MULTI TREE ALGORITHMS FOR COMPUTATIONAL STATISTICS AND PHYSICS

A Thesis
Presented to
The Academic Faculty

by

William B. March

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computational Science and Engineering

Georgia Institute of Technology
August 2013

MULTI TREE ALGORITHMS FOR COMPUTATIONAL STATISTICS AND PHYSICS

To Kristin, for being there the whole time

ACKNOWLEDGEMENTS

Many people played a crucial role in my graduate education, and it will be difficult to thank them all properly. The following is almost certainly not comprehensive.

First, I would like to thank my loving and patient wife Kristin. She has been with me for the whole process, from before grad school until now, and it would not have been possible without her care and support. My family have also been extremely supportive throughout. My parents Lee and Deb have been an excellent source of support and encouragement, having been through the process before. My brother Ben has also been an excellent source of encouragement.

I would also like to thank my fellow graduate students in the FASTlab and beyond. The two Ryans, Riegel and Curtin, have patiently endured my continuous small coding questions and debugging puzzles and answered them effectively. Parikshit Ram and Nishant Mehta endured (and even managed to answer) my frequent literature questions of the form "Didn't that guy from somewhere do something like that?" Jim Waters added a much-needed Simpson's quote at more than one opportune time. In all, without their help and support, the entire experience would have been far less enjoyable and rewarding.

The members of my committee were a valuable resource without whom this research would not be possible. I would like to thank Andy Connolly for hosting me at U.W. in the Summer of 2011 and furthering my work on n-point correlation functions. David Sherrill has encouraged my work in computational chemistry for many years and has been particularly supportive even of ideas that ultimately did not pay off. Rich Vuduc and George Biros provided valuable insights and criticisms at several points throughout the process.

Last, but certainly not least, I would like to thank my advisor Alex Gray. In addition to a great deal of valuable advice on both my research and the larger task of undertaking research at all, he encouraged me to be ambitious in my research topics and goals. Although this was certainly not the easy path, I think the end result is well worth the extra sweat and tears. Without him being willing to take a chance on me, none of this would have been possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The Fast Multipole Method of Greengard and Rokhlin does the seemingly impossible: it approximates the quadratic scaling N-body problem in linear time. The key is to avoid explicitly computing the interactions between all pairs of N points. Instead, by organizing the data in a space-partitioning tree, distant interactions are quickly and efficiently approximated. Similarly, dual-tree algorithms, which approximate or eliminate parts of a computation using distance bounds, are the fastest algorithms for several fundamental problems in statistics and machine learning – including all nearest neighbors, kernel density estimation, and Euclidean minimum spanning tree construction.

We show that this overarching principle – that by organizing points spatially, we can solve a seemingly quadratic problem in linear time – can be generalized to problems involving interactions between sets of three or more points and can provide orders-of-magnitude speedups and guarantee runtimes that are asymptotically better than existing algorithms. We describe a family of algorithms, *multi-tree* algorithms, which can be viewed as generalizations of dual-tree algorithms. We support this thesis by developing and implementing multi-tree algorithms for two fundamental scientific applications: n-point correlation function estimation and Hartree-Fock theory.

First, we demonstrate multi-tree algorithms for n-point correlation function estimation. The n-point correlation functions are a family of fundamental spatial statistics and are widely used for understanding large-scale astronomical surveys, characterizing the properties of new materials at the microscopic level, and for segmenting and processing images. We present three new algorithms which will reduce the dependence of the computation on the size of the data, increase the resolution in the result without additional time, and allow probabilistic estimates independent of the problem size through sampling. We provide both empirical evidence to support our claim of massive speedups and a theoretical analysis showing linear scaling in the fundamental computational task. We demonstrate the impact of a carefully optimized base case on this computation and describe our distributed, scalable, open-source implementation of our algorithms.

Second, we explore multi-tree algorithms as a framework for understanding the bottleneck computation in Hartree-Fock theory, a fundamental model in computational chemistry. We analyze existing fast algorithms for this problem, and show how they fit in our multi-tree framework. We also show new multi-tree methods, demonstrate that they are competitive with existing methods, and provide the first rigorous guarantees for the runtimes of all of these methods. Our algorithms will appear as part of the PSI4 computational chemistry library.

CHAPTER I

INTRODUCTION

1.1 Thesis Statement

Multi-tree algorithms, a class of higher-order generalizations of the Fast Multipole Method, can provide orders-of-magnitude speedups on fundamental problems in data analysis and scientific computing while providing guaranteed runtime bounds and a unifying framework for understanding these problems and their solutions.

We support this thesis through an exploratory analysis of multi-tree algorithms in action on problems of fundamental scientific importance. We develop, empirically evaluate, and theoretically bound and characterize multi-tree algorithms for n-point correlation function estimation in astronomy and cosmology and for Hartree-Fock theory in computational chemistry. By obtaining speedups and useful theoretical characterizations for real problems, we demonstrate that multi-tree algorithms may be useful for a broad range of tasks.

1.2 Overview of This Dissertation

The N-body problem is a fundamental computational task in the analysis and application of many scientific models. Given a set of N particles and some physical potential $\Phi(\cdot)$ which is a function of the distance between a pair of particles, we must compute the total potential at each particle due to all of the others: $\Phi_i = \sum_{j=1}^{N} \Phi(\|r_i - r_j\|)$. With scientific simulations and data sets already massive and growing, the $O(N^2)$ scaling that this task seems to require makes it prohibitively expensive.

However, the Fast Multipole Method of Greengard and Rokhlin [76] can approximate the solution in $O(N)$ time. At a high level, the method indexes the points using a space partitioning tree, such as an octree. Using this indexing, we can then approximate the interactions between distant nodes in the tree, thus eliminating many pairwise potential evaluations. By carrying these approximations up and down the tree, we arrive at an approximate solution with bounded error in linear time.

The all nearest neighbors problem is a fundamental task in machine learning and data mining, where it serves as the basis for a variety of classification, regression and clustering algorithms. Given a set of data points in $\mathbb{R}^d$, we must find the nearest neighbor of each point under a given metric. Once again, this task seemingly requires $O(N^2)$ work – we compute all pairwise distances, then scan for the smallest for each point.

Once again, clever tree-based algorithms can overcome this obstacle. We again index the data with a space-partitioning tree. We consider a pair of tree nodes at a time, one for *queries* and one for *references* – the points for which we are searching for a nearest neighbor and those which might be a nearest neighbor, respectively. For each query, we maintain an upper bound on the distance to the true nearest neighbor obtained from the closest reference point seen so far. We store the maximum upper bound over all its points in each node. When the lower bound distance between a query and reference node is greater than this maximum upper bound for the query, we can immediately dismiss all the references as possible nearest neighbors for the queries. This algorithm can be shown to eliminate most of the pairwise distance computations, allowing us to find the nearest neighbor of every point in $O(N)$ time [141].

This doubly-recursive, or *dual-tree*, algorithm has been the basis of efficient algorithmic solutions to many other prolbems in machine learning and computational geometry, including Euclidean minimum spanning tree construction [112], kernel summations for kernel density estimation [72, 108], naive Bayes classification [69], and mean shift clustering [172].

Both the FMM and the all nearest neighbors algorithms have the dual-tree structure in common – we obtain speedups by considering pairs of nodes in a space-partitioning tree and using the bounds we obtain from them to compute approximations or avoid parts of the computation entirely.

There are other fundamental problems in computational science and data mining and machine learning which scale even worse – they consist of all interactions between sets of three, four, or even more points. Fundamental problems in astronomy, materials science, medical image processing, drug design, and computational chemistry fit this general framework. This dissertation applies the algorithmic principles and methods learned from the dual-tree setting to these higher-order problems.

We begin by detailing the algorithmic principles and techniques underlying dual-tree algorithms. We define dual- and multi-tree algorithms. We pursue an extended example of these techniques through their application to Euclidean minimum spanning tree construction. We then turn to applications of multi-tree algorithms in computational science.

We focus on two problems in particular. First, we discuss the task of estimating the n-point correlation functions – a set of fundamental spatial statistics capable of fully characterizing any point process. In addition to their generality, these functions are fundamental in astronomy and cosmology and materials science. The estimators for these functions consist of counting all n-tuples of points. If performed directly, this requires $O(N^n)$ time. However, like most scientific data sets, sky surveys currently consist of billions of points and are rapidly growing. Thus, efficient algorithms are essential to keep pace.

We define the computational tasks required, then present several new multi-tree algorithms for the n-point correlation function estimation problem. In addition to algorithms for the basic computational task, we show how to efficiently compute the npcf at multiple scales in a single computational pass and how to incorporate resampling-based variance estimation directly into our algorithm. We present a newly optimized base case implementation and an efficient distributed implementation of our algorithms, both of which create substantial speedups over previous methods. We also show a theoretical analysis of our algorithms, proving that we have solved a task seemingly requiring $O(N^n)$ work in $O(N)$ time with no approximations.

Our second problem is the construction of the Fock matrix in Hartree-Fock theory. Hartree-Fock theory is a fundamental approximation used to compute wavefunctions of molecules in computational chemistry. Furthermore, this method serves as the first step in more accurate correlation methods. Since understanding molecules at the quantum level is crucial for applications in drug design, materials design, and other applications, an efficient and accurate method for this problem is important. The bottleneck computation in Hartree-Fock theory is the construction of the Fock matrix, an N by N matrix, each entry of which is a double-summation over the entire data set. Since N depends on both the size of the molecule being considered (number of atoms) and the accuracy needed, this $O(N^4)$ scaling is prohibitively expensive.

We describe the Fock matrix construction task in detail, paying particular attention to the key two-electron integrals. We show that existing fast algorithms for Fock matrix construction fit within our multi-tree framework. We then describe two new multi-tree algorithms. We provide the first detailed theoretical analysis of existing methods, and show their true dependence on both the size of the input and properties of the computation being performed. We present a preliminary empirical evaluation of our algorithms, and discuss their current limitations.

1.3 Contributions of this Dissertation

In support of this thesis, we make several concrete contributions. Here, we present a list of these, which are described in detail in the following chapters.

1.3.1 Algorithms

- We develop a dual-tree algorithm for Euclidean minimum spanning tree construction. This algorithm is the fastest method available for the EMST problem for more than two-dimensional data, providing an order-of-magnitude speedup over the previous state-of-the-art.

- We develop a general multi-tree algorithm for n-point correlation function estimation. We also present algorithms for performing this computation at multiple scales simultaneously and for efficiently obtaining variance estimates through resampling. Each of these algorithms can provide up to an order-of-magnitude speedup over previous methods. These improvements stack for truly massive performance improvements.

- We present two new multi-tree algorithms for the entire Fock matrix construction problem. These algorithms are the first linear scaling methods to compute the entire Fock matrix, rather than just one term in it.

1.3.2 Theory

- We define dual- and multi-tree algorithms. We also discuss adaptive algorithm analysis – a form of algorithm analysis which takes properties of the data into account to obtain tighter and more informative runtime bounds.

- We prove that our dual-tree EMST algorithm runs in nearly optimal $O(N \log N \, \alpha(N))$ time, where $\alpha(N)$ is a negligibly small factor.

- We show the first proof that the n-point correlation functions can be estimated in $O(N)$ time for an important subset of inputs, which is optimal for exact algorithms. We prove that our new algorithms achieve this bound.

- We unify existing fast methods for Fock matrix construction under our multi-tree framework, which were previously considered to fundamentally rely on the details of the computational task.
- We present the first formal runtime proofs for Fock matrix construction algorithms. These methods are frequently claimed to run in linear time. We present the first formal analysis of this claim, and show that it holds under some significant conditions. We also show that our new algorithm satisfies this bound, as well.

1.3.3 Implementations

- We contributed our EMST algorithm to the MLPACK open-source machine learning library [39] using any general space-partitioning tree.
- We developed an open-source n-point correlation estimation library, NPOINT. This algorithm includes all our efficient algorithms, an optimized base case computation responsible for another order-of-magnitude speedup, and a distributed, MPI based version of our algorithm.
- We have contributed efficient implementations of the currently optimal CFMM and LinK methods for Fock matrix construction to the Psi4 open-source computational chemistry package [169], thus extending the package's capabilities to much larger molecules than were previously possible. We have provided preliminary implementations of our algorithms as well.

1.4 Publications

Some of the work presented in this thesis has been previously published in the peer-reviewed literature.

- Optimizing the Computation of N-Point Correlations on Large-Scale Astronomical Data, *Supercomputing*, 2012 [114]. In this paper, we developed our optimized n-point correlation estimation base case and demonstrated that it is capable of up to an order-of-magnitude speedup over the previous best implementation.

- Fast Algorithms for Comprehensive N-point Correlation Estimates. In *SIGKDD*, 2012 [113]. In this paper, we develop our efficient multi-scale npcf estimation algorithm and our jackknife resampling algorithm. We show that each of these algorithms are capable of up to an order-of-magnitude speedup and that these effects can be stacked.

- Fast Euclidean Minimum Spanning Tree: Algorithm, Analysis, Applications. In *SIGKDD*, 2010 [112]. In this paper, we develop our fast Euclidean MST algorithm. We demonstrate the effectiveness of this algorithm on both *kd*-trees and cover trees, and show that it provides speedups over the previous state-of-the-art on both real and synthetic data from 3 to thousands of dimensions. We also present an adaptive analysis of the runtime of our algorithm and show the that our algorithm's runtime is within a negligible factor of optimal.

- Multitree Algorithms for Large-Scale Astrostatistics. In Advances in Machine Learning and Data Mining for Astronomy (Edited Volume), 2012 [115]. In this review article, we describe the applications of multi-tree algorithms in astronomy, including n-point correlation function estimation, naive Bayes classification, kernel density estimation, and Euclidean minimum spanning trees.

- Linear Time Algorithms for Pairwise Statistical Problems. In *NIPS*, 2009 [141]. We show adaptive algorithm analyses of several dual-tree algorithms, including those for all nearest neighbor and kernel density estimation.

- MLPACK: A Scalable C++ Machine Learning Library. In *JMLR*, 2013 [39]. We describe the open-source machine learning library MLPACK and demonstrate its efficient implementation of many fundamental machine learning algorithms, including dual-tree algorithms.

- Tree-Independent Dual-Tree Algorithms. In *ICML*, 2013 [40]. We examine one advantage of the overarching dual-tree framework – the ability to easily use different data structures within one algorithmic framework.

This dissertation is the first time all of our n-point correlation estimation algorithms have been described together, with their total speedups exhibited. Our theoretical analyses these algorithms have also not been previously published. Our work on Fock matrix construction is also previously unpublished.

1.5 Outline of the Dissertation

The rest of this dissertation is organized into four chapters. In Chapter 2, we define dual- and multi-tree algorithms and give an overview of the algorithmic techniques responsible for their speedups for many different problems. We also explore the application of alternative forms of algorithm analysis to these techniques and show how they can yield much tighter runtime bounds than traditional worst-case analysis. We also present an extended example of Euclidean minimum spanning tree computation.

In Chapter 3, we turn to our first example of multi-tree algorithms in action – n-point correlation function estimation. We define the n-point correlations and discuss the fundamental role they play in a variety of scientific and data mining applications, including astronomy and cosmology, materials science, and image processing. We then examine the fundamental computational task underlying estimation of the npcf and show that it is well suited to multi-tree algorithms.

We then show several extensions of the basic multi-tree algorithm which yield additional speedups. We develop an algorithm capable of estimating the npcf at many scales simultaneously, using one pass through a multi-tree algorithm. We also demonstrate a dynamic programming algorithm for efficiently computing jackknife variance estimates of the npcf using our multi-tree algorithm. We also show optimized base cases for 3-point correlation function estimation and demonstrate their impact on the overall algorithm. We develop a distributed implementation of all of these methods and describe our open-source library making these available.

In Chapter 4, we explore another application of multi-tree algorithms – Hartree-Fock theory. HF theory is a fundamental tool in computational chemistry, both for obtaining approximate wavefunctions for molecules and as the first step in more accurate calculations. We briefly sketch HF theory and highlight the rate-limiting step – the construction of the Fock matrix. We show that this computation is a double-summation over N elements which must be performed for all pairs of N queries – a perfect candidate for a multi-tree algorithm.. We examine existing fast algorithms for different parts of this problem, in particular two of the most widely used – the Continuous Fast Multipole Method [176] and Linear K (LinK) algorithm [127]. We also describe several multi-tree algorithms for this problem. We show that all of these algorithms, including those in the literature, can be viewed as multi-tree algorithms, thus providing a unifying framework for understanding the properties and opportunities for speedups in these algorithms.

