Fast Algorithms for Comprehensive N-point Correlation Estimates

William B. March
Georgia Institute of Technology
266 Ferst Dr.
Atlanta, GA, USA
march@gatech.edu

Andrew J. Connolly
University of Washington
3910 15th Ave. NE Seattle, WA, USA
ajc@astro.washington.edu

Alexander G. Gray
Georgia Institute of Technology
266 Ferst Dr.
Atlanta, GA, USA
agray@cc.gatech.edu

ABSTRACT

The n-point correlation functions (npcf) are powerful spatial statistics capable of fully characterizing any set of multidimensional points. These functions are critical in key data analyses in astronomy and materials science, among other fields, for example to test whether two point sets come from the same distribution and to validate physical models and theories. For example, the npcf has been used to study the phenomenon of dark energy, considered one of the major breakthroughs in recent scientific discoveries. Unfortunately, directly estimating the continuous npcf at a single value requires $O(N^n)$ time for N points, and n may be 2, 3, 4 or even higher, depending on the sensitivity required. In order to draw useful conclusions about real scientific problems, we must repeat this expensive computation both for many different scales in order to derive a smooth estimate and over many different subsamples of our data in order to bound the variance.

We present the first comprehensive approach to the entire n-point correlation function estimation problem, including fast algorithms for the computation at multiple scales and for many subsamples. We extend the current state-of-the-art tree-based approach with these two algorithms. We show an order-of-magnitude speedup over the current best approach with each of our new algorithms and show that they can be used together to obtain over 500x speedups over the state-of-the-art in order to enable much larger datasets and more accurate scientific analyses than were possible previously.

Categories and Subject Descriptors

J.2 [**Physical Sciences and Engineering**]: Astronomy; G.4 [**Mathematical Software**]: Algorithm design and analysis

Keywords

N-point Correlation Functions, Jackknife Resampling

KDD'12, August 12–16, 2012, Beijing, China.

1. INTRODUCTION

In this paper, we discuss a hierarchy of powerful statistics: the *n-point correlation functions*, which constitute a widely-used approach for detailed characterizations of multivariate point sets. These functions, which are analogous to the moments of a univariate distribution, can completely describe any point process and are widely applicable.

Applications in astronomy. The n-point statistics have long constituted the state-of-the-art approach in many scientific areas, in particular for detailed characterization of the patterns in spatial data. They are a fundamental tool in astronomy for characterizing the large scale structure of the universe [20], fluctuations in the cosmic microwave background [28], the formation of clusters of galaxies [32], and the characterization of the galaxy-mass bias [17]. They can be used to compare observations to theoretical models through perturbation theory [1, 6]. A high-profile example of this was a study showing large-scale evidence for dark energy [8] – this study was written up as the Top Scientific Breakthrough of 2003 in *Science* [26]. In this study, due to the massive potential implications to fundamental physics of the outcome, the accuracy of the n-point statistics used and the hypothesis test based on them were a considerable focus of the scientific scrutiny of the results – underscoring both the centrality of n-point correlations as a tool to some of the most significant modern scientific problems, as well as the importance of their accurate estimation.

Materials science and medical imaging. The materials science community also makes extensive use of the n-point correlation functions. They are used to form three-dimensional models of microstructure [13] and to characterize that microstructure and relate it to macroscopic properties such as the diffusion coefficient, fluid permeability, and elastic modulus[31, 30]. The n-point correlations have also been used to create feature sets for medical image segmentation and classification [22, 19, 2].

Generality of npcf. In addition to these existing applications, the n-point correlations are completely general. Thus, they are a powerful tool for any multivariate or spatial data analysis problem. Ripley [23] showed that any point process consisting of multidimensional data can be completely determined by the distribution of counts in cells. The distribution of counts in cells can in turn be shown to be completely determined by the set of n-point correlation functions [20]. While ordinary statistical moments are defined in terms of the expectation of increasing powers of X, the n-point functions are determined by the cross-correlations

of counts in increasing numbers of nearby regions. Thus, we have a sequence of increasingly complex statistics, analogous to the moments of ordinary distributions, with which to characterize any point process and which can be estimated from finite data. With this simple, rigorous characterization of our data and models, we can answer the key questions posed above in one statistical framework.

Computational challenge. Unfortunately, directly estimating the n-point correlation functions is extremely computationally expensive. As we will show below, estimating the npcf potentially requires enumerating all n-tuples of data points. Since this scales as $O(N^n)$ for N data points, this is prohibitively expensive for even modest-sized data sets and low-orders of correlation. Higher-order correlations are often necessary to fully understand and characterize data [32]. Furthermore, the npcf is a continuous quantity. In order to understand its behavior at all the scales of interest for a given problem, we must repeat this difficult computation many times. We also need to estimate the variance of our estimated npcf. This in general requires a resampling method in order to make the most use of our data. We must therefore repeat the $O(N^n)$ computation not only for many scales, but for many different subsamples of the data.

In the past, these computational difficulties have restricted the use of the n-point correlations, despite their power and generality. The largest 3-point correlation estimation thus far for the distribution of galaxies used only approximately 10^5 galaxies [16]. Higher-order correlations have been even more restricted by computational considerations.

Large data. Additionally, data sets are growing rapidly, and spatial data are no exception. The National Biodiversity Institute of Costa Rica has collected over 3 million observations of tropical species along with geographical data [11]. In astronomy, the Sloan Digital Sky Survey [25] spent eight years imagining the northern sky and collected tens of terabytes of data. The Large Synoptic Survey Telescope [14], scheduled to come online later this decade, will collect as much as 20 terabytes *per night* for ten years. These massive datasets will render n-point correlation estimation even more difficult without efficient algorithms.

Our contributions. These computational considerations have restricted the widespread use of the npcf in the past. We introduce two new algorithms, building on the previous state-of-the-art algorithm [9, 18], to address this entire computational challenge. We present the first algorithms to efficiently overcome two important computational bottlenecks.

