000, 50,000 and 100,000.**

# 8. HYBRID TECHNIQUE

We have shown that the sampling method is fast but inaccurate for small and very large distances (Figure 14). On the other hand, the *kd*-tree technique is fast for small and very large distances (Figure 9). We can thus obtain better results by combining these two techniques. Specifically, we apply the *kd*-tree algorithm to small and very large distances, and the sampling technique to the distances in the middle.

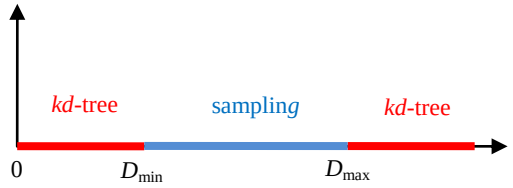


**Figure 15. Illustration of the hybrid technique.**

**Input**: Required maximum relative error *epsilon*
**Output**: Mean and standard deviation of $cf_1, cf_2, \ldots, cf_M$

- Find turning points $D_{min}$ and $D_{max}$ according to *epsilon*.
- For distances inside [$D_{min}$, $D_{max}$), select the number of sampled objects $S$ according to *epsilon*, and use the sampling algorithm in Section 7.
- For distances inside [$d_0$, $D_{min}$) and [$D_{max}$, $d_M$), use the *kd*-tree algorithm described in Section 5.

**Figure 16. Hybrid algorithm.**

The related parameter tuning involves setting the "turning points" $D_{min}$ and $D_{max}$, and determing the appropriate sample size $S$. We have selected these parameters based on empirical results, with the purpose of achieving the given accuracy in minimal running time.

In Figure 17, we compare the running time of the hybrid algorithm given different allowed approximation errors (*epsilon*).

In Figure 18, we compare the running time of the hybrid algorithm to other techniques on the set of 4.5 million objects. The proposed hybrid algorithm is much faster than the exact computation even if we limit the approximation error to 0.2%. When the running time of the hybrid algorithm is the same as that of the fractal algorithm, its error is much smaller.

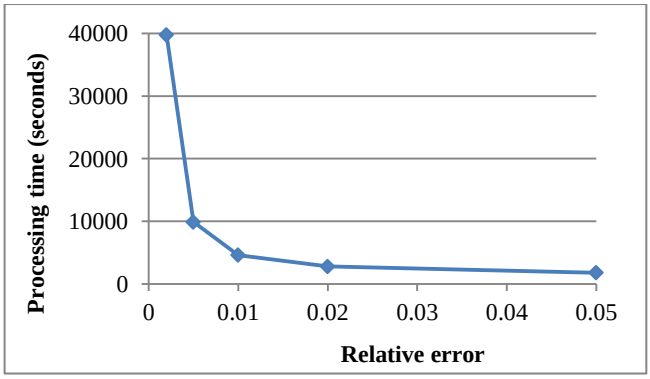


**Figure 17. Running time of the hybrid algorithm with different allowed errors. The five data points correspond to the errors of 0.2%, 0.5%, 1%, 2% and 5%.**

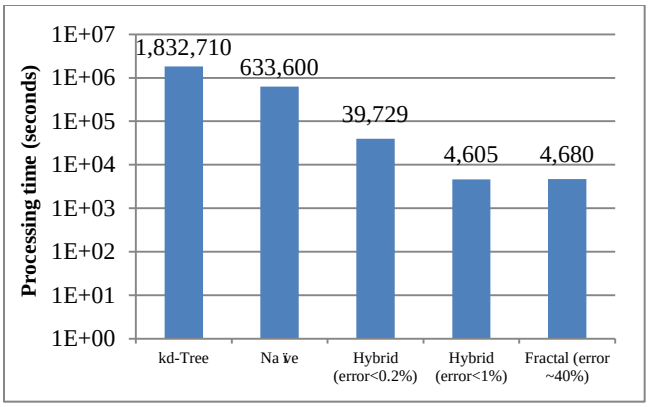


**Figure 18. Running time of the described techniques on the 4.5 million objects dataset.**

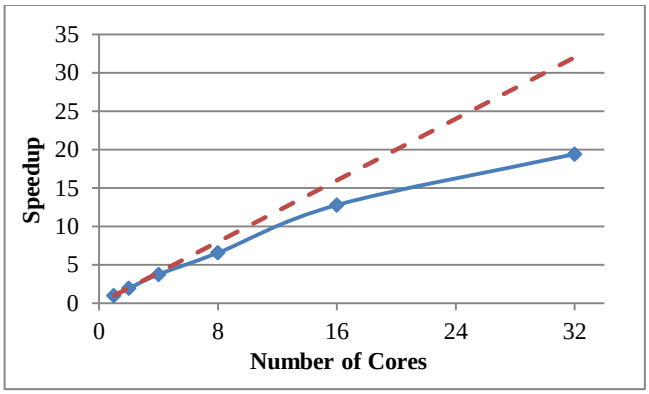


**Figure 19. Scalability of the distributed hybrid algorithm on the dataset with 1.1 billion objects. We set $d_0$ = 0.006, $d_M$ = 65, $C$ = 1.044, $M$ = 216, and allow the error of 1%. The dashed line represents the ideal case where the speedup equals the number of cores used in the computation.**

## 9. DISTRIBUTED HYBRID ALGORITHM

We have used Hadoop (http://hadoop.apache.org) to distribute the computation, which reduces the running time through parallel processing of samples.