We demonstrate that our multi-tree algorithm is competitive with the best algorithms for exchange matrix construction. We also show that using the unified multi-tree framework, we can prove the first rigorous runtime bounds for all of these methods – the CFMM, LinK, and our multi-tree algorithms. We conclude by observing that many other problems in computational chemistry may be amenable to solution via multi-tree algorithms.

Finally, we conclude in Chapter 5. We discuss ongoing work and further opportunities suggested by this work.

CHAPTER II

MULTI-TREE ALGORITHMS

In this chapter, we define and examine dual-tree and multi-tree algorithms. We discuss the origins of these methods and show their applicability to a wide class of problems involving a computation over all pairs of points. We then discuss examples of higher-order computations – ones that involve all triples (or more generally, n-tuples) of points, rather than just pairs. We then show an extended example of dual-tree algorithms in action on the Euclidean minimum spanning tree problem. We particularly emphasize the desirable properties of dual-tree algorithms that we hope to achieve with multi-tree algorithms. These include efficient empirical runtimes and their amenability to adaptive algorithm analysis for theoretical performance guarantees.

We begin with a general discussion of space-partitioning trees – the key data structure used in dual- and multi-tree algorithms. We then introduce dual-tree algorithms and present two important problems that can be efficiently solved with them – all nearest neighbors and the N-body problem. We sketch the generalization of dual-tree algorithms to multi-tree algorithms. We conclude with our dual-tree EMST algorithm.

2.1 Space Partitioning Trees

In this dissertation, we make frequent use of a class of data structures for indexing points in $\mathbb{R}^d$ by position. We refer to these collectively as space-partitioning trees. Here, we give some overall definitions and notations that will be used subsequently. We also define several commonly used types of space partitioning trees.

We begin with a general definition of space partitioning trees [40].

Definition 2.1.1. *A space partitioning tree on a dataset $D \subset \mathbb{R}^d$ is an undirected, connected, acyclic, rooted simple graph with the following properties:*

- *Each node (or vertex), contains a number of points (possibly zero) and is connected to at most one parent node a number of child nodes (possibly zero).*
- *There is one node in every space partitioning tree with no parent; this is the root node of the tree.*
- *Each point in D is contained in at least one node of the tree.*
- *Each node N of the tree represents a convex subset of $\mathbb{R}^d$ which contains each of the points in the node as well as the convex subsets represented by each child of the node.*

Throughout, we refer to nodes of a tree with capital letters and points within them as lower case letters. We can specify a space partitioning tree by giving the convex subset corresponding to a node and the method of splitting a node into subsets. We frequently represent both a node N and the set of points contained in N and its descendants interchangeably when no confusion can result.

As we will see below, the key to using space partitioning trees for efficient solutions to geometric problems is in computing upper and lower bounds on the distances between a point and any point in a tree node. For a point q and tree node R, we will require the minimum and maximum possible distances between q and any point r in R.

$$\begin{aligned} d^{\min}(q, R) &= \min\nolimits_{r \in R} d(q, r) \\ d^{\max}(q, R) &= \max\nolimits_{r \in R} d(q, r) \end{aligned} \tag{2.1.1}$$

As mentioned in the definition, each node of a tree includes a convex bounding set containing all of its points. If we can efficiently compute the minimum and maximum distance between the point q and the interior of this bounding set, we immediately obtain these bounds.

Similarly, given two tree nodes Q and R, we can obtain pairwise distance bounds.

$$\begin{aligned} d^{\min}(Q,R) &= \min_{q\in Q}\min_{r\in R} d(q,r) \\ d^{\max}(Q,R) &= \max_{q\in Q}\max_{r\in R} d(q,r) \end{aligned} \tag{2.1.2}$$

These are obtained by computing the minimum or maximum distance from the interior of the bounding set of Q to the interior of the bounding set of R.

The particular space partitioning trees used in this dissertation use axis-aligned rectangles or balls as their convex bounding sets. In the case of balls for nodes N_1, N_2 of radii r_1, r_2 centered at points $c_1, c_2 \in \mathbb{R}^d$, these bounds are easy to compute:

$$\begin{aligned} d^{\min}(N_1,N_2) &= \max\{d(c_1,c_2) - r_1 - r_2, 0\} \\ d^{\max}(N_1,N_2) &= d(c_1,c_2) + r_1 + r_2 \end{aligned} \tag{2.1.3}$$

For axis-aligned rectangles, let the low and high endpoints of the rectangle be stored in arrays l_1, l_2 and h_1, h_2. Then, we can compute the distances as follows:

$$\begin{aligned} d^{\min}(N_1,N_2) &= \sum_{i=0}^{d} \min\{fillmein\} \\ d^{\max}(N_1,N_2) &= \sum_{i=0}^{d} \max\{|h_1[i] - l_2[i]|, |h_2[i] - l_1[i]|\} \end{aligned} \tag{2.1.4}$$

We now describe the three main types of trees used in the remainder of this work: octrees, kd-trees, and cover trees.

2.1.1 Quad- and Oct-trees

The quadtree (in two dimensions) and octree (in three) are two widely used space partitioning trees. The convex subset consists of an axis-aligned cube. The root is the smallest such cube which contains all of the points. We can split a node by partitioning it into four sub-cubes with edge length half that of the original (or into 2^d sub-cubes in d dimensions). We then assign points to children based on which sub-cube they lie in.

Throughout, we use the term octree without reference to the dimensionality. This is because we are typically working in three dimensions. When we are not, an octree is to be understood to refer to its d-dimensional variant.

The octree is widely used for several reasons. It is both simple to visualize and understand. It can also be implemented straightforwardly, and more sophisticated implementations avoid the use of pointers entirely for greater efficiency. The octree's simple bounding set and regular structures also make octree-based algorithms easy to analyze. The tree requires only linear storage space, and can be constructed in $O(N \log N)$ time for N points.

Although the branching factor is fixed, it is exponential in the dimensionality of the data. Furthermore, an adversarially chosen data set may cause the tree to have up to linear depth. These restrictions will require some careful analysis when used in theory, which is discussed in later chapters.

2.1.2 *kd*-trees

Quadtrees (and their higher-dimensional versions) are well suited to low-dimensional problems. However, as the dimensionality increases, they quickly become intractable due to their exponential branching factor. Additionally, many data sets have high intrinsic dimensionality, but actually lie along some manifold of lower dimension.

Friedman, Bentley, and Finkel introduced a data structure capable of using this observation in 1976 [59]. They introduced the *kd-tree*. Here, the convex bounds for a node are axis-aligned rectangular boxes which are no longer constrained to be cubes. We split a node by partitioning its bounding box at the midpoint (or median) of its longest dimension. We create a left and right child node using the points on either side of this boundary. We create new bounding boxes for each child consisting of the smallest axis aligned rectangle which contains all of the points.

These trees enjoy a considerable advantage in higher extrinsic dimensions, since they have a fixed and manageable branching factor. However, the highly variable nature of the splits and the possibility of very uneven distributions in sizes of bounding rectangles can make these trees difficult to use in theoretical analyses.

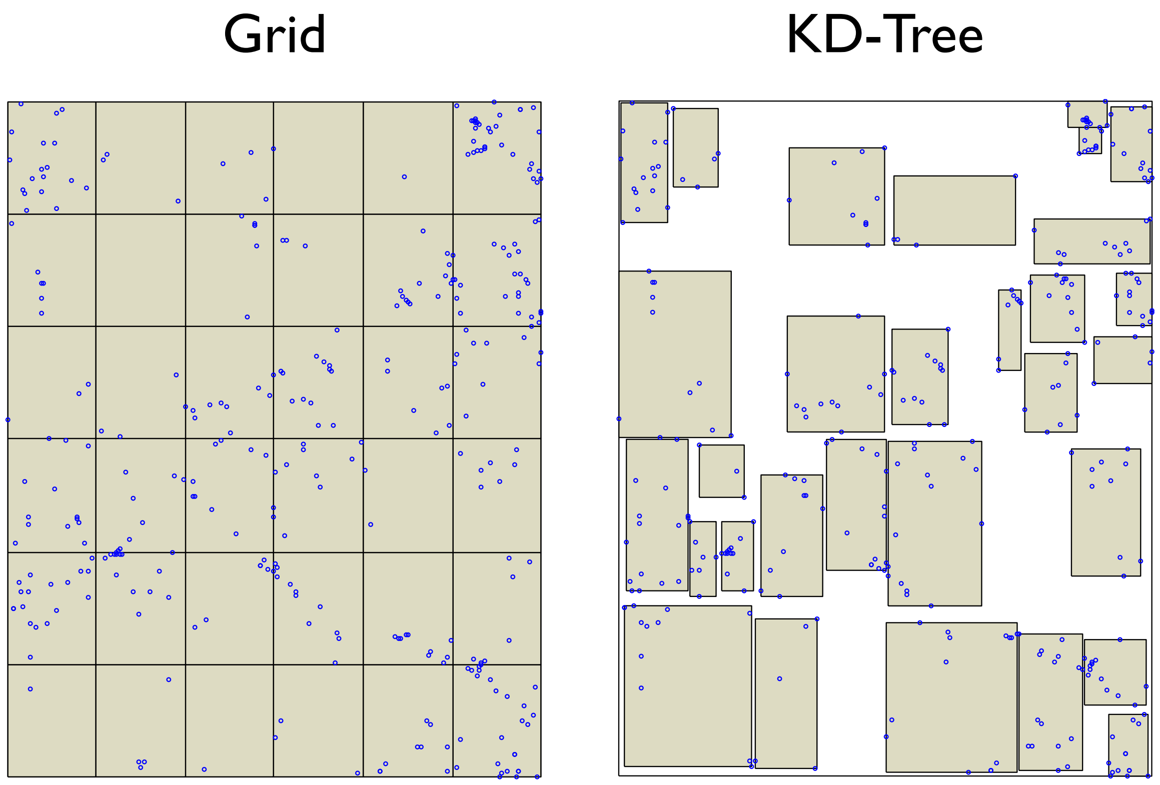


Figure 1: A visualization of boxes in a uniform grid (quadtree) compared to a *kd*-tree.

kd-trees can be constructed in $O(N \log N)$ time and require linear storage. For higher-dimensional problems, they are almost always empirically superior to octrees. However, the widely varying box sizes and shapes can make them more difficult to use in theoretical analyses.

2.1.3 Cover Trees

The cover tree [19] data structure attempts to combine the adaptability of *kd*-trees with the theoretical applicability of simpler trees. Each node of a cover tree consists of a single point and a bounding radius around that point. We split a parent node by first creating a self-child consisting of the same point and a bounding ball of half the original radius [1]. The self-child contains any points in the parent that are contained in

[1]Any constant factor may be used to decrease the radius of child nodes. In practice, a factor of 1.3 is often effective. Here, as in the references, we restrict the discussion to factors of two for simplicity.

this half-radius ball. We then account for the other points by choosing one at random and placing a new half-radius ball with it at the center. All previously unaccounted for points are placed in this node. We continue this process until all points are placed in a child node.

This tree is similar to the quadtree in that children have a regular structure which makes the data structure amenable to theoretical analyses. However, it differs from both examples discussed above in that it has a variable branching factor.

In addition to being empirically useful, one of the key advantages to the cover tree is its applicability to theoretical analyses. This analysis is typically carried out through a measure of intrinsic dimensionality – the *expansion constant*. It can be shown that the branching factor depends only on this measure, not on the size of the data or the extrinsic dimensionality. Furthermore, for fixed expansion constant, the depth of the tree is logarithmic in the size of the data and the size of the tree is linear.

A cover tree consists of a set of nested sets C_i, each at a scale i. A node in the cover tree consists of a single point and links to the node's children. The root is a single point at level ∞. As we descend the tree, the scale decreases, until $C_{-\infty}$ contains the entire set of points. For convenience, we index nodes in the cover tree with the node's point and use p_i to denote the node indexed by point p at level i of the tree. The cover tree maintains three invariants for each of its levels:

1. Nesting: $C_i \subseteq C_{i-1}$

2. Covering: For every $p \in C_{i-1}$, there exists a $q \in C_i$ such that $d(p, q) \leq 2^i$ and exactly one such q is a parent of p. Note that this implies that if p' is any descendant of a point $p \in C_i$, then $d(p, p') \leq 2^{i+1}$.

3. Separation: For all $p, q \in C_i$, $d(p, q) > 2^i$.

Intuitively, the cover tree consists of an infinite number of levels, with the nodes in each higher level providing a "covering" for the nodes below. At very high levels, the root node covers all the points. As we descend levels, each node shrinks, and more nodes are required to cover the set. As we near the lowest levels, each point is a node. We refer to this "infinite" cover tree as the *implicit representation* and make use of it in algorithm descriptions and proofs.

The *explicit representation* allows us to use the cover tree in practice. In the implicit representation, there are many levels where a node has only itself as a child. To create the explicit representation, we combine all such nodes. Therefore, a node is a single point, and contains pointers to all its children. The explicit representation has $O(N)$ nodes [20].

2.2 Dual-Tree Algorithms

We now turn to the main topic of this chapter: dual-tree algorithms. Dual-tree algorithms are a computational framework that has been applied to many problems in computational statistics, physics, and machine learning. These algorithms are the overall fastest known methods for many problems, including all nearest neighbors [74], kernel density estimation [73], mean shift [172], kernel discriminant analysis [144], and general kernel summations [105, 104].

Dual-tree algorithms are an efficient class of divide-and-conquer algorithms introduced in [74]. We introduce the basic concepts of these algorithms through a concrete example: the **All Nearest Neighbors** (AllNN) problem.

We are given two sets of points: a set of *queries* $\mathcal{Q}$ and a set of *references* $\mathcal{R}$, both embedded in a metric space with distance function $d(\cdot, \cdot)$. Our task is to compute the nearest neighbor in $\mathcal{R}$ of each point in $\mathcal{Q}$:

Definition 2.2.1. ***All Nearest Neighbors Problem.***

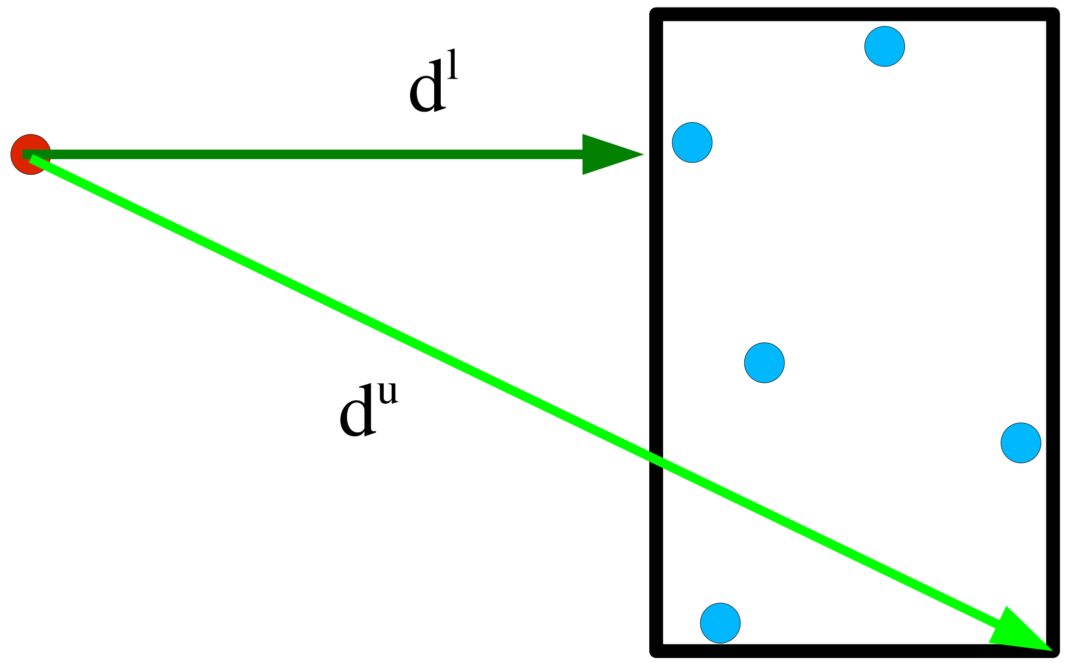
$$\forall q_i \in \mathcal{Q}, \text{ find } r^*(q_i) = \arg\min_{r_j \in \mathcal{R}} d(q_i, r_j)$$

Throughout our discussion, we will assume that $\mathcal{Q}$ and $\mathcal{R}$ are finite subsets of $\mathbb{R}^d$ and that the distance function is the usual Euclidean distance. However, this assumption is only made to simplify the descriptions of the algorithms, which can generalize to any metric space.