- We **estimate the npcf at many scales simultaneously**, thus allowing smoother and more accurate estimates.
- We **efficiently handle jackknife resampling** by sharing work across different parts of the computation, allowing more effective variance estimation and more accurate results.

For each of these problems, we present new algorithms capable of sharing work between different parts of the computation. We prove a theorem which allows us to eliminate a critical redundancy in the computation over multiple scales. We also cast the computation over multiple subsamples in a novel way, allowing a much more efficient algorithm. Each of these new ideas allows an order-of-magnitude speedup over the existing state of the art. These algorithms are therefore able to render n-point correlation function estimation tractable for many large datasets for the first time by allowing N to increase and allow more sensitive and accurate scientific results for the first time by allowing multiple scales and resampling regions. Our work is the first to deal directly with the full computational task.

Overview. In Section 2, we define the n-point correlation functions and describe their estimators in detail. This leads to the $O(J \cdot M \cdot N^n)$ computational challenge mentioned above. We then introduce our two new algorithms for the entire npcf estimation problem in Section 3. We show experimental results in Section 4. Finally, we conclude in Section 5 by highlighting future work and extensions of our method.

1.1 Related Work

Due to the computational difficulty associated with estimating the full npcf, many alternatives to the full npcf have been developed, including those based on nearest-neighbor distances, quadrats, Dirichlet cells, and Ripley's K function (and related functions) (See [24] and [3] for an overview and further references). Counts-in-cells [29] and Fourier space methods are commonly used for astronomical data. However, these methods are generally less powerful than the full npcf. For instance, the counts-in-cells method cannot be corrected for errors due to the edges of the sample window. Fourier transform-based methods suffer from ringing effects and suboptimal variance. See [27] for more details.

Since we deal exclusively with estimating the exact npcf, we only compare against other methods for this task. The existing state-of-the-art methods for exact npcf estimation use multiple space-partitioning trees to overcome the $O(N^n)$ scaling of the n-point point estimator. This approach was first introduced in [9, 18]. It has been parallelized using the Ntropy framework [7]. We present the serial version here and leave the description of our ongoing work on parallelizing our approach to future work.

2. N-POINT CORRELATION FUNCTIONS

We now define the n-point correlation functions. We provide a high-level description; for a more thorough defintion, see [20, 27]. Once we have given a simple description of the npcf, we turn to the main problem of this paper: the computational task of estimating the npcf from real data. We give several common estimators for the npcf, and highlight the underlying counting problem in each. We also discuss the full computational task involved in a useful estimate of the npcf for real scientific problems.

Problem setting. Our data are drawn from a *point process*. The data consist of a set of points D in a subset of $\mathbb{R}^d$. Note that we are not assuming that the locations of individual points are independent, just that our data set is a fair sample from the underlying ensemble. We assume that distant parts of our sample window are uncorrelated, so that by averaging over them, we can approximate averages over multiple samples from the point process.

Following standard practice in astronomy, we assume that the process is homogeneous and isotropic. Note that the n-point correlations can be defined both for more general point processes and for continuous random fields. The estimators for these cases are similar to the ones described below and can be improved by similar algorithmic techniques.

Defining the npcf. We now turn to an informal, intuitive description of the hierarchy of n-point correlations. Since we have assumed that properties of the point process are translation and rotation invariant, the expected number of points in a given volume is proportional to a global density ρ. If we consider a small volume element dV, then the probability of finding a point in dV is given by:

$$dP = \rho dV \tag{1}$$

with dV suitably normalized. If the density ρ completely characterizes the process, we refer to it as a Poisson process.

Two-point correlation. The assumption of homogeneity and isotropy does not require the process to lack structure. The positions of points may still be correlated. The joint probability of finding objects in volume elements dV_1 and dV_2 separated by a distance r is given by:

$$dP_{12} = \rho^2 dV_1 dV_2 (1 + \xi(r)) \tag{2}$$

where the dV_i are again normalized. The two-point correlation $\xi(r)$ captures the increased or decreased probability of points occurring at a separation r. Note that the 2-point correlation is characterized by a single scale r and is a continuously varying function of this distance.

Three-point correlation. Higher-order correlations describe the probabilities of more than two points in a given configuration. We first consider three small volume elements, which form a triangle (See Fig. 1(b)). The joint probability of simultaneously finding points in volume elements dV_1, dV_2, and dV_3, separated by distances r_{12}, r_{13}, and r_{23}, is given by:

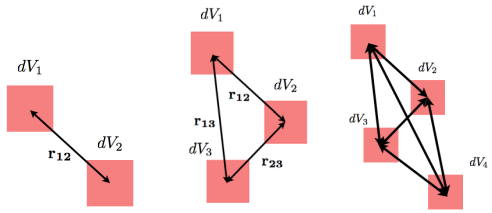
$$\begin{aligned} dP_{123} &= \rho^3 dV_1 dV_2 dV_3 \left[1 + \xi(r_{12}) + \xi(r_{23}) + \xi(r_{13}) \right. \\ &\quad \left. + \zeta(r_{12}, r_{23}, r_{13})\right] \end{aligned} \tag{3}$$

The quantity in square brackets is sometimes called the *complete (or full) 3-point correlation function* and ζ is the *reduced 3-point correlation function*. We will often refer to ζ as simply the 3-point correlation function, since it will be the quantity of computational interest to us. Note that unlike the 2-point correlation, the 3-point correlation depends both on distance and configuration. The function varies continuously both as we increase the lengths of the sides of the triangle and as we vary its shape, for example by fixing two legs of the triangle and varying the angle between them.

Higher-order correlations. Higher-order correlation functions (such as the 4-point correlation in Fig. 1(c)) are defined in the same fashion. The probability of finding n-points in a given configuration can be written as a summation over the n-point correlation functions. For example, in addition to the reduced 4-point correlation function η, the complete 4-point correlation depends on the six 2-point terms (one for each pairwise distance), four 3-point terms (one for each triple of distances), and three products of two 2-point functions. The reduced four-point correlation is a function of all six pairwise distances. In general, we will denote the n-point correlation function as $\xi^{(n)}(\cdot)$, where the argument is understood to be $\binom{n}{2}$ pairwise distances. We refer to this set of pairwise distances as a *configuration*, or in the computational context, as a *matcher* (see below).