To implement distributed processing, we have made one adjustment to the hybrid algorithm. Specifically, we no longer apply *kd*-tree algorithm to the set of all objects. Instead, we apply it to several samples from the overall dataset, and then compute the mean and standard deviation. The samples used in the *kd*-tree computation are larger than the samples used in the naive computation, which ensures sufficient accuracy for small and very large distances.

The Map-Reduce framework [3] readily supports the sampling operation. During the Map phase, we select random samples. During the Reduce phase, we process the selected samples in parallel.

In Table 1, we show the processing time with 32 cores, for the maximal allowed error of 1%. We have used two larger datasets: the Coyote Universe set [7], which contains 1.1 billion objects, and the DMKraken set, provided by our collaborators at the McWilliams Center for Cosmology at Carnegie Mellon, which contains 5 billion objects.

**Table 1. Time of computing correlation functions on a compute cluster with 32 cores, for the allowed error of 1%. We set $d_0$ = 0.006, $d_M$ = 65, $C$ = 1.044, and $M$ = 216 in this experiment.**

| Number of objects in the overall dataset (before sampling) | Running time (Seconds) |
|---|---|
| 4.5 million | 888 |
| 1.1 billion | 1529 |
| 5 billion | 2436 |

In Figure 19, we show the dependency of the speedup on the number of CPU cores.

## 10. CONCLUSIONS

We have presented a hybrid approximate algorithm for building the histogram of pairwise distances between celestial objects, which allows fast accurate approximation for dataset with billions of objects. Our implementations are available at http://www.pdl.cmu.edu/AstroDISC/disc-dist-code.shtml.

The related future work may involve improving the parameter tuning process. We will also explore other distributed computation frameworks, which may be more suitable for the hybrid algorithm.

## 11. ACKNOWLEDGMENTS

We thank Rupert Croft and Tiziana Di Matteo for helping us understand the related cosmological problem and providing feedback on our results. We also thank Julio Lopez, Christos Faloutsos, and Helen Mukomel for their comments and insights.

This work was sponsored in part by grants from Google, the Moore Foundation, National Science Foundation Open Cloud Consortium, the Petascale Data Storage Institute (PDSI), the McWilliams Center for Cosmology and the companies of the Parallel Data Laboratory Consortium (PDL).

## 12. REFERENCES

[1] Kevork N. Abazajian *et al*. The seventh data release of the Sloan Digital Sky Survey. Astrophysical Journal Supplement Series, 182(2), pages 543–558, 2009.

[2] Alberto Belussi and Christos Faloutsos. Estimating the selectivity of spatial queries using the 'correlation' fractal dimension. In Proceedings of the Twenty-First International Conference on Very Large Data Bases, pages 299–310, 1995.

[3] Jeff Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In Proceedings of the Symposium on Operating System Design and Implementation, 2004.

[4] Tiziana Di Matteo, Jörg Colberg, Volker Springel, Lars Hernquist, and Debora Sijacki. Direct cosmological simulations of the growth of black holes and galaxies. Astrophysical Journal, 676(2), 2008.

[5] Bin Fu, Kai Ren, Julio Lopez, Eugene Fink, and Garth Gibson. DiscFinder: A data-intensive scalable cluster finder for astrophysics. In Proceedings of the Twentieth ACM International Symposium on High Performance Distributed Computing, 2010.

[6] Alexander Gray and Andrew Moore. “N-body” problems in statistical learning. Advances in Neural Information Processing Systems 13, pages 521–527. MIT Press, 2000.

[7] Katrin Heitmann, Martin White, Christian Wagner, Salman Habib, and David Higdon. The Coyote Universe I: Precision determination of the nonlinear matter power spectrum. Astrophysical Journal, 715(1), 2008.

[8] Barry Lasker *et al*. The second-generation Guide Star Catalog: Description and properties. The Astrophysical Journal, 136, pages 735, 2008.

[9] G. M. Morton. A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing. IBM Germany Scientific Symposium Series, 1966.

[10] Jim Peebles. The Large Scale Structure of the Universe. Princeton University Press, 1980.