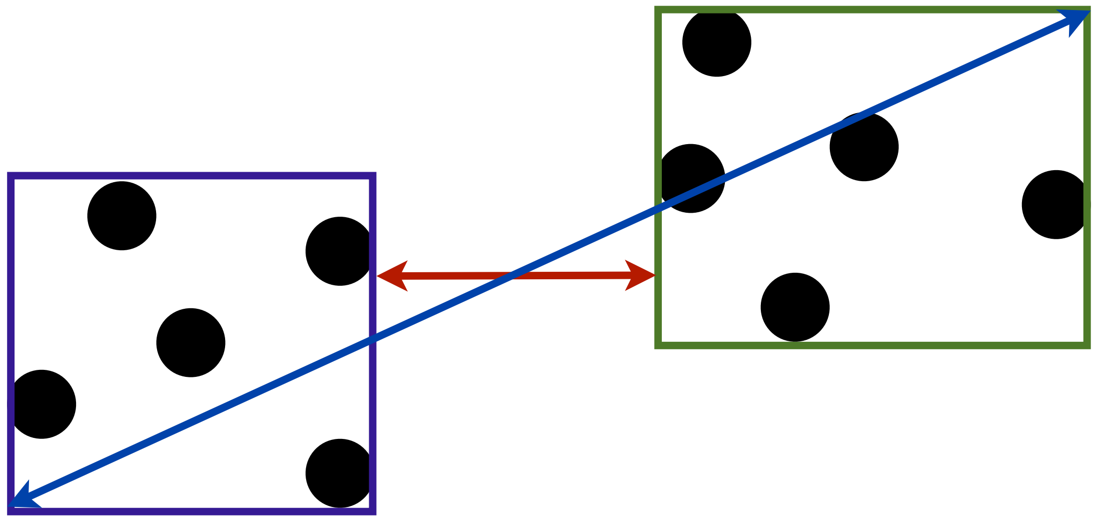
The simplest algorithmic solution for the AllNN problem is simply to examine every query in turn, compute the distance between it and every reference, and store the closest reference. We refer to this as the "naive" or "exhaustive" algorithm. Since this method must compute all of the $\binom{N}{2}$ pairwise distances, it requires $O(N^2)$ time.

Instead, consider a point q_i and a group of points R in Fig. 2(a). Let the points in R be contained in a convex set, such as that provided by a space-partitioning tree. In the figure, we have shown a bounding rectangle from a *kd*-tree. As noted previously, we can quickly compute distance bounds $d^{\min}(q, R)$ and $d^{\max}(q, R)$. Furthermore, consider the case where we have already computed the distance between q_i and at least one point in the reference set. The smallest distance we have seen so far provides an upper bound on the distance between q_i and the true nearest neighbor. Call this candidate nearest neighbor distance $\hat{d}(q_i)$. If $\hat{d}(q_i) < d^l(q_i, R)$, then we know that none of the points in R can be the true nearest neighbor of q_i. We can therefore avoid computing all of the pairwise distances $d(q_i, R)$.

In order to use this observation in a fast algorithm, we must consider possible query-reference pairs in a different order than in the naive method. Rather than iterate through the pairs one-by-one, we will employ a divide-and-conquer approach. Using a space-partitioning tree data structure built on the set $\mathcal{R}$, we can recursively split the computation into smaller, easier to handle subproblems. Using the bounding information in the tree, we can make use of the observation above. When the distance bounds indicate that no point in the reference subset being considered can be the nearest neighbor of the query, we can *prune* the subcomputation and avoid considering some of the pairs. If we are able to prune enough, we can improve substantially on the naive algorithm.



(a) Comparing a point with a node.



(b) Comparing two nodes.

Figure 2: An illustration of pruning in the computation of all nearest neighbors.

We can now describe an improved divide-and-conquer algorithm using a space-partitioning tree built on the set of references. We use a *kd*-tree for concreteness. We consider each of the query points in turn along with the root node of our *kd*-tree. Assume we are considering a point q_i and a node R, as in Fig. 2(a). If the node is a leaf, we compute the distance between q_i and every point in R, and if one of these is the smallest distance seen so far, we update the candidate nearest neighbor. Otherwise, there are two possibilities. If the lower bound distance $d^l(q_i, R)$ is less than the candidate distance $\hat{d}(q_i)$, it is possible for R to contain the true nearest neighbor of q_i. Therefore, we recursively consider both children of R. If $d^l(q_i, R) > \hat{d}(q_i)$, then no point in R is the nearest neighbor of q_i. In this case, we can *prune* the rest of the tree under node R.

For a given query point q_i, this *single-tree algorithm* will find the nearest neighbor using a cover tree in $O(\log N)$ time [19]. Since we must compute this algorithm for every query, the total running time is $O(N \log N)$.

We can make even better use of the observation in Fig. 2. Instead of considering the distance between a point and a bounding box, we can compute bounds on the distances between points in two bounding boxes – Fig. 2(b). We consider a query node Q and a reference node R, along with the largest candidate neighbor distance for the points in Q, $\hat{d}(Q) = \max_{q_i \in Q} \hat{d}(q_i)$. Now, if $d^l(Q, R)$ is greater than $\hat{d}(Q)$, no point in R can be the nearest neighbor of any point in Q, and we can prune *both* Q and R. When we pruned in the single tree algorithm, if the number of points in node R is $|R|$, we avoided computing $|R|$ distances. If we can make use of this new observation, we will save $|Q| \cdot |R|$ computations with each prune.

We use this observation to improve on the single-tree algorithm. We construct two trees, one on references and one on queries (hence *dual-tree* algorithm). We consider pairs of nodes at a time and use the distance bounds shown in Fig. 2(b). If the bounds show that the nodes are too distant, then we can prune. Otherwise, we split one (or

Algorithm 2.2.1 AllNN (Tree Node Q, Tree Node R)

```
    if $Q$ and $R$ are leaves then
      for all $q_i \in Q$ do
3:      for all $r_j \in R$ do
          if $d(q_i, r_j) < \hat{d}(q_i)$ then
            $\hat{d}(q_i) = d(q_i, r_j); n(q_i) = r_j$
6:        end if
        end for
      end for
9:    $\hat{d}(Q) = \max_{q_i \in Q} \hat{d}(q_i)$
    else if $d^{\min}(Q, R) > \hat{d}(Q)$ then
      // prune
12: else
      for all $Q' \in \mathcal{C}(Q)$ do
        for all $R' \in \mathcal{C}(R)$ do
15:       AllNN(Q', R')
        end for
      end for
18:   $\hat{d}(Q) = \max_{Q' \in \mathcal{C}(Q)} \hat{d}(Q')$
    end if
```

both) nodes, and recursively consider the two (or four) resulting pairs. We start by considering the root node twice.

2.2.1 The N-Body Problem

We have introduced the fundamental concepts of dual-tree algorithms through one of the first problems they were used to solve efficiently. However, a class of very similar algorithms were developed in parallel to solve problems in computational physics.

The N-body problem is a fundamental computational task in classical physics, with analogs in many other fields. We consider a model in which point masses interact through some potential Φ, which is a function of their positions. Commonly, Φ depends only on the distance between them[2]. We again use the query-reference terminology used above. We can also divide the problem into a single query and all query version.

[2]In the statistical and machine learning literature, this problem is directly analogous to the all kernel-summation problem.

Problem 2.2.2. N***-body Problem.*** *Given sets of reference and query points R and Q, compute the potential at each point q. More succinctly, compute:*

$$\forall q \in Q, \Phi_q = \sum_{r \in R} \Phi(q, r) \tag{2.2.1}$$

We showed that space partitioning trees can be used to accelerate the solution to the nearest neighbor problem by pruning reference nodes that cannot contain the true solution. However, it is not immediately clear how to do this for the N-body problem. If the kernel function Φ has infinite support (as is common in physics), then no reference points can be discarded. However, in general, an approximate solution is sufficient. We now turn to the construction of single-tree algorithms for efficiently approximating Problem 2.2.2.

The key here is another simple observation. If a set of reference points R is very compact and distant from the query point q, then the values $\Phi(q, r)$ are almost the same for each $r \in R$. The Barnes-Hut algorithm [13] takes advantage of this observation in a single tree algorithm. The algorithm was originally presented on a quadtree, but can be generalized.

In each node R of the tree, we maintain the total mass m_R and center of mass c_R. This can be easily computed as a preprocessing step using a bottom-up traversal. We specify an approximation parameter θ. We can then prune a node R if

$$s/d(q, c_R) < \theta \tag{2.2.2}$$

where s is the side-length of the bounding cube of R.

When we can prune, we add the contribution of a single virtual particle of mass m_R at c_R to the total potential.

This algorithm is again efficient in both theory and practice, since it is both widely used and generally claimed to run in $O(N \log N)$ time. The version usually

Algorithm 2.2.2 Barnes-Hut (query point q, Quadtree node R, approximation parameter θ)

```
    // Maintain $n(q)$ as the closest point found so far to $q$ and $d(q) = d(q, n(q))$.
    **if** $R$ is a leaf **then**
3:    **for all** $r_j \in R$ **do**
        **if** $q \neq r_j$) **then**
          $\Phi_q += \Phi(q, r_j)$
6:      **end if**
      **end for**
    **else if** $s/d(q, c_R) < \theta$ **then**
9:    $\Phi_q += \Phi(q, c_j)$
    **else**
      **for all** $R' in \mathcal{C}(R)$ **do**
12:     **Barnes-Hut**$(q, R')$
      **end for**
    **end if**
```

presented (and the one shown here) is only for quad trees. We can generalize the algorithm by using the distance bounds to compute general upper and lower bounds on the kernel. We then specify an approximation tolerance and prune if we can show that the maximum possible error (the distance between our approximation and the bounds) is smaller than the tolerance. This is the basis for efficient, dual-tree kernel summation algorithms [75].

The dual-tree idea was first presented in the context of the N-body problem by Appel [8]. The Barnes-Hut algorithm uses the observation that a compact group of reference points can be treated as a single pseudo-point with small error. Appel's algorithm applies this observation to queries as well. For a compact set of query points, the potential due to a single reference point at each query is nearly the same.

Rather than fixing a query point and traversing a tree built on reference points, we build trees on each set and traverse them simultaneously [3]. This algorithm is generally considered to run in $O(N)$ time for common input distributions [53]. In practice, it can offer speedups over Barnes-Hut, but these are often difficult to achieve.

[3]Note that the query and reference points are often the same set. Here, we discuss the more general bichromatic case.

This algorithm can be further improved through more sophisticated pruning techniques and approximations. The Fast Multipole Method [76] uses multipole expansions to more accurately treat the interactions between distant query and reference nodes. Although the FMM is not generally implemented as a recursive algorithm, it fundamentally consists of interactions between pairs of nodes which are either pruned or expanded, depending on some pruning criterion.

The FMM uses the same observation as Appel's algorithm – the potential due to one group of points on another, distant group can be approximated with a simpler calculation. Rather than simply grouping all of the source points into a single pseudo-point, the FMM more fully captures their distribution through a multipole expansion.

We can compute, for each node, a multipole expansion approximating the potential due to points in the node at a distant point. Through translation operators, we can transform the far-field expansion at one node – i.e. the effect its points have on distant points – into a near-field expansion at a target node – the effect distant points have on this node.

Note that unlike the dual-tree algorithms described above, the FMM is not recursive. It could easily be implemented as a recursive algorithm through a top-down dual-tree traversal. This is generally not done for performance efficiency. However, the FMM does use the same fundamental principle as the other dual-tree algorithms discussed: the problem is divided into subproblems, some of which are efficiently approximated to reduce the overall runtime.

2.3 Extending Dual-Tree Algorithms to Multi-Tree Algorithms

We have described two fundamental problems and given dual-tree algorithms to solve each of them. We now provide a working definition for dual- and multi-tree algorithms. Although more rigorous work is being done on this topic [40], we stick to a

Algorithm 2.2.3 FMM(Point set S)

```
    Construct an octree $T$ on $S$
    // Upward pass
    for all leaves $L$ in $T$ do
      Compute a multipole expansion for points in $L$
 5: end for
    Pass expansions up the tree using translation operators
    // Downward pass
    for all nodes in $T$, starting with the root and proceeding level-by-level do
      Find all nodes $T'$ on the same level as $T$ whose interactions have not been
      approximated
10:   if $T$ and $T'$ are well-separated then
        Translate $T'$'s expansion to $T$ as part of $T$'s near-field expansion
      end if
      Pass all near-field expansions for $T$ to its children
    end for
15: Compute any remaining interactions directly
```

simpler working definition for this dissertation. This is fine because our goal is to explore multi-tree algorithms in practice on specific problems of fundamental scientific interest, rather than a general theory of the topic. Therefore, we stick to a simpler definition and leave the full definition to future work.

At a high-level, a dual-tree algorithm applies divides the overall problem into subproblems via a space partitioning tree. The data points are indexed in a tree, and then the algorithm compares a pair of tree nodes at a time. This pair of nodes represents a subset of the original computation. For example, in the N-body problem, one node represents a set of points for which we are finding potentials and the other represents a set of points which are contributing to those potentials. The entire computation is given by Equation 2.2.1, and the subset represented by nodes S and T is given by

$$\forall q \in S; \Phi_q += \sum_{r \in T} \Phi(q, r) \tag{2.3.1}$$

The efficiency improvements in a dual-tree algorithm come from the ability to *prune*. If we are careful in our choice of subdivisions, we can eliminate or approximate some of these subdivisions with little work. In the FMM, the interaction between distant nodes is approximated using multipole expansions. In the dual-tree AllNN algorithm, a query reference pair can be eliminated entirely if the distance between them is greater than the distance to any candidate neighbor seen so far for the references.

This divide-and-conquer approach is possible because of the structure of the computation. For both the N-body problem and all nearest neighbors, consist of a reduction (sum and minimum, respectively) performed over one set for each of the elements in another set. In place of sum or minimum, we could decompose the computation in the same way for any associative and commutative operation. Similarly, the "for-all" over queries can be viewed as a commutative and associative operation, since the order in which we consider queries is irrelevant to the final result.

The following chapters will examine problems which consist of more than two operations, possibly done over more than two sets. The first example we will discuss in the proceeding chapters is the estimation of n-point correlation functions. As we will show, the fundamental computational task for 3-point correlation estimation is of the form

$$\sum_{i \in I} \sum_{j \in J} \sum_{k \in K} f(x_i, x_j, x_k) \tag{2.3.2}$$

This problem can be arbitrarily divided into distinct subproblems in the same way as the all nearest neighbor and N-body problems. Thus, we can naturally try to extend the dual-tree idea to this problem. Rather than considering a pair of tree nodes and attempting to prune the computation between them, we will look at sets of three nodes at a time. We will again choose a space-partitioning tree and formulate a pruning rule and a traversal pattern.

The central claim of this thesis is that this generalization is fruitful in exactly the same way as the original dual-tree idea – we can obtain algorithms which are both extremely fast in practice and provably efficient under suitable assumptions.

2.4 An Extended Example – Euclidean Minimum Spanning Trees

Before turning to multi-tree algorithms, we first present a more thorough example of dual-tree algorithms in action. A detailed look at the process of actually designing and implementing an efficient dual- or multi-tree algorithm will help to understand the algorithms in later chapters. We present a new algorithm for the fundamental and widely applied Euclidean Minimum Spanning Tree (EMST) problem.

Problem 2.4.1. *Given a set of points S in $\mathbb{R}^d$, find the lowest weight spanning tree in the complete graph on S with edge weights given by the Euclidean distances between points.*

With references in the literature as early as 1926, the MST problem is one of the oldest and most thoroughly studied problems in computational geometry [137]. In addition to this long-standing theoretical and algorithmic interest, the MST is useful for many practical data analysis problems. Many optimization problems can be posed as the search for the MST in a network [137]. The MST is also used as an approximation for the traveling salesman problem [82], in document clustering [178], analysis of gene expression data [49], wireless network connectivity [171], percolation analyses [21], and modeling of turbulent flows [161], among other areas. These problems are commonly solved in the Euclidean setting.