2.1 Estimating the NPCF

We have shown that the n-point correlation function is a fundamental spatial statistic and have sketched the definitions of the n-point correlation functions in terms of the underlying point process. We now turn to the central task of this paper: the problem of estimating the n-point correlation from real data. We describe several commonly used estimators and identify their common computational task.



(a) 2-point (b) 3-point (c) 4-point

Figure 1: Visual interpretation of the n-point correlation functions.

We begin by considering the task of computing an estimate $\widehat{\xi}^{(n)}(\mathbf{r})$ for a given configuration. For simplicity, we consider the 2-point function first. Recall that $\xi(r)$ captures the increased (or decreased) probability of finding a pair of points at a distance r over finding the pair in a Poisson distributed set. This observation suggests a simple Monte Carlo estimator for $\xi(r)$. We generate a random set of points R from a Poisson distribution with the same (sample) density as our data and filling the same volume. We then compare the frequency with which points appear at a distance close to r in our data versus in the random set.

Simple estimator. Let $\mathrm{DD}(r)$ denote the number of pairs of points (x_i, x_j) in our data, normalized by the total number of possible pairs, whose pairwise distance $d(x_i, x_j)$ is close to r (in a way to be made precise below). Let $\mathrm{RR}(r)$ be the number of points whose pairwise distances are in the same interval (again normalized) from the random sample (DD stands for data-data, RR for random-random). Then, a simple estimator for the two-point correlation is [20, 27]:

$$\hat{\xi}(r) = \frac{\mathrm{DD}(r)}{\mathrm{RR}(r)} - 1 \tag{4}$$

This estimator captures the intuitive behavior we expect. If pairs of points at a distance near r are more common in our data than in a completely random (Poisson) distribution, we are likely to obtain a positive estimate for ξ.

This simple estimator suffers from suboptimal variance and sensitivity to noise. The Landy-Szalay estimator [12]

$$\hat{\xi}(r) = \frac{DD(r) - 2DR(r) + RR(r)}{RR(r)} \tag{5}$$

overcomes these difficulties. Here the notation $DR(r)$ denotes the number of pairs (x_i, y_j) at a distance near r where x_i is from the data and y_j is from the Poisson sample (DR – data-random pairs). Other widely used estimators use the same underlying quantities – pairs of points satisfying a given distance constraint [12, 10].

Three-point estimator. The 3-point correlation function depends on the pairwise distances between three points, rather than a single distance as before. We will therefore need to specify three distance constraints, and estimate the function for that configuration. The Landy-Szalay estimator for the 2-point function can be generalized to any value of n, and retains its improved bias and variance [29]. We

again generate points from a Poisson distribution, and the estimator is also a function of quantities of the form $D^{(n)}$, or $D^{(i)}R^{(n-i)}$. These refer to the number of unique triples of points, all three from the data or two from the data and one from the Poisson set, with the property that their three pairwise distances lie close to the distances in the matcher.

All estimators count tuples of points. Any n-point correlation can be estimated using a sum of counts of n-tuples of points of the form $D^{(i)}R^{(n-i)}(\mathbf{r})$, where i ranges from zero to n. The argument is a vector of distances of length $\binom{n}{2}$, one for each pairwise distance needed to specify the configuration. We count unique tuples of points whose pairwise distances are close to the distances in the matcher in some ordering.

2.2 The Computational Task

Note that all the estimators described above depend on the same fundamental quantities: the number of tuples of points from the data/Poisson set that satisfy some set of distance constraints. Thus, our task is to compute this number given the data and a suitably large Poisson set. Enumerating all n-tuples requires $O(N^n)$ work for N data points. Therefore, we must seek a more efficient approach. We first give some terminology for our algorithm description below. We then present the entire computational task.

Matchers. We specify an n-tuple with $\binom{n}{2}$ constraints, one for each pairwise distance in the tuple. We mentioned above that we count tuples of points whose pairwise distances are "close" to the distance constraints. Each of the $\binom{n}{2}$ pairwise distance constraints consists of a lower and upper bound: $r_{ij}^{(l)}$ and $r_{ij}^{(u)}$. In the context of our algorithms, we refer to this collection of distance constraints $\mathbf{r}$ as a *matcher*. We refer to the entries of the matcher as $r_{ij}^{(l)}$ and $r_{ij}^{(u)}$, where the indices i and j refer to the volume elements introduced above (Fig. 1). We sometimes refer to an entry as simply r_{ij}, with the upper and lower bounds being understood.

Satisfying matchers. Given an n-tuple of points and a matcher $\mathbf{r}$, we say that the tuple *satisfies* the matcher if there exists a permutation of the points such that each pairwise distance does not violate the corresponding distance constraint in the matcher. More formally:

Definition 1. Given an n-tuple of points $(p_1, \ldots, p_n)$ and a matcher $\mathbf{r}$, we say that the tuple **satisfies** the matcher if there exists (at least one) permutation σ of $[1, \ldots, n]$ such that

$$r_{\sigma(i)\sigma(j)}^{(l)} < \|p_i - p_j\| < r_{\sigma(i)\sigma(j)}^{(u)} \tag{6}$$

for all indices $i, j \in [1, \ldots n]$ such that $i < j$.

The computational task. We can now formally define our basic computational task:

Definition 2. **Computational Task 1: Compute the counts of tuples $D^{(i)}R^{(j)}(\mathbf{r})$.** Given a data set D, random set R, and matcher $\mathbf{r}$, and $0 \leq i \leq n$, compute the number of unique n-tuples of points with i points from D and $n-i$ points from R, such that the tuple satisfies the matcher.

Computing these quantities directly requires enumerating all unique n-tuples of points, which takes $O(N^n)$ work and is prohibitively slow for even two-point correlations.

Multiple matchers. The estimators above give us a value $\widehat{\xi}^{(n)}(\mathbf{r})$ at a single configuration. However, the n-point correlations are continuous quantities. In order to fully characterize them, we must compute estimates for a wide range of configurations. In order to do this, we must repeat the computation in Defn. 2 for many matchers, both of different scales and configurations.

Definition 3. **Computational Task 2: Multiple matchers.** Given a data set D, random set R, and a collection of M matchers $\{\mathbf{r}_m\}$, compute $D^{(i)}R^{(j)}(\mathbf{r}_m)$ for each $1 \leq m \leq M$.

This task requires us to repeat Task 1 M times, where M controls the smoothness of our overall estimate of the npcf and our quantitative picture of its overall behavior. Thus, it is generally necessary for M to be large.

Resampling. Simply computing point estimates of any statistic is generally insufficient for most scientific applications. We also need to bound the variance of our estimator and compute error bars. In general, we must make the largest possible use of the available data, rather than withholding much of it for variance estimation. Thus, a resampling method is necessary.

Jackknife resampling is a widely used variance estimation method and is popular with astronomical data [15]. It is also used to study large scale structure by identifying variations in the npcf across different parts of the sample window [16]. We divide the data set into subregions. We eliminate each region from the data in turn, then compute our estimate of the npcf. We repeat this for each subset, and use the resulting estimates to bound the variance. This leads to our third and final computational task.

Definition 4. **Computational Task 3: Jackknife resampling.** We are given a data set D, random set R, a set of M matchers $\mathbf{r}_m$, and a partitioning of D into J subsets D_k. For each $1 \leq k \leq J$, construct the set $D_{(-k)} = D/D_k$. Then, compute $D_{(-k)}^{(i)}R^{(j)}(\mathbf{r})$.

This task requires us to repeat Task 1 J times on sets of size $D - D/J$. Note that J controls the quality of our variance estimation, with larger values necessary for a better estimate.

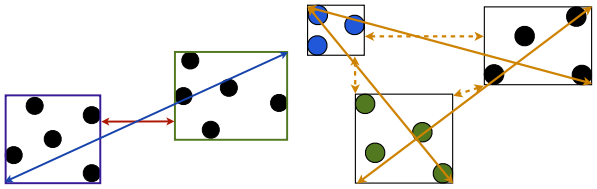
The complete computational task. We can now identify the complete computational task for n-point correlation estimation. Given our data and random sets, a collection of M matchers, and a partitioning of the data into J subregions, we must perform Task 3. This in turn requires us to perform Task 2 J times. Each iteration of Task 2 requires M computations of Task 1. Therefore, the entire computation requires $O(J \cdot M \cdot N^n)$ time if done in a brute-force fashion. In the next section, we describe our algorithmic approach to simultaneously computing all three parts of the computation, thus allowing significant savings in time.

3. ALGORITHMS

We have identified the full computational task of n-point correlation estimation. We now turn to our new algorithm. We begin by addressing previous work on efficiently computing the counts $D^{(i)}R^{(j)}(\mathbf{r})$ described above (Computational Task 1.) We first describe the multi-tree algorithm for computing these counts. We then give our new algorithm for directly solving computations with multiple matchers (Computational Task 2) and our method for efficiently computing counts for resampling regions (Computational Task 3).

3.1 Basic Algorithm

We build on previous, tree-based algorithms for the n-point correlation estimation problem [9, 18]. The key idea is to employ multiple kd-trees to improve on the $O(N^n)$ scaling of the brute-force approach.



(a) Comparing two nodes. (b) Comparing three nodes.

Figure 2: Computing node-node bounds for pruning.

***kd*-trees.** The kd-tree [21, 5] is a binary space partitioning tree which maintains a bounding box for all the points in each node. The root consists of the entire set. Children are formed recursively by splitting the parent's bounding box along the midpoint of its largest dimension and partitioning the points on either side. We can build a kd-tree on both the data and random sets as a pre-processing step. This requires only $O(N \log N)$ work and $O(N)$ space.

We employ the bounding boxes to speed up the naive computation by using them to identify opportunities for *pruning*. By computing the minimum and maximum distances between a pair of kd-tree nodes (Fig. 2), we can identify cases where it is impossible for any pair of points in the nodes to satisfy the matcher.

Dual-tree algorithm. For simplicity, we begin by considering the two-point correlation estimation (Alg. 1). Recall that the task is to count the number of unique pairs of points that satisfy a given matcher. We consider two tree nodes at a time, one from each set to be correlated. We compute the upper and lower bounds on distances between points in these nodes using the bounding boxes. We can then compare this to the matcher's lower and upper bounds. If the distance bounds prove that all pairs of points are either too far or too close to possibly satisfy the matcher, then we do not need to perform any more work on the nodes. We can thus *prune* all child nodes, and save $O(|T_1| \cdot |T_2|)$ work. If we cannot prune, then we split one (or both) nodes, and recursively consider the two (or four) resoling pairs of nodes. If our recursion reaches leaf nodes, we compare all pairs of points exhaustively.

We begin by calling the algorithm on the root nodes of the tree. If we wish to perform a DR count, we call the algorithm on the root of each tree. Note also that we only want to count unique pairs of points. Therefore, we can prune if T_2 comes before T_1 in an in-order tree traversal. This ensures that we see each pair of points at most once.