In particular, we are interested in using the EMST to compute hierarchical clusterings [68, 184]. One such clustering is obtained by deleting all edges longer than a specified cutoff in the MST, generating a clustering through the remaining connected components. By varying the scale of the cutoff, this generates a hierarchical clustering. In the clustering literature, this is often referred to as a *single-linkage clustering* and is frequently represented by a *dendrogram*. While the single-linkage clustering is very simple and can be sub-optimal for many applications, it can form the basis of more insightful clusterings. The single linkage clustering can be pruned to obtain more useful astronomical results [14]. MST's also form the inner loop for methods to identify non-parametric clusters in noisy data [179]. Furthermore, theoretically optimal clusterings can be obtained efficiently from the single-linkage clustering [10].

In astronomy, EMST-based clustering is used to analyze deep-space surveys and simulations of the early universe. Each level of single-linkage clustering is known as a *friend-of-friends* clustering [147, 9]. The EMST is used to identify dark matter haloes in simulations, which are believed to be crucial to galaxy formation [101]. Clustering is also applied to sky surveys to identify the super-large scale structure of the universe, which sheds light on the conditions of the early universe and the mechanisms of galaxy formation [14].

2.4.1 Related Work

In this work, we focus on the earliest known minimum spanning tree algorithm, Borůvka's algorithm, which dates from 1926. See [124] for a translation and commentary on Boruvka's original papers. As in Kruskal's algorithm, a minimum spanning forest is maintained throughout the algorithm. Kruskal's algorithm adds the minimum weight edge between any two components of the forest at each step, thus requiring $N - 1$ steps to complete. Borůvka's algorithm finds the minimum weight edge incident with each component, and adds all such edges, thus requiring at most $\log N$ steps and a total running time of $O(m \log n)$. We define the *nearest neighbor pair* of a component C as the pair of points $q \in C, r \notin C$ that minimizes $d(q, r)$. Finding the nearest neighbor pair for each component and adding the edges (p, q) to the forest is called a *Boruvka step.* Boruvka's algorithm then consists of forming an initial spanning forest with each point as a component and iteratively applying Boruvka steps until all components are joined.

Many MST algorithms rely on Tarjan's blue rule [166], which says the minimum weight edge across any edge cut is in the minimum spanning tree. This allows us to greedily form cuts in the graph and add the minimum weight edge across each. Algorithms using this rule include those attributed to Kruskal [98] and Prim [139], which require $O(m \log n)$ and $O(m + n \log n)$ time, respectively, on a graph with n points and m edges. Both algorithms maintain one or more components in spanning forest and use the cut between one component of the forest and the rest of the graph, adding the edges found in this way one at a time.

Shamos & Hoey [152] applied the Voronoi diagram to constructing the MST in the Euclidean plane. The Voronoi diagram can be constructed in $O(N \log N)$ time for N points and contains $O(N)$ edges. Since the MST is a subset of the edges in the dual of the Voronoi diagram, the MST can be found in $O(N \log N)$ time using one of the algorithms above. This bound worsens to $O(N^2 \log N)$ in three or more dimensions, fundamentally limiting this method to two dimensional cases. Preparata and Shamos [138] give a lower bound for the EMST problem of $\Omega(N \log N)$, which is the tightest known lower bound.

Bentley and Friedman [16] developed an EMST algorithm using *kd*-tree-based nearest neighbor searches to find the next edge to add in Prim's algorithm. While their method lacks a formally rigorous bound, they estimate that it requires $O(N \log N)$ time for most distributions of points. An alternate implementation of this approach is given in [125]. In 1982, Yao gave a bound of $O(N^{2-a(k)}(\log N)^{1-a(k)})$ where $a(k) = 2^{-(k+1)}$ for points in a k-dimensional metric space, along with a $O((N \log N)^{1.8})$ bound for points in three dimensions [182]. Agarwal *et al.* (1991) related the running time to the *bichromatic closest pair (BCP)* problem. Given a set of red and a set of blue points, the bichromatic closest pair is the red point r and blue point b such that $d(r, b)$ is minimized. They showed a bound of $O(F_d(N, N) \log^d(N))$, where $F_d(N, M)$ is the time to solve the BCP problem with N blue and M red points in d dimensions [3].

Callahan & Kosaraju's *Well-Separated Pair Decomposition (WSPD)* [24] forms the basis of the most recent EMST algorithms. The WSPD is defined as a set of pairs of nodes in a space-partitioning tree such that for each pair of points (p, q), we have $p \in P, q \in Q$ for exactly one pair of nodes (P, Q), and the the nodes in any pair are farther apart than the diameter of either node. It can be shown that the WSPD has $O(N)$ pairs of nodes, and that the MST is a subset of the edges formed between the closest pair of points in each pair of nodes. In [26], the authors use the WSPD to improve Agarwal and coworker's 1991 bound to $O(F_d(N, N) \log N)$. Their algorithm uses the WSPD-based nearest neighbor algorithm to compute neighbors of components for Boruvka's algorithm. The method identifies a list of pairs in the WSPD, for which bichromatic closest pair computations are performed to find edges of the MST. This algorithm is superficially similar to our method, but only locates neighbors for small components in each iteration. It also requires bookkeeping and connectedness queries which are not factored into the analysis, and no experimental results are shown.

Narasimhan *et al.* [123] implement a variant of this method, which they attribute to [26]. In this algorithm, GeoMST, they compute the BCP for each pair in the WSPD, then apply Kruskal's algorithm to the resulting edge set. They improve this method by postponing and avoiding some BCP computations and refer to the resulting algorithm as GeoMST2. This method can be successfully applied to point sets of any dimensionality; however, the constant in the $O(N)$ size of the WSPD grows exponentially in the dimension and is often very large in practice. The authors argue that the algorithm has an expected $O(N \log N)$ running time, but do not prove this rigorously. They also demonstrate favorable running times on several data sets.

These algorithms are the most sophisticated methods for the EMST problem in terms of both theoretical analysis and practical performance. The runtime bound in terms of the bichromatic closest pairs problem is the tightest available given optimistic runtimes for bichromatic closest pairs, but it is incomplete without bounding F_d. Bentley and Friedman's *kd*-tree-based method and the tree- and WSPD-based

GeoMST2 are the most practically viable algorithms. We return to these methods in our experimental analysis.

2.4.2 Dual-Tree Boruvka Algorithm

The running time of Borůvka's algorithm depends on an efficient method to find the nearest neighbor pair of each component. Here, we describe a method to compute all nearest neighbor pairs simultaneously by amortizing some computations across different points. This allows us to implement Boruvka's algorithm more efficiently than previous methods.

Our new algorithm, DualTreeBoruvka, uses a dual-tree method to find the nearest neighbor pair for each component. Algorithm 2.4.1 gives the description of the outer loop. The subroutine UpdateTree handles the propagation of any bounds up and down the tree and resets the upper bounds $d(C_q)$ to infinity. We also make use of a disjoint set data structure [166] to store the connected components at each stage of the algorithm. Our algorithm is independent of the particular space partitioning tree used. In this paper, we present experimental results on two instantiations of the algorithm. Algorithm 2.4.2 uses a *kd*-tree, and algorithm 2.4.3 uses the cover tree[20].

For the remainder of this work, we assume that we are given a set S of N points in $\mathbb{R}^d$. Furthermore, we make the standard assumption that all pairwise distances between points are unique. We make use of the following notation:

- $q \sim r$: q and r belong to the same component of the spanning forest.
- $R \bowtie Q$: all points in node R are in the same component as all points in node Q. Similarly, $r \bowtie q$ in a cover tree denotes that all descendants of r are connected to all descendants of q.
- C_q : the component of the forest containing q
- $d(C_q)$: distance to current nearest neighbor of component C_q (initialized to ∞).

Algorithm 2.4.1 Dual-Tree Borůvka (Tree root q)

```
    $E = \emptyset$
    while $|E| < N - 1$ do
3:      FindComponentNeighbors($q, q, e$)
        $E \leftarrow E \cup e$
        UpdateTree($q$)
6:  end while
```

- $e(C_q)$: edge from C_q to its candidate nearest neighbor
- $d(Q, R)$: the minimum distance between the bounding boxes of nodes Q and R

Each node Q maintains an upper bound $d(Q) = \max_{q \in Q} d(C_q)$ and records whether all the points belong to the same component of the spanning forest. A node where all points belong to the same component is referred to as *fully connected.*

Theorem 2.4.2. *The* FindComponentNeighbors *routine in Algorithm 2.4.2 returns the correct nearest neighbor pairs.*

Proof. The algorithm can only prune in two ways. If Q and R are fully connected, then no edges (q, r) with $q \in Q$ and $r \in R$ can be nearest neighbor pairs. The distance-based prune only occurs when for all $q \in Q$, $d(C_q) < d(Q, R)$. Therefore, all components with points in Q must have a candidate neighbor closer than any point in R, which again implies that no edge (q, r) can be a nearest neighbor pair. So, for each $q \in Q$, the correct Boruvka neighbor r of the component C_q cannot be pruned and must be found in the base case. □

The cover tree version of FindComponentNeighbors (Algorithm 2.4.3) follows the all nearest neighbor pseudocode given in [141]. The reference set R_i contains all points at level i that may have a nearest neighbor of a descendant of q_j as one of their descendants. Therefore, points are pruned from R_{i-1} in line 12 only when they are too distant to provide a neighbor. All descendants of q_j are within 2^{j+1} of q_j and all descendants of points in R are within 2^i of a point in R by the covering invariant.

Algorithm 2.4.2 FindComponentNeighbors(kd-tree node Q, kd-tree node R, Edge set e)

```
    if $Q \bowtie R$ then
      return
 3: else if $d(Q,R) > d(Q)$ then
      return
    else if $Q$ and $R$ are leaves then
 6:   for all $q \in Q, r \in R, r \not\sim q$ do
        if $d(q,r) < d(C_q)$ then
          $d(C_q) = d(q,r),\ e(C_q) = (q,r)$
 9:     end if
      end for
      $d(Q) = \max_{q \in Q} d(C_q)$
12: else
      FindComponentNeighbors($Q.left, R.left, e$)
      FindComponentNeighbors($Q.right, R.left, e$)
15:   FindComponentNeighbors($Q.left, R.right, e$)
      FindComponentNeighbors($Q.right, R.right, e$)
      $d(Q) = \max\{d(Q.left), d(Q.right)\}$
18: end if
```

Therefore, any point outside the bound in line 12 cannot be a nearest neighbor for descendants of q_j.

Theorem 2.4.3. *The* FindComponentNeighbors *routine in Algorithm 2.4.3 returns the correct nearest neighbor pair.*

Proof. For a query q_j being considered at level j, the algorithm must guarantee that it finds the nearest neighbor pair both for the component C_q and for all components $C_{q'}$, where q' is a descendant of q_j. Pruning a fully-connected node can never delete the true nearest neighbor pair.

We then consider distance-based pruning. As before, we use the nearest neighbor of the component C_q that the algorithm has seen up to this point in the execution. This candidate neighbor can be either a previously found nearest neighbor of another point in C_q (in which case $d = d(C_q)$), a point $r \in R$ ($d = d(q_j, r)$), or an inferred descendant of a connected point r ($d = d(q_j, r) + 2^i$). If $q_j \sim r$ but $q_j \not\bowtie r$, then r must have a descendant r' that is not connected to q_j. By the covering invariant,

Algorithm 2.4.3 FindComponentNeighbors(Cover tree node q_j, Reference Set R_i, Edge set e)

```
if $i = -\infty$ then
  // base case
  for all $q$ that are descendants of $q_j$ and $r \in R_i$ with $r \not\sim q$ do
    if $d(q, r) < d(C_q)$ then
      $d(C_q) = d(q, r),\ e(C_q) = (q, r)$
    end if
  end for
else if $j < i$ then
  // reference descend
  $R = \{r \in \text{Children}(r') : r' \in R_i \text{ and } r \not\bowtie q_j\}$
  $d = \min \left\{ d(C_q),\ \min_{\substack{r \in R \\ r \sim q_j}} \{d(q_j, r) + 2^i\},\ \min_{\substack{r \in R \\ r \not\sim q_j}} \{d(q_j, r)\} \right\}$
  $R_{i-1} = \{r \in R : d(q_j, r) \leq d + 2^i + 2^{j+2}\}$
  $d(C_q) = d$
  FindComponentNeighbors$(q_j, R_{i-1}, e)$
else
  // query descend
  for all $p_{j-1} \in \text{Children}(q_j)$ do
    FindComponentNeighbors$(p_{j-1}, R_i, e)$
  end for
end if
```

$d(q_j, r') \leq d(q_j, r) + 2^i$. Therefore, d is a valid upper bound for C_q. Since the distance between any point in R and any descendant is bounded by 2^i, any ancestor of the true nearest neighbor of q_j must be within $d + 2^i$, so the algorithm can never prune the ancestor of this neighbor.

We must also show that d is a valid bound for any descendant q' of q. If q and q' are in the same component, then this is clearly true, since bounds are shared across components. Otherwise, q is a candidate neighbor for q' and $d(q, q') \leq 2^{j+1}$. Therefore, we can be sure that $d(C_{q'}) \leq 2^{j+1}$. Let r' be the correct neighbor for q', and let r be the ancestor of r' in R. Then, $d(q_j, r) \leq d(q_j, q') + d(q', r') + d(r', r) \leq 2^{j+1} + 2^{j+1} + 2^i = 2^{j+2} + 2^i$. Therefore, the distance prune cannot remove the neighbor of any descendant of q. $\square$

2.4.3 Runtime Analysis

In this section, we prove our main theoretical result:

Theorem 2.4.4. *For a set S of N points in a metric space with expansion constant c, cluster expansion constant c_p, and linkage expansion constant c_l, the* DualTree-Boruvka *algorithm using a cover tree requires $O(N \log N \, \alpha(N)) \approx O(N \log N)$ time (where $\alpha(N)$ is defined below).*

2.4.3.1 Properties of the Data

The *expansion constant*, due to Karger and Ruhl [92], bounds the maximum increase in the density of points as a function of the distance from any point, and was used in adaptive analysis of nearest neighbors in previous work [20, 141].

Definition 2.4.5. *Let S be a set of points in a metric space (X, d). Let $B_S(p, r) = \{q \in S : d(p, q) \leq r\}$. Then, the* expansion constant *c of S is defined as the smallest c such that for all $p \in X$ and all $r > 0$*

$$|B_S(p, 2r)| \leq c|B_S(p, r)| \qquad (2.4.1)$$

While the expansion constant depends only on the pairwise distances between points, the MST has a “higher-order” structure. In other words, the MST depends on distances between clusters of points in addition to distances between the individual points. Since the expansion constant does not capture this structure, we define two new parameters: the *cluster expansion constant* and *linkage expansion constant.*

We first require a definition of clusters. Independently of how they are computed, successive Boruvka steps define a hierarchical clustering of the data. We can therefore define and use the *Boruvka clustering* without reference to any method for computing it.

Definition 2.4.6. *Given a point set S, the* Boruvka clustering *at level i, D_i, is the clustering obtained from applying a single Boruvka step to the clustering D_{i-1}. D_1 consists of each point as its own cluster.*

Given the Boruvka clustering, we define the new expansion constants. Let $B_i^c(q, r)$ be the set of all components C_p with a point $p \in C_p$ such that $d(q, p) \leq r$. Using this *component-wise ball*, we define the *cluster expansion constant.*

Definition 2.4.7. *The* cluster expansion constant *is the smallest real number c_p such that*

$$|B_i^c(q, 2r)| \leq c_p |B_i^c(q, r)| \tag{2.4.2}$$

for all points $q \in S$, distances $r > 0$, and each level of the Boruvka clustering D_i.

Let C_1 and C_2 be two clusters in the Boruvka clustering at level i and let $S_1 \subseteq C_1$ and $S_2 \subseteq C_2$. Let $B_i^l(S_1, S_2, r)$ be the set of all pairs (p, q) such that $p \in S_1$, $q \in S_2$, and $d(p, q) \leq r$.