Multi-tree algorithm. We can extend this algorithm to the general n case. Instead of considering pairs of tree nodes, we compare an n-tuple of nodes in each step of the algorithm. This *multi-tree* algorithm uses the same basic idea – use bounding information between pairs of tree nodes to identify sets of nodes whose points cannot satisfy the matcher. We need only make two extensions to Alg. 1. First, we must do more work to determine if a particular

Algorithm 1 DualTree2pt (Tree node T_1, Tree node T_2, matcher $\mathbf{r}$)

```
if T1 and T2 are leaves then
  for all points p1 ∈ T1, p2 ∈ T2 do
    if r12^(l) < ||p1 − p2|| < r12^(u) then
      result += 1
    end if
  end for
else if dmin(T1, T2) > r12^(u) or dmax(T1, T2) < r12^(l) then
  Prune
else
  DualTree2pt(T1.left, T2.left)
  DualTree2pt(T1.left, T2.right)
  DualTree2pt(T1.right, T2.left)
  DualTree2pt(T1.right, T2.right)
end if
```

Algorithm 2 MultiTreeNpt (Tree node $T_1, \ldots$, Tree node T_n, matcher $\mathbf{r}$)

```
if all nodes Ti are leaves then
  for all points p1 ∈ T1, ..., points pn ∈ Tn do
    if TestPointTuple(p1, ..., pn, r) then
      result += 1
    end if
  end for
else if not TestNodeTuple(T1, ..., Tn, r) then
  Prune
else
  Let Ti be the largest node
  MultiTreeNpt(T1, ..., Ti.left, ..., Tn, r)
  MultiTreeNpt(T1, ..., Ti.right, ..., Tn, r)
end if
```

tuple of points satisfies the matcher. We accomplish this in Alg. 3 by iterating over all permutations of the indices. Each permutation of indices corresponds to an assignment of pairwise distances to entries in the matcher. We can quickly check if this assignment is valid, and we only count tuples that have at least one valid assignment. The second extension is a similar one for checking if a tuple of nodes can be pruned (Alg. 4). We again iterate through all permutations and check if the distance bounds obtained from the bounding boxes fall within the upper and lower bounds of the matcher entry. As before, for an $D^{(i)}R^{(j)}$ count, we call the algorithm on i copies of the data tree root and j copies of the random tree root.

3.2 Multi-Matcher Algorithm

The algorithms presented above all focus on computing individual counts of points – *i.e.* Computational Task 1, from Sec. 2.2. This approach improves the overall dependence on the number of data points – N – and the order of the correlation – n. However, this does nothing for the other two parts of the overall computational complexity. We now turn to our novel algorithm to count tuples for many matchers simultaneously, thus addressing Computational Task 2.

Intuitively, computing counts for multiple matchers will repeat many calculations. For simplicity, consider the two-point correlation case illustrated in Fig. 3(a). We must count the number of pairs that satisfy two matchers, r_1 and r_2 (assume that the upper and lower bounds for each are very

Algorithm 3 TestPointTuple (points $p_1, \ldots, p_n$, matcher $\mathbf{r}$)

```
    mark all permutations σ of [1, ..., n] as valid
    for all indices i, j ∈ [1, n], i < j do
      d_ij = ||p_i − p_j||
      for all permutations σ do
5:      if r^(l)_{σ(i)σ(j)} > d_ij or r^(u)_{σ(i)σ(j)} < d_ij then
          mark σ as invalid
        end if
      end for
    end for
10: if At least one permutation σ is marked valid then
      return true
    else
      return false
    end if
```

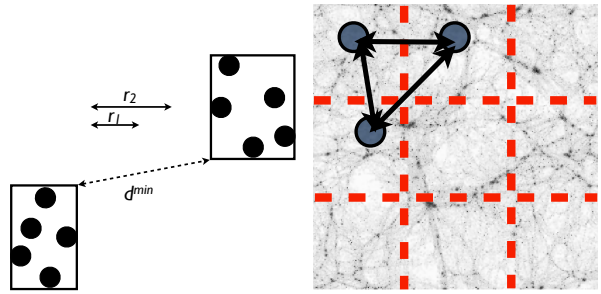
Algorithm 4 TestNodeTuple (Tree node $T_1, \ldots,$ Tree node T_n, matcher $\mathbf{r}$)

```
    mark all permutations σ of [1, ..., n] as valid
    for all indices i ∈ [1, n − 1] and indices j > i do
      d^max_ij = d^max(T_i, T_j);  d^min_ij = d^min(T_i, T_j)
      for all permutations σ do
5:      if r^(l)_{σ(i)σ(j)} > d^max_ij or r^(u)_{σ(i)σ(j)} < d^min_ij then
          mark σ as invalid
        end if
      end for
    end for
10: if At least one permutation σ is marked valid then
      return true
    else
      return false
    end if
```

close to the distances shown for simplicity). If we first perform the computation for r_1 using the dual-tree algorithm (Alg. 1), at some point we will consider the pair of nodes in the figure. We compute the lower bound distance from their bounding boxes and see that we can prune. After this algorithm has finished, we again use Alg. 1, this time with matcher r_2. We may again encounter the same pair of nodes, which requires us to compute the same distance bounds and make the same pruning decision. If we can consider the two matchers simultaneously, we can make the pruning decision for both and save the unnecessary work. We can also save work in the base case. We need only compute the distance between each pair of points once (if we have not been able to prune the pair completely). We can then immediately identify which matcher(s), if any, it satisfies.

Two-point case. In the two-point case, this improvement is straightforward, and has already been shown in [9]. We must only modify Alg. 3 and Alg. 4 to consider a collection of matchers. If the matchers are regularly spaced, then we can identify which (if any) the node or point pair may satisfy in constant time. For arbitrary matchers, we can sort them and find possible matching pairs in logarithmic time.

General case. The general n-point case requires more caution. The presence of permutations in Alg. 4 makes the straightforward approach mentioned above more difficult. Each pair of nodes in each permutation will possibly satisfy



(a) Pruning for multiple matchers.

(b) Resampling regions.

Figure 3: Our two new algorithmic ideas.

different matchers. It becomes prohibitively expensive to determine which, if any, matchers may be satisfied.

Instead, we can avoid searching all permutations entirely. We make use of the following observation, proven below: if the distance bounds violate a matcher in a particular order, they will violate it in all possible permutations.