Definition 2.4.8. *The* linkage expansion constant *is the smallest real number c_l such that*

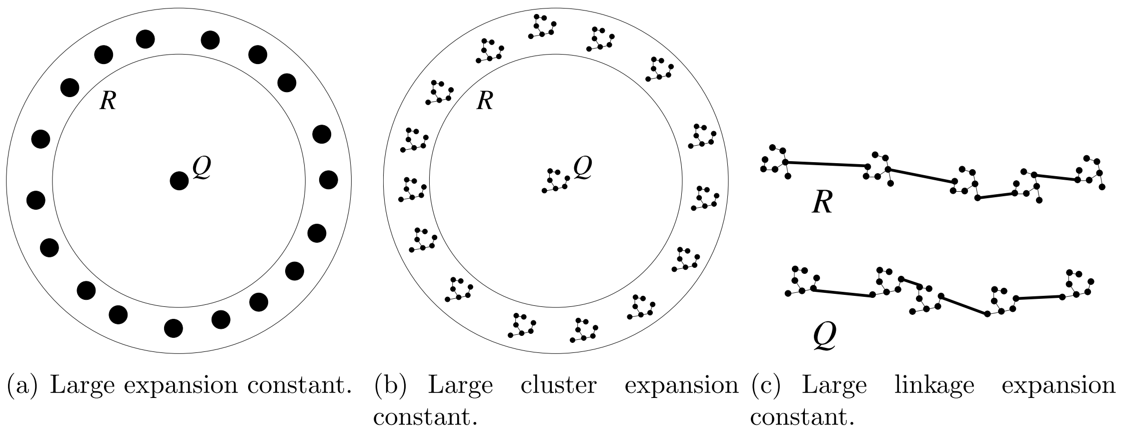
$$|B_i^l(S_1, S_2, 2r)| \leq c_l |B_i^l(S_1, S_2, r)| \tag{2.4.3}$$

for all levels of the Boruvka clustering D_i, clusters C_1 and C_2 at level i, subsets $S_1 \subseteq C_1, S_2 \subseteq C_2$, and distances $r > 0$.

We now turn to the proof of Theorem 2.4.4. We first require a few simple lemmas about cover trees.

Lemma 2.4.9. *(Width Bound) The number of children of any node in the cover tree is bounded by c^4.*

Lemma 2.4.10. *(Depth Bound) The maximum depth in the tree of any point in the explicit representation is $O(c^2 \log N)$.*



(a) Large expansion constant. (b) Large cluster expansion constant. (c) Large linkage expansion constant.

Figure 3: Cases illustrating the expansion constants used in the analysis of the DualTreeBoruvka algorithm.

Proof. Since each Borůvka step reduces the number of components in the spanning forest by a factor of at least two, the entire algorithm requires at most $\log N$ iterations. The construction of the cover tree takes $O(N \log N)$ time (proved in [20]) and only needs to be done once as a preprocessing step. Bookkeeping and cleanup in the tree in between calls to FindComponentNeighbors requires a single depth-first traversal, which takes $O(N)$ time.

Adding edges requires at most $O(N)$ Union operations on the disjoint-set structure, each of which requires $O(\alpha(N))$ time, with $\alpha(N)$ defined as follows. Let $A_k(j) = A_{k-1}^{(j+1)}(j)$ and let $A_0(j) = j + 1$. Then, define

$$\alpha(N) = \min\{k : A_k(1) \geq N\} \tag{2.4.4}$$

Therefore, in order to complete the proof, we only need to show that the FindComponentNeighbors subroutine on a cover tree requires $O(N\,\alpha(N))$ time. □

We now apply our adaptive analysis to bounding the runtime of FindComponentNeighbors.

Theorem 2.4.11. *Under the assumptions of Thm. 2.4.4, the* FindComponentNeighbors *algorithm on a cover tree (Algorithm 2.4.3) finds the nearest neighbor of each component in* $O(N\,\alpha(N)) \approx O(N)$ *time.*

Proof. We show that the amount of work done in each line of the algorithm during the entire execution is at most $O(\max_i |R_i| N\, \alpha(N))$. We complete the proof by showing $\max_i |R_i|$ depends only on c, c_p, and c_l.

Base Case. The base case (lines 1 through 7) is executed at most once for each explicit query node. Each base case requires $\max_i |R_i|$ FIND operations, each of which requires $O(\alpha(N))$ time, so this step takes $O(\max_i |R_i| \cdot N\, \alpha(N))$ time.

Query Descends. Each query node in the explicit representation is expanded at most once (line 18), so this step requires $O(N)$ time overall.

Reference Descends. On the other hand, a reference node may be expanded more than once. When a query node is expanded, its reference cover set R_i needs to be duplicated for each child of the query. By the width bound, this creates at most c^4 duplications. Therefore, the total number of reference nodes considered in Line 12 is $O(c^4 N)$.

At each level, $|R| \leq c^4 \max_i |R_i|$. Since the maximum depth of a node is $O(c^2 \log N)$ (depth bound), the number of nodes considered in Line 14 is $O(c^6 \max_i |R_i| \log N)$. Considering possible duplication across queries, the total number of calls to Line 14 is at most $O(c^{10} \max_i |R_i| \log N)$. Computing d in each reference descend involves checking the connectedness of q_j and r, which requires $O(\alpha(N))$ time, for a total running time of $O(c^{10} \max_i |R_i| \log N\, \alpha(N))$.

Bounding $|R_i|$. For a given query q_j and reference cover set R_i, we compute the upper bound distance d. Then, $R_{i-1} = \{r \in R : d(q_j, r) \leq d + 2^i + 2^{j+2}\}$. Since $j < i$ in this part of the algorithm, and since the query and reference trees are identical, $j = i - 1$. Therefore, $B(q_j, d + 2^i + 2^{j+2}) = B(q_j, d + 2^{i+1} + 2^i)$.

Consider two cases: first let $d \leq 2^{i+2}$. Then, as in [20], we bound number of balls of radius 2^{i-2} that can be packed into $B(q_j, d + 2^{i+1})$ by:

$$\begin{aligned}
&|B(q_j, d + 2^{i+1} + 2^i + 2^{i-2})| \\
&\quad \leq \; |B(p, 2(d + 2^{i+1} + 2^i) + 2^{i-2})| \\
&\quad \leq \; |B(p, 2^{i+4})| \\
&\quad \leq \; c^6 |B(p, 2^{i-2})|
\end{aligned}$$

Each ball of radius 2^{i-2} can contain at most one point in C_{i-1} by the separation invariant. Therefore, the number of points in $B(q_j, d + 2^{i+1} + 2^i) \cap C_{i-1} \subseteq R_{i-1}$ is at most c^6.

Consider the other case where $d > 2^{i+2}$. Without loss of generality, assume that we have computed k previous iterations. First note that all points within $B(q_j, d - 2^{i+1})$ must be connected to q_j. Otherwise, let q' be a point in $B(q_j, d - 2^{i+1})$ that is not connected to q_j. Then, q' has a grandparent q'' at level C_{i-1} such that $d(q', q'') \leq 2^i$. Therefore,

$$d(q_j, q'') \leq d(q_j, q') + d(q', q'') < d - 2^{i+1} + 2^i = d - 2^i$$

Therefore, $d(q_j, q'') + 2^i < d$ and $q_j \not\bowtie q''$, which contradicts the definition of d in line 11.

The number of components that q_j may have to search is bounded by

$$\begin{aligned}
|B_p^c(q_j, d + 2^{i+1} + 2^i)| \;&\leq\; |B_k^c(q_j, 2d)| \\
&\leq\; c_p^2 |B_k^c(q_j, d/2)| \\
&\leq\; c_p^2 |B_k^c(q_j, d - 2^{i+1})|
\end{aligned}$$

As noted above, all points within $d - 2^{i+1}$ of q_j are connected to q_j, so the only component in $B_k^c(q_j, d - 2^{i+1})$ is C_q.

We now bound the number of points within a component that q_j may have to consider. Let C_r be a component distinct from C_q. Let $L(q_j)$ denote the set of all leaves that are descendants of q_j. Let $d' = \min_{q \in L(q_j), r \in C_r} d(q, r)$. Then,

$$\begin{aligned}
&|B_k^l(C_q \cap L(q_j), C_r, d + 2^{i+1} + 2^i)| \\
&\quad \leq \ |B_k^l(C_q \cap L(q_j), C_r, 4(d - 2^{i+1}))| \\
&\quad \leq \ c_l^2 |B_k^l(C_q \cap L(q_j), C_r, (d - 2^{i+1})| \\
&\quad \leq \ c_l^2 |B_k^l(C_q \cap L(q_j), C_r, d')|
\end{aligned}$$

By the above argument, there can be at most one pair in $B_k^l(C_q \cap L(q_j), C_r, d')$. Therefore, there are at most c_l^2 points in C_r contained in $B(q_j, d + 2^{i+1} + 2^i)$. In the worst case, each of these points is at level C_{i-1} of the tree and must be considered in R_{i-1}. There are at most c_p^2 components C_r that can contribute points, so the maximum number of points in R_{i-1} is $c_p^2 c_l^2$.

Combining these cases, we have $\max_i |R_i| \leq \max\{c^6, c_p^2 c_l^2\}$. Therefore, the running time is:

$$\begin{aligned}
O\big(N + c^4 N + \max\{c^6, c_p^2 c_l^2\} \cdot N\, \alpha(N) \\
+ \max\{c^6, c_p^2 c_l^2\} \cdot c^{10} \log N\, \alpha(N)\big)
\end{aligned}$$

which completes the proof. □

2.4.4 Results

We present results for *kd*-tree-based and cover-tree-based DualTreeBoruvka. For comparison, we implemented the other fast EMST methods mentioned in section 2.4.1. Specifically, we compare against the single-fragment EMST algorithm from Bentley and Friedman [16], which is an implementation of Prim's algorithm. The algorithm uses a single-tree algorithm on a *kd*-tree to find the next edge to add at each step. We also show results for the WSPD-based algorithm GeoMST2 [123], described above.

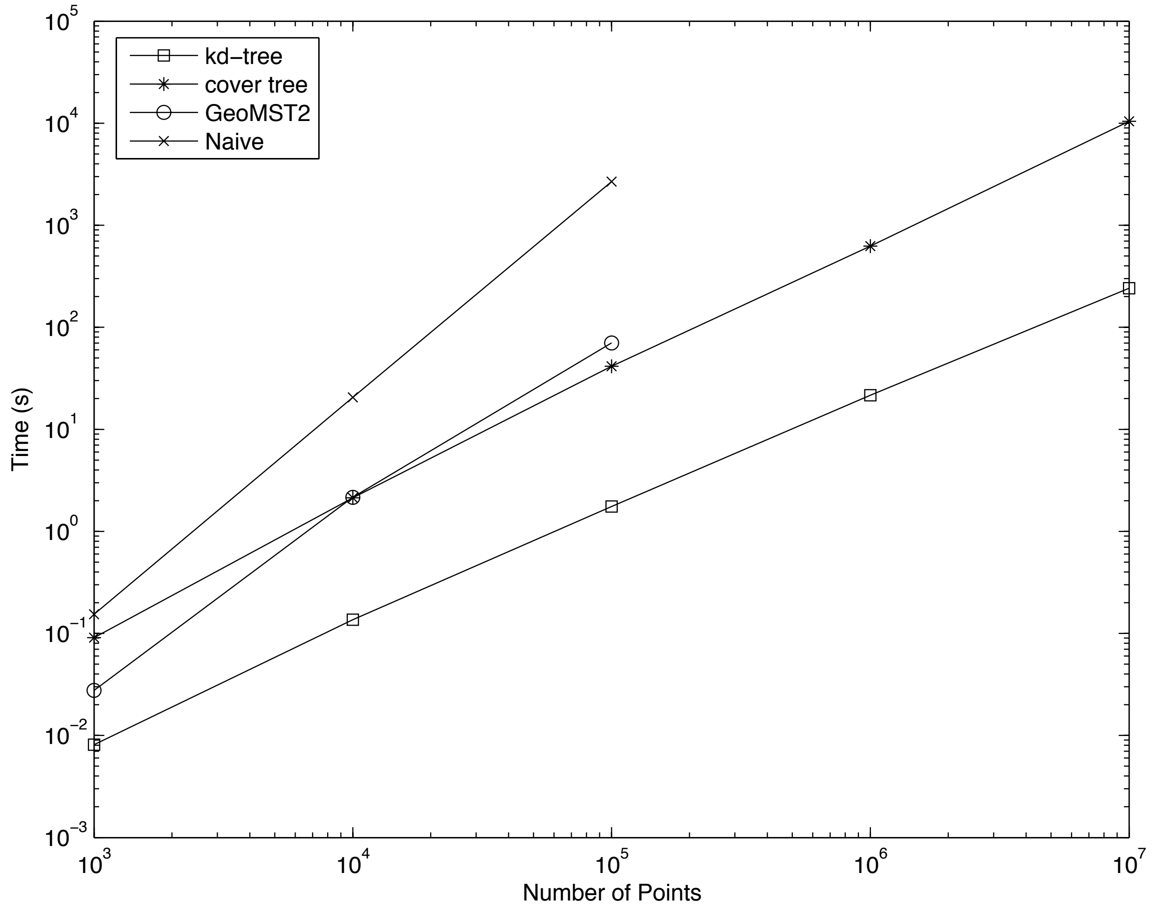


Figure 4: Runtimes of EMST construction algorithms. The data are generated from a mixture of 10 Gaussians in three dimensions. The *kd*-tree and cover tree results refer to the DualTreeBoruvka algorithm on those trees. Times are in seconds.

Finally, we compare against a naïve implementation of Boruvka's algorithm in which nearest neighbor pairs are computed by iterating over all pairs of points.

The experiments here are on four datasets: one synthetic and three sets of astronomy data. The synthetic data are drawn from a mixture of ten evenly weighted Gaussians placed uniformly at random in the unit cube in three dimensions. Figure 4 compares timing results on these data. Figure 5 shows runtimes on four dimensional samples of spectral data from the Sloan Digital Sky Survey. Table 1 has results for two other astronomy datasets: a 40,000 point, 3,840-dimensional set of color spectra from the SDSS, and a million point, 3 dimensional set of (x, y, z) coordinates from a galaxy-formation simulation.

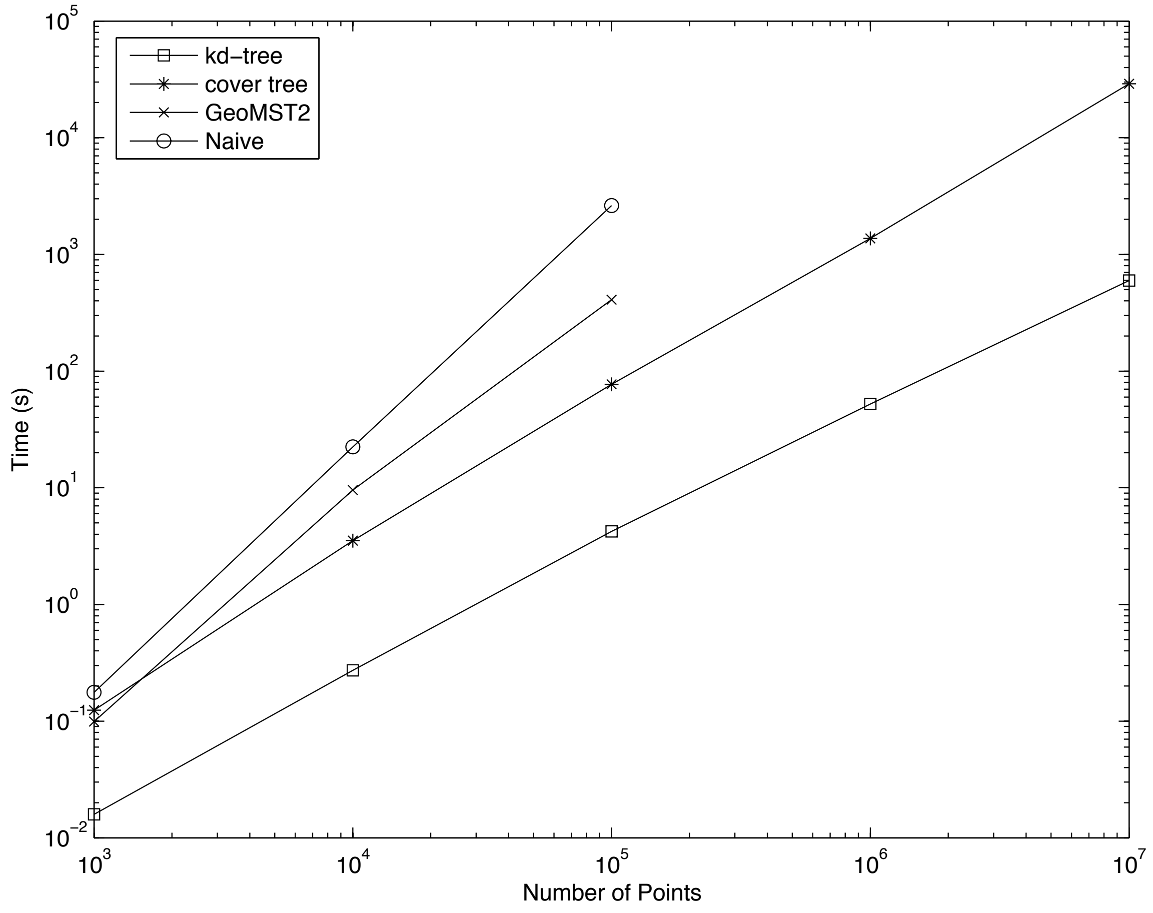


Figure 5: Runtimes of EMST construction algorithms. The *kd*-tree and cover tree results refer to the DualTreeBoruvka algorithm on those trees. The data are four dimensional spectra from the SDSS. Times are in seconds.

The algorithms are implemented using the MLPACK machine learning library, and are currently available as part of the library [39]. The code was compiled with gcc version 4.1.2 with the -O2 flag. All experiments were performed on a 3.0 GHz Intel Xeon processor with 8GB of RAM running Linux.

We attempted to run all the algorithms on all the sets of data. However, the naïve experiments were limited by time, since the brute-force algorithm scales quadratically. Thus results are missing for the larger sets. The GeoMST2 algorithm is limited by available memory. Although the WSPD contains $O(N)$ pairs of nodes, the constant factor can be very large. The constant in the $O(N)$ analysis scales exponentially with the dimension [24], so the storage bottleneck becomes tighter with higher-dimensional data. Missing timings for GeoMST2 indicate that the available memory was exceeded. In our experiments, the Bentley-Friedman algorithm is more efficient than either of these.

In both the synthetic data (Figure 4) and the SDSS data (Figure 5), DualTree-Boruvka on a *kd*-tree is the fastest method, by a factor of 2.8 and 4.6 over the Bentley-Freidman method, respectively. On both figures, we plot the slope of the predicted $N \log N$ performance, scaled to align with the timings for our method.

Our results also consider dimensionality of the data. In the three- and four-dimensional data given in Figures 4 and 5, the *kd*-tree based DualTreeBoruvka is fastest. Unlike most EMST algorithms, our method can also efficiently handle high-dimensional data, as shown in table 1. For the high-dimensional SDSS data, the two methods using *kd*-trees require roughly the same time. DualTreeBoruvka on a cover tree, however, is faster by a factor of 2.9.

2.5 Conclusion

We have developed the core concepts of dual- and multi-tree algorithms through several examples. Having illustrated the details of dual-tree algorithm development

Table 1: Comparison of DualTreeBoruvka and Bentley-Friedman timings for EMST construction. The first row is a set of spectra from the Sloan Digital Sky Survey. The second is a synthetic data set from a mixture of 10 Gaussians. Timings are in seconds.

N	dim	DTB *kd*	DTB cover	BF [16]
40,000	3840	45825.18	**15791.37**	45780.43
1,000,000	3	**17.39**	333.45	42.54

through the EMST example, we now turn to the core contributions of this dissertation: new multi-tree algorithms for important scientific problems.

CHAPTER III

N-POINT CORRELATION FUNCTIONS

We now turn to our first application of multi-tree algorithms – estimation of the n-point correlation functions. 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, 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. Since scientific data sets are large and growing, a more efficient solution is crucial.

In this chapter, we demonstrate that multi-tree algorithms can be applied to the npcf estimation problem. We present a comprehensive study of the npcf estimation problem show a re-implemented and optimized multi-tree algorithm for the *raw correlation counts* problem, the bottleneck computational task in npcf estimation (defined below), originally introduced in [74, 120]. We then show several new contributions:

- We present a new multi-tree algorithm capable of estimating the npcf at many different scales simultaneously.
- We show a new algorithm for efficient variance estimation of the npcf using the jackknife which can be used in conjunction with our multi-tree algorithms.

- We show detailed CPU optimizations of the base case of our algorithm.
- We discuss our open-source massively parallel distributed implementation of all of these algorithms.
- We present detailed adaptive analyses of our algorithms, showing for the first time that the n-point correlation can be estimated in $O(N)$ time under assumptions that hold for computations of real scientific interest.
- We demonstrate the largest full 3-point correlation estimation computation on the large scale galaxy structure to-date.

We demonstrate that each of our algorithmic improvements and the optimized base case offer speedups of an order-of-magnitude or more. Taken together, these contributions make previously impossible studies of the npcf of the galaxy distribution tractable on commodity clusters. Much of this work has been previously published in KDD [113] and Supercomputing [114].

3.1 Applications of the N-Point Correlation Functions

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 [130], fluctuations in the cosmic microwave background [165], the formation of clusters of galaxies [177], and the characterization of the galaxy-mass bias [118]. They can be used to compare observations to theoretical models through perturbation theory [12, 60]. A high-profile example of this was a study showing large-scale evidence for dark energy [66] – this study was written up as the Top Scientific Breakthrough of 2003 in *Science* [150]. 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.

The materials science community also makes extensive use of the n-point correlation functions. They are used to form three-dimensional models of microstructure [109] and to characterize that microstructure and relate it to macroscopic properties such as the diffusion coefficient, fluid permeability, and elastic modulus[168, 167]. The n-point correlations have also been used to create feature sets for medical image segmentation and classification [143, 121, 34].

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 [145] 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 [130]. 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.

3.2 The N-Point Correlation Functions

We begin by defining the npcf. We start with a definition of the problem setting: *point processes*. We consider our data set to be a single sample drawn from some ensemble of possible samples. We then define the n-point correlation functions themselves, which describe the expected clustering properties of samples from the ensemble. We then turn to the algorithmic problem of interest in this chapter: the task of estimating the npcf from a real data set.

3.2.1 Spatial Data and Spatial Statistics

We now define the n-point correlation functions. We provide a high-level description; for a more thorough defintion, see [130, 163]. Once we have given a simple description of the npcf, we turn to the main problem of interest here: 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.

Throughout, we deal with *point processes*. Our sample consists of a finite set of points D in a compact, bounded subset of $\mathbb{R}^d$, sometimes referred to as the sample window. This set is drawn from some ensemble of possible realizations. Note that we do not assume that the individual points are independent, only that our sample is a fair representative of the ensemble of all possible samples. Therefore, rather than averaging over individual points, we must average over sample point sets drawn from this ensemble. In general, when making inferences about such a process, we invoke an ergodic hypothesis. Under this assumption, averages taken over distant parts of the sample window approximate averages over different samples from the ensemble. We can then make inferences from a single data set, assuming that it is large enough to allow averaging over distant regions.

Throughout our discussion, we assume that the process is homogeneous and isotropic. This is standard practice when studying astronomical data sets, since all cosmological models make this assumption as well. In other applications, such as materials science, this assumption may not hold. However, 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.

3.2.2 Defining the N-Point Correlation Functions

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. Consider a small volume element dV. In the limit as dV becomes small, the probability of finding more than one point in D in dV becomes negligible. Therefore, we can say that

$$dP \propto \rho dV \tag{3.2.1}$$

with only a normalization constant necessary to make this an equality. If the density ρ completely characterizes the process, we refer to it as a Poisson process.

The assumption of homogeneity and isotropy does not require the process to lack structure. The positions of points may still be correlated. This is illustrated in Figure 6. In a Poisson process, the joint probability of finding objects in disjoint volume elements dV_1 and dV_2 separated by a distance r is given by:

$$dP_{12} \propto \rho^2 dV_1 dV_2 \tag{3.2.2}$$

We obtain this directly from Equation 3.2.1 by noting that in a Poisson process, the events that a point lies in dV_1 and one lies in dV_2 are independent. Therefore, the joint probability of the two events is just the product of their individual probabilities.

However, in general, a point process does not need to be Poisson. The two events, finding a point in dV_1 and another point in dV_2, can be dependent. Let $\mathbf{r}$ be the vector pointing from dV_1 to dV_2[1]. We can modify Equation 3.2.2 to reflect this fact

[1]Note that we are working in the limit as dV_i becomes small, so the precise choice of endpoints of the vector within the regions is not significant.

as follows:

$$dP_{12} \propto \rho^2 dV_1 dV_2 (1 + \xi(\mathbf{r})) \tag{3.2.3}$$

The expression $(1 + \xi(\mathbf{r}))$ is the *two-point correlation function* (2pcf). The quantity $\xi(\cdot)$ is sometimes called the *reduced* 2pcf, or when no confusion can result, just the two-point correlation function. If $\xi(\mathbf{r}) = 0$, then we recover the definition in Equation 3.2.2. Positive values of $\xi(\cdot)$ indicate that the events are correlated, while negative values indicate anti-correlation.

Since we have assumed that the point process is homogeneous and isotropic, then the 2pcf must be as well. Therefore, we can write $\xi(\cdot)$ as a function of only the distance $\|r\|$. In other words, the 2pcf does not depend on the particular choice of small regions or on their position within the sample window, but only on the distance between them. We therefore have a continuous function of a single variable, the distance between the two small regions of interest.

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. 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:

$$dP_{123} \propto \rho^3 dV_1 dV_2 dV_3 (1 + \xi(r_{12}) + \xi(r_{23}) + \xi(r_{13}) + \zeta(r_{12}, r_{23}, r_{13})) \tag{3.2.4}$$

As above, 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 correlation functions (such as the 4-point correlation 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.

$$dP_{1234} = \rho^4 dV_1 dV_2 dV_3 dV_4 \left[1 + \sum \xi(r_{ij}) + \sum \zeta(r_{ij}, r_{ik}, r_{jk}) + \sum \xi(r_{ij})\xi(r_{kl}) + \eta\right] \tag{3.2.5}$$

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 a set of $\binom{n}{2}$ pairwise distances. We refer to this set of pairwise distances as a *configuration*, or in the computational context, as a *matcher* (see below).

We have defined a hierarchy of correlation functions of increasing complexity. These can be viewed as analogs of the moments of a single random variable, X. While 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.

Like the moments for a univariate distribution, the collection of n-point correlation functions can completely characterize any point process. Ripley [145] showed that any point process can be completely determined by the distribution of counts in cells. We can define the point process by specifying the probability distribution $P(N(A) = n)$ for all bounded areas (or volumes) A and integers n. The distribution of counts in cells can in turn be shown to be completely determined by the set of n-point correlation functions [130].

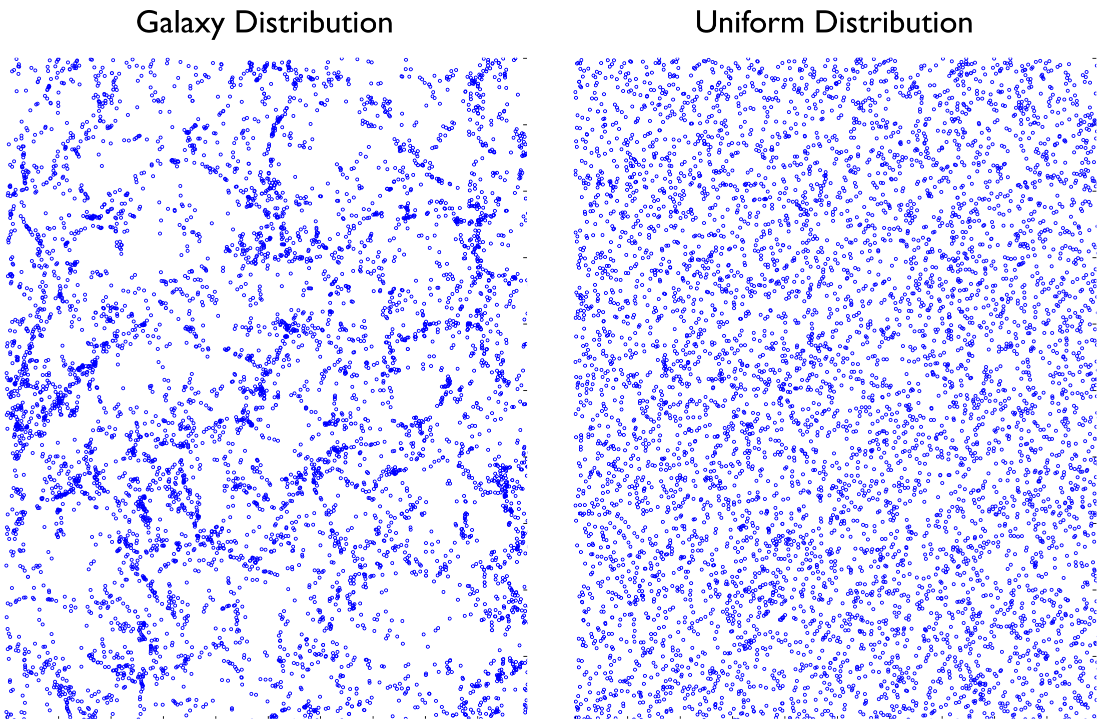


Figure 6: A visualization of two data sets with different correlation functions. Both data sets consist of approximately 10^6 points contained in a cube with sides of length 1000 Mpc/h. In both cases, we have taken a subsample in a slice of the sample window of thickness 5 Mpc/h in the z and projected the points in the slice onto the x-y plane. The figure on the left shows data from an N-body simulation of the distribution of galaxies. The ensemble generating this distribution has non-zero 2 and 3-point correlation. The data on the right are from a Poisson ensemble, with all n-point correlations equal to zero. Both images contain approximately 6,500 points.

3.2.3 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 chapter: the problem of estimating the n-point correlation from real data. We describe several commonly used estimators and identify their common computational tasks.

Note that the (reduced) npcf $\xi^{(n)}(\cdot)$ is a function of the underlying point process. In general, we do not directly have any information on this process. Instead, we are given one or more point sets drawn from the process. We must therefore use these data to *estimate* the npcf.

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.

Let 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 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 [130, 163]:

$$\hat{\xi}(r) = \frac{\mathrm{DD}(r)}{\mathrm{RR}(r)} - 1 \tag{3.2.6}$$

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 the completely random (Poisson) distribution, we are likely to obtain a positive estimate for ξ. Conversely, if our data are drawn from a Poisson or nearly Poisson distribution, this estimator will be close to zero, as in Equation 3.2.2.

This simple estimator suffers from suboptimal variance and sensitivity to noise. Several alternative estimators utilize the same underlying Monte Carlo idea, while providing improved variance and shot noise sensitivity.

The Landy-Szalay estimator [103]

$$\hat{\xi}(r) = \frac{DD(r) - 2DR(r) + RR(r)}{RR(r)} \tag{3.2.7}$$

is a widely used improvement over Equation 3.2.6. 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 denotes data-random pairs).

Another popular estimator for the 2pcf is due to Hamilton [78].

$$\hat{\xi}(r) = \frac{DD(r) \cdot RR(r)}{DR(r)^2} \tag{3.2.8}$$

Note that both these alternatives use the same quantities as Equation 3.2.6, namely counts of pairs of points satisfying a distance constraint.

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 characteristics [164]. We again generate points from a Poisson distribution, and the 3pcf estimator is also a function of quantities of the form DDD, DDR, DRR, or RRR. These refer to the number of unique triples of data and random points whose pairwise distances lie close to given constraints, again normalized by the total number of such triples. For the three point correlation, the Szapudi-Szalay estimator [164] is

$$\hat{\zeta}(\cdot) = \frac{DDD - 3DDR + 3DRR - 3RRR}{RRR} \tag{3.2.9}$$

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 set of $\binom{n}{2}$ pairwise distance, pair of points. We count unique tuples of points whose pairwise distances are close to the distances in the matcher in some ordering. Note that a tuple of points may satisfy the constraints when their pairwise distances are considered in one order but not in another. We count such tuples exactly once.

We make use of the following (symbolic) notation.

$$[D-R]^n = \sum_{i=0}^{n} D^i R^{n-i} \tag{3.2.10}$$

Then, the estimator is given by

$$\hat{\xi^{(n)}}(\cdot) = \frac{[D-R]^n}{R^n} \tag{3.2.11}$$

3.3 The Computational Task

Unfortunately, directly estimating the n-point correlation functions is extremely computationally expensive. In principle, computing a count $D^n(\mathbf{r})$ requires us to enumerate all unique n-tuples of 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 [177]. 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

[119]. Higher-order correlations have been even more restricted by computational considerations.