THEOREM 3.1. *Given a matcher* $\mathbf{r}$*, let* $\left\{u_i : 1 \le i \le \binom{n}{2}\right\}$ *be the set of upper bounds* $r^{(u)}_{ij}$*, in the matcher, sorted in ascending order. Let* $\left\{d^l_i : 1 \le i \le \binom{n}{2}\right\}$ *be the set of lower-bound distances* d^{min}_{ij} *obtained from the bounding boxes of* n *tree nodes, also sorted in ascending order. Then, if* $d^l_i > u_i$ *for any* i*, no* n*-tuple of points from the nodes can satisfy the matcher.*

PROOF. Any permutation σ of the bounding boxes assigns each lower-bound distance d^l_i to some upper bound distance in the matcher $u_{\sigma(i)}$. In order for this permutation to work, we need that $d^l_i < u_{\sigma(i)}$ for all i. Let i' be the index such that $d^l_{i'} > u_{i'}$ in the condition of the theorem. Then, any permutation must map i' to an index j. If $j < i'$, then $d^l_{i'} > u_j = u_{\sigma(i')}$. If $j > i'$, then $\sigma(j) < i'$. Therefore, $d^l_j > u_{\sigma(j)}$. Therefore, no permutation can satisfy the matcher. □

We can easily arrive at a similar conclusion regarding upper bounds.

Multi-matcher pruning. Using these observations, we can improve the pruning rule in Alg. 4. With each n-tuple of nodes, we store the upper and lower bound distances, sorted in ascending order. We can compare these with the sorted distances in the matcher, using the observations above. If any lower bound distance is greater than a corresponding maximum matcher distance, we prune. When we recurse, we compute the $n-1$ new upper and lower bounds on node distances and update the sorted lists.

The total running time for pruning checks is therefore reduced from something proportional to $n!$ to a single loop of size $\binom{n}{2}$ along with comparable overhead from updating the sorted lists of bounds. Unfortunately, for common values of n (2 and 3), this is not a significant advantage. However, this approach makes a much greater difference in the multi-matcher version of the algorithm.

Specifying multiple matchers. We specify a set of matchers $\mathbf{r}_m$ by giving a minimum and maximum range, $r^{\min}_{ij}, r^{\max}_{ij}$ for each of its $\binom{n}{2}$ dimensions along with a num-

Algorithm 5 MultiTestNodeTuple (Tree node $T_1, \ldots,$ Tree node T_n, matcher minima $r_i^{\min}$, matcher maxima $r_i^{\max}$)

```
Compute node-node bounds l_i and u_i and sort
for i = 1 : (n choose 2) do
  if l_i > r_i^max or u_i < r_i^min then
    return false
  end if
end for
return true
```

Algorithm 6 EfficientResampling (Data sets $D_i : 1 \leq i \leq J$, matcher $\mathbf{r}$)

```
Construct a tree T_i on each D_i
for all unique n tuples of indices (i_1, ..., i_n) do
  result = MultiTreeNpt(T_{i_1}, ..., T_{i_n}, r)
  for all j ∉ {i_i, ..., i_n} do
    results[j] += result
  end for
end for
```

ber of equally-sized bins b_{ij} for each dimension. Each possible choice of bins, one from each dimension, then forms a single matcher. The total set of matchers consists of $\prod b_{ij}$ matchers, where the product runs over all dimensions of the matcher.

We can then provide a multi-matcher version of Alg. 4. We sort the minimum and maximum matcher ranges in ascending order. We can then compare these against the bounds from the nodes' bounding boxes.

We extend Alg. 3 by marking a permutation invalid if the point-point distance is greater than the maximum matcher distance or less than the minimum. If the distance falls in between, we can identify which bin contains it in constant time. We store a vector of possible bins for each permutation, and if a permutation has a valid bin for each dimension, we can increment our count for that dimension. Note that we still only count each tuple at most once.

Alternative matcher specifications. Our specification of multiple matchers may seem somewhat restrictive. On the one hand, our framework allows for a wide range of possible configurations by allowing each dimension to be specified separately. However, we have assumed that no bins overlap and that the thickness of each bin is fixed. Neither of these restrictions are fundamental to our method. We are currently developing an extension which will allow overlapping bins and varying thicknesses, such as commonly occurs in three point computations. This modification requires only limited extensions to the pruning rule and base case.

3.3 Efficient Resampling Algorithm

We now turn to our second new algorithm – a method to efficiently compute counts for resampling regions (Computational Task 3). Recall from Sec. 2.2 that we are given a partitioning of the data set into subsets D_i. We then construct i data sets $D_{(-i)} = \{i \in D | i \notin D_i\}$. We now compute counts which satisfy the matcher for each set $D^{(-i)}$. Assuming that the subregions each contain roughly the same number of points, we see that for J subregions and an n-point estimation algorithm which requires $T(N)$ time for N data points, we require $O(JT(N - N/J))$ time for each matcher. Instead, we can rearrange this computation to share work between different subcomputations.

Redundant work. The intuition for our new approach is shown in Fig. 3(b). Assume we are computing a count of the form $DDD(\mathbf{r})$ and that the three points shown in the figure satisfy the matcher. For all but the three regions in which the tuple's points lie, we will need to do the work to find and count the tuple since it factors into the total DDD count for all the $D_{(-i)}$. Therefore, we will perform all the work needed to find this tuple $J - 3$ times.

New algorithmic strategy. We can avoid this extra work by working with the subsets D_i directly. For instance, if we compute the count $D_iD_iD_i(\mathbf{r})$, this result will appear in the counts for all $D_{(-j)}D_{(-k)}D_{(-l)}(\mathbf{r})$. We can then compute this count once and add its intermediate result into the $J - 1$ final results. In general, we consider n distinct sets D_i, compute the number of tuples from them that satisfy the matcher, and add that intermediate count into the result for each $D_{(-j)}$ that does not appear in the computation. We show this approach in Alg. 6. We maintain an array of results of length J, where the j^{th} entry corresponds to the count $D_{(-j)} \cdots D_{(-j)}(\mathbf{r})$. This method can be easily extended to include random sets by including the random set as one of the D_i in the input to Alg. 6.