Note that all the estimators described previously depend on the same fundamental quantities: the number of tuples of points from the data/random set that satisfy some set of distance constraints. Thus, our task is to efficiently compute this number given the data and a suitably large Poisson set. Clearly, a direct approach, one which enumerates all n-tuples of points, will require $O(N^n)$ work. However, the scales of interest for the npcf estimators are much smaller than the sample window containing the data. Therefore, most tuples will not be counted and we can take advantage of this observation to formulate a more efficient algorithm.

3.3.1 Specifying Distance Constraints and Satisfying Matchers

Above, we described the npcf in terms of a set of $\binom{n}{2}$ distance constraints. We now make this idea more concrete. 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 . We sometimes refer to an entry as simply r_{ij}, with the upper and lower bounds being understood.

We can write a matcher in a matrix form:

$$\mathbf{r} = \begin{bmatrix} \cdot & r_{12} & r_{13} & \cdots & r_{1n} \\ r_{12} & \cdot & r_{23} & \cdots & r_{2n} \\ \ddots & \ddots & \ddots & \ldots & \vdots \\ r_{1n} & r_{2n} & r_{3n} & \cdots & \cdot \end{bmatrix} \tag{3.3.1}$$

If we specify upper and lower bounds, we would have two such tables – one for the lower and upper bounds. The matrix is symmetric, and the diagonal entries are left unspecified.

As an alternative to individual upper and lower bounds, we can specify a thickness dr. This thickness can depend on the scale of the matcher or not, as needed. We then specify a single matrix of constraints, and the upper and lower bounds are understood to be $r_{ij} \pm dr/2$.

When no confusion will arise, we will simply denote a matcher as $\mathbf{r}$, where the vector is understood to contain either the $2 \cdot \binom{n}{2}$ lower and upper-bound distance constraints or $nchoose2$ distance constraints a thickness.

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 3.3.1. *Given an n-tuple of points $(p_1, \ldots, p_n)$ in $\mathbb{R}^d$ 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*

$$r^{(l)}_{\sigma(i)\sigma(j)} < \|p_i - p_j\| < r^{(u)}_{\sigma(i)\sigma(j)} \tag{3.3.2}$$

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

Equivalently, in the case of upper and lower bound matchers, we have that

$$r_{\sigma(i)\sigma(j)} - \frac{dr}{2} < \|p_i - p_j\| < r_{\sigma(i)\sigma(j)} + \frac{dr}{2} \tag{3.3.3}$$

3.3.2 Computational Tasks

We can now identify the central computational task in npcf estimation: *Raw Correlation Count.*

Definition 3.3.2. ***Computational Task 1: Raw Correlation Counts.*** *Given a matcher $\mathbf{r}$, data set D, random set R, and integer $0 \leq i \leq n$, the task of computing $D^{(i)}R^{(n-i)}(\mathbf{r})$: the number of unique n-tuples of points, i from D, $n - i$ from R, such that the tuple satisfies the matcher is the **Raw Correlation Count** task.*

The *Raw Correlation Count* problem is the central computational bottleneck in npcf estimation.

Note that computing one of the estimators given above (such as Equation 3.2.11) simply requires solving Computational Task 1 $n+1$ times. Therefore, the work required to compute a point estimate of the npcf will be directly proportional to the work required to solve Computational Task 1.

The estimators above give us a value $\widehat{\xi}^{(n)}(\mathbf{r})$ at a single configuration. However, the n-point correlations are continuous quantities of each distance constraint in the matcher. In general, we will need to understand the behavior of the npcf at a variety of scales and configurations. Since we do not have a method to estimate the continuous npcf directly, we must estimate it at several discrete points. This requires us to repeatedly compute an estimator such as Equation 3.2.11 for many different matchers. Clearly, we will obtain a more precise view of the continuous npcf with more point estimates.

This leads us to our second computational task.

Definition 3.3.3. ***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 matcher $m \in M$.*

This task requires us to repeat Task 1 $O(M)$ times, where M controls the smoothness of our overall estimate of the npcf and our quantitative picture of its overall behavior. Therefore, the total work is $O(M \cdot T(N)$, where $T(N)$ is the time required to perform Task 1 for N data points.

As noted above, our data are a single sample from the underlying point process. We invoke an ergodic assumption, which allows us to use averages over our large data set in place of averages over many samples from the point process. Ideally, we would have several samples from the point process. We could then repeat the npcf estimation on each set, and from these estimates obtain the sample variance of our estimator.

However, additional samples are often prohibitively expensive or even impossible to obtain. For instance, if we model the distribution of galaxies as a point process, we cannot obtain any sample other than the one available for us to observe in the universe. Data sets obtained from N-body simulations may require enormous computational resources, preventing us from generating more than one. Therefore, we must use a bootstrapping method for internal variance estimation.

Jackknife resampling is a widely used variance estimation method [153] and is popular with astronomical data [111]. It is also used to study large scale structure by identifying variations in the npcf across different parts of the sample window [119]. 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 3.3.4. ***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 $D^{(i)}_{(-k)}R^{(j)}(\mathbf{r})$.*

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.

3.3.3 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 $O(J \cdot M \cdot T(N - \frac{N}{J})$ time.

We now turn to efficient algorithms for each of these computational tasks. We begin with a multi-tree algorithm for Task 1, which reduces the work $T(N)$ from $O(N^n)$ to a tractable amount. We then discuss new algorithms for Tasks 2 and 3 which extend our multi-tree algorithms for massive speedups.

3.4 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 [146] and [37] for an overview and further references).

Counts-in-cells [164] and Fourier space methods [133, 134] 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 [163] and cannot fully account for errors due to the edge of the sample window.

Methods to approximate the npcf have also been developed. Some methods use a Fourier transform-based approach to approximate the counts required for npcf estimators [187]. While this approach can be fast, it introduces additional errors into the statistic due to both the Fourier transform and the inexact nature of the counts computed. Other methods compute the approximate npcf using space-partitioning trees [185]. However, given the scientific importance of the results being investigated with the npcf and the sensitivity of the results, it is crucial to reduce all possible sources of error. Therefore, we confine our attention to exact methods for computing the npcf.

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 [74, 120]. It has been parallelized using the Ntropy framework [65]. This is the currently fastest implementation of general n-point correlation estimation and was used to compute the largest correlations of galaxy positions prior to this work [117, 118].

3.5 Algorithms

We have identified the full computational task of n-point correlation estimation. We now turn to our new algorithmic contributions.

3.5.1 Basic Multi-Tree Algorithm

We build on previous, tree-based algorithms for the n-point correlation estimation problem [74, 120]. The key idea is to employ multiple *kd*-trees to improve on the $O(N^n)$ scaling of the brute-force approach.

For simplicity, we begin by considering the two-point correlation estimation (Alg. 3.5.1). Recall that the task is to count the number of unique pairs of points that satisfy a given matcher. As in the dual-tree nearest neighbor and EMST algorithms discussed in Chapter 2, we build a tree on the data set. We then traverse the tree and use the bounding information stored in each node to identify opportunities for pruning.

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

Algorithm 3.5.1 DualTree2pt (Tree node T_1, Tree node T_2, matcher $\mathbf{r}$)

```
if $T_1$ and $T_2$ are leaves then
  for all points $p_1 \in T_1, p_2 \in T_2$ do
    if $r_{12}^{(l)} < \|p_1 - p_2\| < r_{12}^{(u)}$ then
      result += 1
    end if
  end for
else if $T_1$ and $T_2$ are both reference or data nodes and $T_1 < T_2$ then
  Prune to avoid overcounting
else if $d^{\min}(T_1, T_2) > r_{12}^{(u)}$ or $d^{\max}(T_1, T_2) < r_{12}^{(l)}$ then
  Prune
else
  DualTree2pt($T_1$.left, $T_2$.left)
  DualTree2pt($T_1$.left, $T_2$.right)
  DualTree2pt($T_1$.right, $T_2$.left)
  DualTree2pt($T_1$.right, $T_2$.right)
end if
```

(or four) resoling pairs of nodes. If our recursion reaches leaf nodes, we consider 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 and T_1 and T_2 are nodes on a tree built on the same data set. This ensures that we see each pair of points at most once.

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. 3.5.1. First, we must do more work to determine if a particular tuple of points satisfies the matcher. We accomplish this in Alg. 3.5.3 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

Algorithm 3.5.2 MultiTreeNpt (Tree node $T_1, \ldots,$ Tree node T_n, matcher $\mathbf{r}$)

```
    if all nodes $T_i$ are leaves then
      for all points $p_1 \in T_1, \ldots,$ points $p_n \in T_n$ do
        if TestPointTuple($p_1, \ldots, p_n, \mathbf{r}$) then
          result += 1
5:      end if
      end for
    else if not TestNodeTuple($T_1, \ldots, T_n, \mathbf{r}$) then
      Prune
    else
10:   Let $T_i$ be the largest node
      MultiTreeNpt($T_1, \ldots, T_i$.left$, \ldots, T_n, \mathbf{r}$)
      MultiTreeNpt($T_1, \ldots, T_i$.right$, \ldots, T_n, \mathbf{r}$)
    end if
```

at least one valid assignment. The second extension is a similar one for checking if a tuple of nodes can be pruned (Alg. 3.5.4). 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.

We can also consider more efficient ways of traversing the tree. We give one such method in Alg. 3.5.5. Consider a matcher $\mathbf{r}$. Let $\hat{r}^{(l)} = \min r_{ij}^{(l)}$ and $\hat{r}^{(u)} = \max r_{ij}^{(u)}$. Then, we know that if a pair of nodes are separated by more than $r_{ij}^{(u)}$ (or less than $r_{ij}^{(l)}$), any tuple of nodes that contains this pair can be pruned. Using this observation, we can avoid the full tree traversal in Alg. 3.5.2. Instead, we proceed in two phases. First, for each tree leaf, we construct an interaction list containing all leaves at a distance in the range $[\hat{r}^{(l)}, \hat{r}^{(u)}]$, which can be done efficiently a simple dual-tree traversal. Then, for each leaf T_l, we iterate through all unique n-tuples of nodes consisting of T_l and $n-1$ nodes from its interaction list. We can then compute the base case on this node tuple.

This algorithm can provide some efficiency gains over the full multi-tree algorithm. However, the full advantage of this algorithm becomes apparent when it is combined

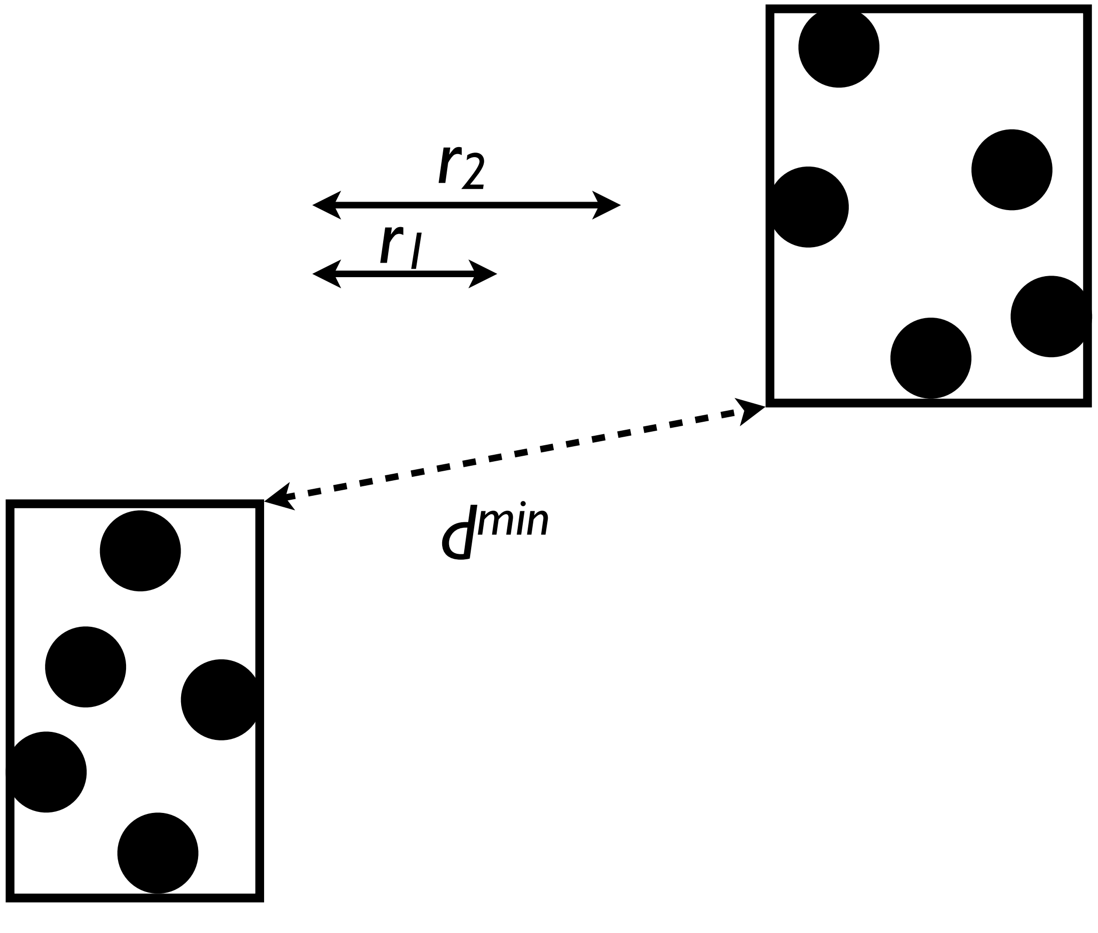


Figure 7: An illustration of pruning multiple matchers simultaneously.

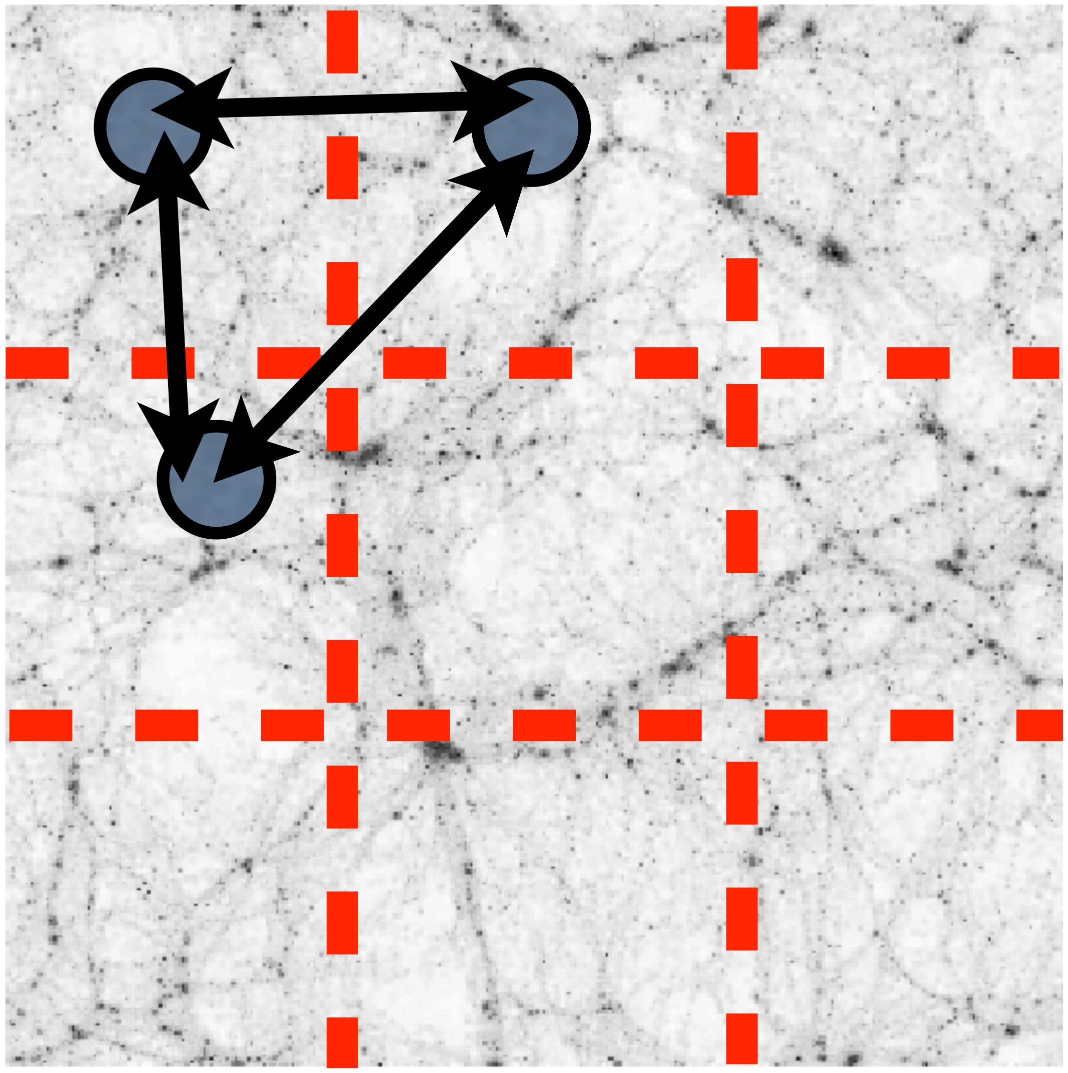


Figure 8: An illustration of our new efficient jackknife resampling algorithm.