For J resampling regions, each containing roughly N/J points, this algorithm requires $O(\binom{J}{n} \cdot T(N/J))$ time, where $T(N)$ is the time required for a single n-point correlation computation on N points. The naive version loops over all resampling regions, so it requires $O(J \cdot T(N))$ time. While the combinatorial dependence on n may seem to be a disadvantage, it is easily made up for by the reduced problem size for each n-point computation. As we will show in our experimental results, this method provides a considerable speedup.

We have presented two new algorithms which, along with the previous multi-tree algorithm [9, 18], address the entire computational problem of estimating n-point correlations. Our two algorithms can be used in conjunction for maximum speedups. We simply provide the set of matchers described in Sec. 3.2 to the efficient resampling algorithm (Alg. 6). We now turn to some empirical results for our new algorithms.

4. EXPERIMENTAL RESULTS

Algorithms. We have presented two new algorithms for n-point correlation estimation. We show results with our new *efficient resampling* algorithm (Alg. 6) and with a naive resampling ($O(J \cdot T(N))$) approach, which simply loops over resampling regions. We also show results with our *multi-matcher* method and a naive-matcher $O(M \cdot T(N))$ algorithm which simply loops over matchers and calls Alg. 2. We therefore present results for each new method alone, and in combination. We also show results for the original multi-tree algorithm [18], which we call the single-matcher, naive-resampling algorithm. We have also estimated runtimes for the brute-force $O(J \cdot M \cdot N^n)$ algorithm.

In our charts and tables, we use the following labels for these algorithms:

- *single-naive* – the original multi-tree algorithm [9] inside loops over resampling regions and matchers.
- *single-efficient* – the original multi-tree algorithm [9] inside a loop over matchers but using our improved resampling algorithm (Alg. 6).

- *multi-naive* – our improved multi-matcher algorithm inside a loop over resampling regions.
- *multi-efficient* – our multi-matcher algorithm used together with the efficient resampling method.

Algorithm	Time (s)	Speedup (brute-force)	Speedup (over [9])
multi-efficient	328.6	$\mathbf{1.4 \times 10^6}$	**105**
multi-naive	10229	4.6×10^3	3.36
single-efficient	1365.5	3.4×10^5	25.2
single-naive	34397	1.4×10^3	–

Table 1: Mock catalog experimental results for 2-point correlations, 10^6 points. 30 resampling regions, 20 matchers.

Algorithm	Time (s)	Speedup
multi-efficient	10057	$\mathbf{2.2 \times 10^5}$
multi-naive	96620	2.3×10^4
naive-efficient	21935	9.9×10^4

Table 2: Mock catalog experimental results for 3-point correlations, 10^6 points. 12 resampling regions, 18 matchers. Speedups are over the brute-force algorithm. The naive-naive algorithm was omitted because of time constraints.

Implementation details. We implemented our algorithms in C++. We used the *kd*-tree implementation in the MLPACK scalable machine learning library [4]. All experiments were performed in serial on a 6 core, AMD Phenom 3.3 GHz machine.

Datasets. We provide experimental results on uniform (Poisson data) in three dimensions and a mock galaxy catalog generated from N-body simulations. Both these sets are representative of the data used in actual n-point correlation estimation. Galaxy catalogs reproduce the correlation statistics of observed data. Computations involving Poisson distributed sets, such as those in the estimators in Section 2.1, represent a significant fraction of the total computational overhead.

Results. In Fig. 4, we show runtimes for all four algorithms on uniform data of different sizes. The numbers of matchers and resampling subsets are shown with each figure. Note that for all three figures, each of our new methods provides a large speedup, shown in Table 3. We see that in our experiments, each of our new algorithms provides roughly an order-of-magnitude speedup. These effects stack when used together, so we see speedups of well over 100.

The efficient resampling algorithm provides a greater advantage than the multi-matcher method in our experiments. However, this effect is less pronounced in our 4-point experiments (Fig. 4(c)). These experiments used more matchers than the others, suggesting that the savings due to the multi-matcher algorithm will increase with more matchers.

5. CONCLUSION & FUTURE WORK

We have described the first fast algorithms for the entire n-point correlation function estimation problem. Both of these new algorithms build on the current state-of-the-art tree-based approach by identifying computational bottlenecks in the entire npcf estimation problem. One of our methods computes the counts necessary for estimating the correlations at multiple scales in a single pass. The other efficiently incorporates the jackknife resampling step required to estimate the variance. We have shown that each of these methods can provide an order-of-magnitude speedup over the current best algorithms and that they can be used in conjunction for speedups up to 500 or greater.

We are currently developing two extensions of this approach. First, we are implementing another, more general multi-matcher algorithm which will be capable of handling overlapping bins. Second, we are working on a parallel implementation of our approach for the largest data sets.

n	Algorithm	Speedup (brute-force)	Speedup (over [9])
2	multi-efficient	$\mathbf{1.08 \times 10^6}$	**77.4**
	multi-naive	4.95×10^4	3.6
	single-efficient	3.7×10^5	26.6
3	multi-efficient	$\mathbf{5.9 \times 10^4}$	**228.6**
	multi-naive	2.03×10^3	7.87
	single-efficient	8.47×10^3	32.8
4	multi-efficient	$\mathbf{2.3 \times 10^6}$	**583**
	multi-naive	5.28×10^3	21.3
	single-efficient	1.2×10^4	2.72

Table 3: Speedups over the brute-force algorithm and current state-of-the-art. Timings are from the largest values of N in Fig. 4.

6. REFERENCES

[1] J. Bardeen et al. The statistics of peaks of gaussian random fields. *The Astrophysical Journal*, 304:15–61, 1986.

[2] L. Cooper et al. Two-point correlation as a feature for histology images. In *Computer Vision and Pattern Recognition Workshops*, volume 13, pages 79–86, 2010.

[3] N. Cressie. *Statistics for Spatial Data*. John Wiley & Sons, 1991.

[4] R. R. Curtin et al. MLPACK: A Scalable C++ Machine Learning Library. In *BigLearning: Algorithms, Systems, and Tools for Learning at Scale*, 2011.

[5] J. H. Friedman et al. An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Trans. Math. Softw.*, 3:209–226, 1977.

[6] J. Fry. The galaxy correlation hierarchy in perturbation theory. *The Astrophysical Journal*, 279:499–510, 1984.

[7] J. Gardner et al. A framework for analyzing massive astrophysical datasets on a distributed grid. In *Astronomical Society of the Pacific Conference Series*, volume 376, page 69, 2007.

[8] T. Giannantonio et al. High redshift detection of the integrated Sachs-Wolfe effect. *Physical Review D*, 74(6):063520, 2006.

[9] A. G. Gray and A. W. Moore. 'N-Body' problems in statistical learning. In *Neural Information Processing Systems*, volume 4, pages 521–527, 2000.

[10] A. Hamilton. Toward better ways to measure the galaxy correlation function. *The Astrophysical Journal*, 417:19, 1993.

[11] INBio. *The National Biodiversity Institute of Costa Rica*. `http://www.inbio.ac.cr/en`.

[12] S. Landy and A. Szalay. Bias and variance of angular correlation functions. *The Astrophysical Journal*, 412:64–71, 1993.

[13] D. Li et al. 3D reconstruction of carbon nanotube composite microstructure using correlation functions. *Journal of Computational and Theoretical Nanoscience*, 7:1462–1468, 2010.

[14] LSST. *The Large Synoptic Survey Telescope*. `www.lsst.org`.

[15] R. Lupton et al. The SDSS imaging pipelines. In *Astronomical Data Analysis Software and Systems X*, volume 238, page 269, 2001.

[16] C. McBride. *Our Non-Gaussian Universe: Higher Order Correlation Functions in the Sloan Digital Sky Survey*. PhD thesis, University of Pittsburgh, 2010.

[17] C. McBride et al. Three-point correlation functions of SDSS galaxies: constraining galaxy-mass bias. *The Astrophysical Journal*, 739:85, 2011.

[18] A. Moore et al. Fast algorithms and efficient statistics: N-point correlation functions. *Mining the Sky*, pages 71–82, 2001.

[19] K. Mosaliganti et al. Tensor classification of n-point correlation function features for histology tissue segmentation. *Medical Image Analysis*, 13:156–166, 2009.

[20] P. Peebles. *The Large-Scale Structure of the Universe*. Princeton University Press, 1980.

[21] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer, 1985.

[22] R. Ridgway et al. Image segmentation with tensor-based classification of n-point correlation functions. In *MICCAI Workshop on Medical Image Analysis with Applications in Biology*, 2006.

[23] B. Ripley. Locally finite random sets: Foundations for point process theory. *The Annals of Probability*, 4:983–994, 1976.

[24] B. Ripley. *Spatial statistics*. Wiley-Blackwell, 2004.

[25] SDSS. *The Sloan Digital Sky Survey*. `www.sdss.org`.

[26] C. Seife. Breakthrough of the year: Illuminating the dark universe. *Science*, 302(5653):2017–2172, December 19 2003.

[27] I. Szapudi. Introduction to higher order spatial statistics in cosmology. *Data Analysis in Cosmology*, pages 457–492, 2009.

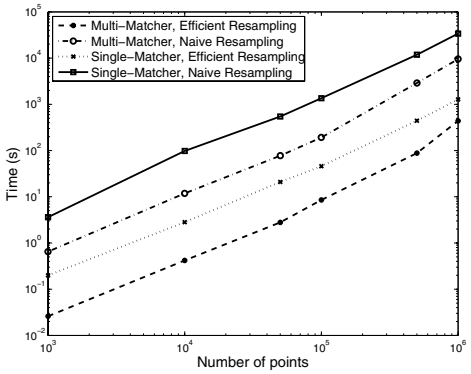
[28] I. Szapudi et al. Fast cosmic microwave background analyses via correlation functions. *The Astrophysical Journal Letters*, 548:L115, 2001.

[29] I. Szapudi and A. Szalay. A new class of estimators for the N-point correlations. *The Astrophysical Journal Letters*, 494:L41, 1998.

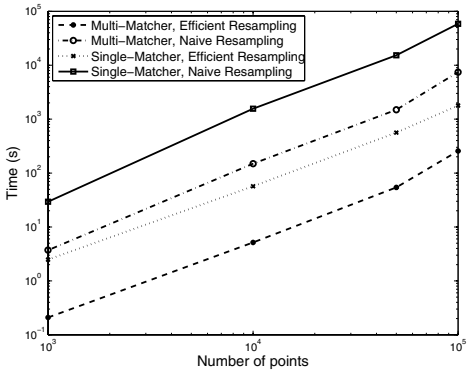
[30] S. Torquato. *Random Heterogeneous Materials*. Springer, 2002.

[31] S. Torquato and G. Stell. Microstructure of two-phase random media. I. The n-point probability functions. *The Journal of Chemical Physics*, 77:2071, 1982.

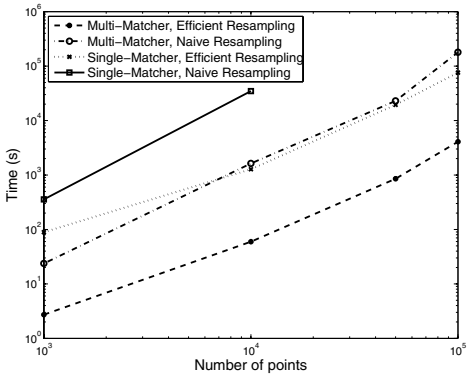
[32] S. White. The hierarchy of correlation functions and its relation to other measures of galaxy clustering. *Monthly Notices of the Royal Astronomical Society*, 186:145–154, 1979.



(a) Two-point correlation estimation on uniformly distributed data with 20 matchers and 30 resampling regions.



(b) Three-point correlation estimation on uniformly distributed data with 50 matchers and 30 resampling regions.



(c) Four-point correlation estimation on uniformly distributed data with 216 matchers and 30 resampling regions.

Figure 4: n-point correlation estimation computation times on log-log scale. All timings are in seconds.