Algorithm 3.5.3 TestPointTuple (points $p_1, \ldots, p_n$, matcher $\mathbf{r}$)

```
    mark all permutations $\sigma$ of $[1, \ldots, n]$ as valid
    for all indices $i, j \in [1, n], i < j$ do
      $d_{ij} = \|p_i - p_j\|$
      for all permutations $\sigma$ do
5:      if $\sigma$ is marked valid then
          if $r^{(l)}_{\sigma(i)\sigma(j)} > d_{ij}$ or $r^{(u)}_{\sigma(i)\sigma(j)} < d_{ij}$ then
            mark $\sigma$ as invalid
          end if
        end if
10:   end for
    end for
    if At least one permutation $\sigma$ is marked valid then
      return true
    else
15:   return false
    end if
```

with our optimized base case implementation (discussed below).

3.5.2 Multi-Matcher Algorithm

The algorithms presented above all focus on computing individual counts of points – *i.e.* Computational Task 1, from Section 3.3.2. 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 a two-point correlation computation. Let A and B be a pair of nodes separated by a distance that is large compared to any distances in the matcher. If we compute the raw correlation counts for multiple matchers by iterating over matchers and using Algorithm 3.5.1, we will eventually consider nodes A and B. Since the minimum distance between them is large when compared to the matcher, we will prune this pair. On the next iteration, for a new matcher, we will again compute

Algorithm 3.5.4 TestNodeTuple (Tree node $T_1, \ldots,$ Tree node T_n, matcher $\mathbf{r}$)

```
    mark all permutations $\sigma$ of $[1, \ldots, n]$ as valid
    **for all** indices $i \in [1, n-1]$ and indices $j > i$ **do**
      $d_{ij}^{\max} = d^{\max}(T_i, T_j)$
      $d_{ij}^{\min} = d^{\min}(T_i, T_j)$
 5:   **for all** permutations $\sigma$ of $[1, \ldots, n]$ **do**
        **if** $\sigma$ is marked valid **then**
          **if** $r_{\sigma(i)\sigma(j)}^{(l)} > d_{ij}^{\max}$ or $r_{\sigma(i)\sigma(j)}^{(u)} < d_{ij}^{\min}$ **then**
            mark $\sigma$ as invalid
          **end if**
10:     **end if**
      **end for**
    **end for**
    **if** At least one permutation $\sigma$ is marked valid **then**
      **return** true
15: **else**
      **return** false
    **end if**
```

the minimum distance between A and B, and since it is still large, we will prune again. We can 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, the pair satisfies.

In the two-point case, this improvement is straightforward, and has already been shown in [74]. We must only modify Alg. 3.5.3 and Alg. 3.5.4 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.

The general n-point case requires more caution. The presence of permutations in Alg. 3.5.4 makes the straightforward approach mentioned above more difficult. Each pair of nodes in each permutation will possibly satisfy different matchers. It becomes prohibitively expensive to determine which, if any, matchers may be satisfied.

Algorithm 3.5.5 InteractionListNpt (Tree root T, matcher $\mathbf{r}$)

```
    FormInteractionListsDualTree($T, T$)
    for all leaves $T_l$ of $T$ do
      for all unique $(n-1)$-tuples of nodes $T'_1, \ldots, T'_{n-1}$ in $L(T_l)$ do
        if TestNodeTuple($T_l, T'_1 \ldots, T'_{n-1}, \mathbf{r}$) then
5:        for all unique $n$-tuples of points $p_1 \in T_l, \ldots,$ points $p_n \in T'_{n-1}$ do
            if TestPointTuple($p_1, \ldots, p_n, \mathbf{r}$) then
              result++
            end if
          end for
10:     end if
      end for
    end for
```

Algorithm 3.5.6 FormInteractionListsDualTree
(Tree nodes T_1, T_2, distance $\hat{r}$)

```
    if $T_1$ and $T_2$ are leaves then
      Add $T_2$ to $T_1$'s list (and vice versa)
    else if $d^{\min}(T_1, T_2) > \hat{r}$ then
      Prune
5:  else
      Split the larger node and recurse
    end if
```

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 3.5.1. *Given a matcher* $\mathbf{r}$*, let* $\left\{u_i : 1 \leq i \leq \binom{n}{2}\right\}$ *be the set of upper bounds* $r_{ij}^{(u)}$*, in the matcher, sorted in ascending order. Let* $\left\{d_i^l : 1 \leq i \leq \binom{n}{2}\right\}$ *be the set of lower-bound distances* d_{ij}^{min} *obtained from the bounding boxes of* n *tree nodes, also sorted in ascending order. Then, if* $d_i^l > u_i$ *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_i^l to some upper bound distance in the matcher $u_{\sigma(i)}$. In order for this permutation to work, we need that $d_i^l < u_{\sigma(i)}$ for all i. Let i' be the index such that $d_{i'}^l > u_{i'}$ 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. $\square$

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

Corollary 3.5.2. *Let $\left\{l_i : 1 \leq i \leq \binom{n}{2}\right\}$ be the set of lower bounds in the matcher, sorted in ascending order. Let $\left\{d^u_i : 1 \leq i \leq \binom{n}{2}\right\}$ be the set of upper-bound distances obtained from the bounding boxes of n tree nodes, also sorted in ascending order. Then, if $d^u_i < l_i$ for any i, no n-tuple of points from the nodes can satisfy the matcher.*

Using these observations, we can improve the pruning rule in Alg. 3.5.4. With each n-tuple of nodes, we store the upper and lower bound distances, sorted in ascending order. Given a collection of matchers, we can extract the largest upper bound and smallest lower bound among all matchers for each of the $\binom{n}{2}$ entries. We can then sort these distances and use the theorems above.

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.

We can then provide a multi-matcher version of Alg. 3.5.4. 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. 3.5.3 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 (or bins) it contains and mark them as possibly valid. 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.

Algorithm 3.5.7 MultiTestNodeTuple (Tree node $T_1, \dots,$ 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 $i = 1 : \binom{n}{2}$ do
    if $l_i > r_i^{\max}$ or $u_i < r_i^{\min}$ then
      return false
5:  end if
  end for
  return true
```

So far, we have discussed multiple matchers in a completely generic way, without discussing any structural relationships that we may be able to exploit for our algorithms.

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

Another widely used matcher specification for 3-point correlations fixes two sides of a triangle and varies the angle between them. This angle matcher specification is widely used to study configuration dependence in the 3pcf [131, 117, 118]. We have implemented specialized base cases and pruning rules for this case. The user can input one or more sets of lengths for a triangle, along with a range of angles to be considered. Using a simplified version of our general multi-matcher pruning algorithm, we can efficiently compute results for all of these matchers.

Our specifications 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.

3.5.3 Efficient Resampling Algorithm

We now turn to another new algorithm – a method to efficiently compute counts for resampling regions (Computational Task 3). Recall from Section 3.3.2 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.

The intuition for our new approach is shown in Fig. 8. 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.

We can avoid this extra work by working with the subsets D_i directly. For instance, if we compute the count $D_i D_i D_i(\mathbf{r})$, 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. 3.5.8. We maintain an array of results of length J, where the jth entry corresponds to the count $D_{(-j)} \cdots D_{(-j)}(\mathbf{r})$. This method can be easily

Algorithm 3.5.8 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 $(i_1, \ldots, i_n)$ in $[1, \ldots, J]$ do
      result = MultiTreeNpt($T_{i_1}, \ldots, T_{i_n}, \mathbf{r}$)
      for all $j \notin \{i_i, \ldots, i_n\}$ do
5:      results[j] += result
      end for
    end for
```

extended to include random sets by including the random set as one of the D_i in the input to Alg. 3.5.8.

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.

This algorithm can easily be combined with our multi-matcher algorithm above.

3.6 Base Case Optimizations

We focus here on the base case computation of the *Raw Correlation Counts* (defined in Section 3.3.2). We restrict our attention to counts for the 3-point correlation for simplicity. We have three sets of points A, B, C. Our task is to identify all triples of points (a, b, c) where $a \in A$, $b \in B$, $c \in C$ and the triple satisfies a given matcher. A brute force algorithm for the base case calculation would simply iterate through all triples using a triply-nested loop and test each triple against the matcher. Assuming the sets of points have cardinality N, such an approach involves $O(N^3)$ distance calculations and $O(N^3)$ conditional evaluations to determine if the matcher is satisfied.

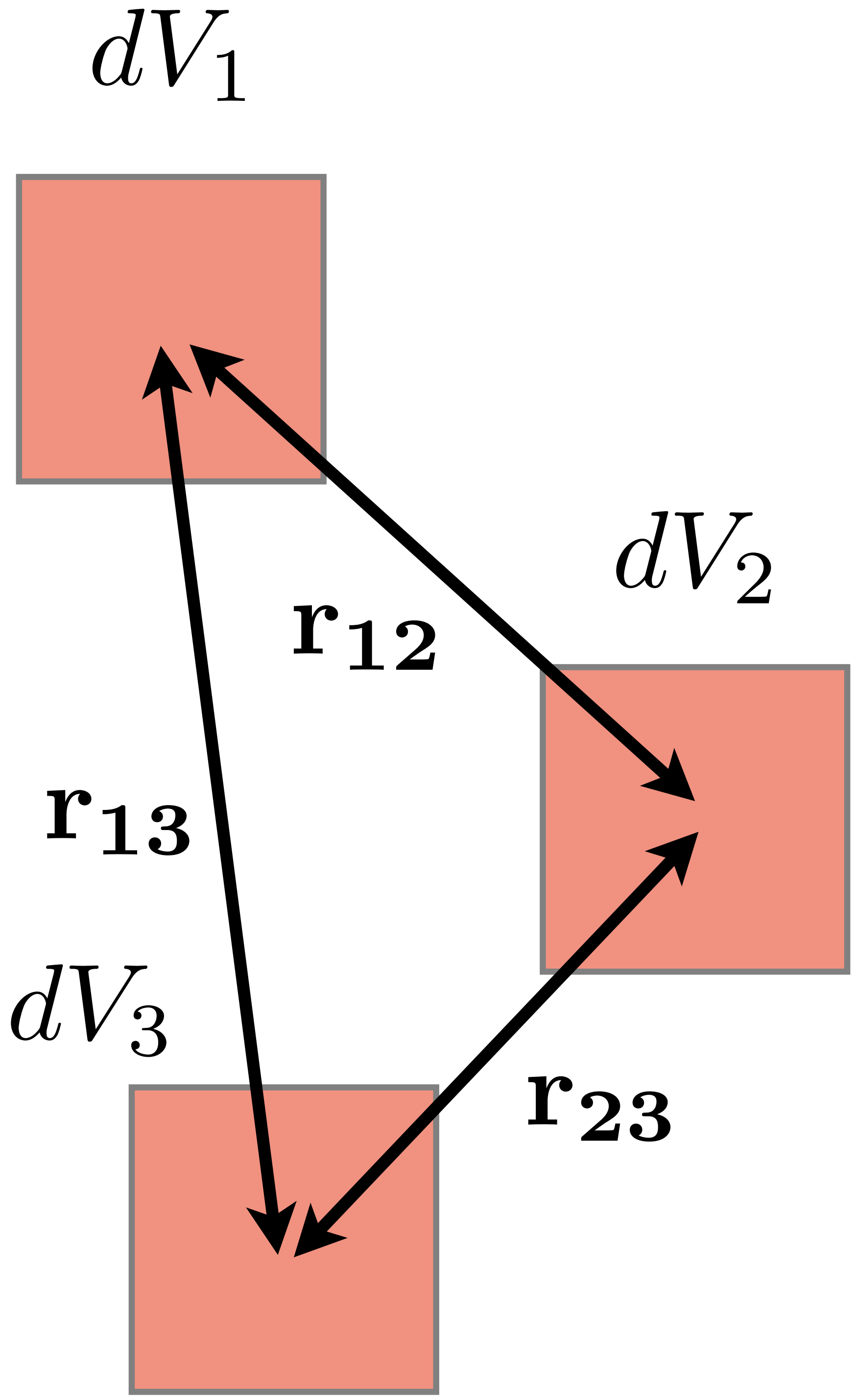


Figure 9: An illustration of the counts required for the three-point correlation function.

The brute-force approach computes many redundant distances. We use a two-phase base case calculation that first computes all of the unique pairwise distances and stores the result of whether or not the distance satisfied each of the matcher constraints. Note that the distances themselves need not be stored. We then iterate through all triples and check the stored set of matcher results to determine if the matcher has been satisfied. Thus, this algorithm involves $O(N^2)$ distance calculations, $O(N^2)$ conditional checks, and $O(N^3)$ matcher satisfiability checks. We demonstrate in the following section that the $O(N^3)$ matcher satisfiability checks can be performed without the use of conditionals.

3.6.1 Bitwise Interpretation of ***Raw Correlation Counts***

We now turn to an approach to count the number of triples satisfying a matcher using only bitwise logical operations and population counts. Let r_{12}^{AB} represent a K-width set of bits corresponding to pairs of points from sets A and B and indicating whether or not the matcher distance r_{12} was satisfied by each pair of points. In other words, the i^{th} bit of r_{12}^{AB} is one if the distance between the i^{th} pair is between $r_{12}^{(l)}$ and $r_{12}^{(u)}$. For three sets of points A, B, and C, there are nine such K-width values to represent the three different matcher thresholds and three different pairings of sets of points. We can then increment the total matcher count for a set of K triples by

$$\text{POPCNT}\left(\begin{array}{c} (r_{12}^{AB} \wedge r_{23}^{AC} \wedge r_{13}^{BC}) \vee (r_{12}^{AB} \wedge r_{13}^{AC} \wedge r_{23}^{BC}) \vee (r_{23}^{AB} \wedge r_{12}^{AC} \wedge r_{13}^{BC}) \vee \\ (r_{23}^{AB} \wedge r_{13}^{AC} \wedge r_{12}^{BC}) \vee (r_{13}^{AB} \wedge r_{12}^{AC} \wedge r_{23}^{BC}) \vee (r_{13}^{AB} \wedge r_{23}^{AC} \wedge r_{12}^{BC}) \end{array}\right) \tag{3.6.1}$$

where POPCNT indicates the population count (i.e., the number of bits set to one).

Each of the six parenthesized bit-field conjunctions above represents one of the possible assignments of pairwise distances to edges of the triangle given by the matcher. The conjunction of terms will be one if the matcher is fully satisfied by that arrangement of edges and distances and the disjunction of all six arrangements will thus be one if any such arrangement satisfies the matcher. Therefore, counting the set bits in the resulting K-width bit vector will count the number of triples that satisfy the matcher.

For many K-width vectors, the above computation involves only logical operations, population counts, and integer additions to accumulate the total count. No distances are calculated and no conditionals are required in the $O(N^3)$ portion of the two-phase algorithm. This significantly increases the performance of the part of the kernel that dominates the asymptotic computational complexity.

3.6.2 Base Case Merging

The interaction list tree traversal algorithm (Alg. 3.5.5) provides opportunities for further optimizations. We consider the interaction list for a leaf node A and a node in the list B. We then merge together all remaining nodes in the interaction list for A into a single "meta-node" C. Thus, we generate a set of base cases with node C as large as possible to exploit the efficient bit-field-based kernels.

As we show in Section 3.8.1.2, optimal runtimes for the entire algorithm come from leaf nodes with 10-20 points. The resulting base cases will not use all the bits of even a 32-bit register. By combining nodes into one large node, we can make use of an entire word and test many potential tuples in parallel.

3.6.3 Architecture-Specific Optimizations

While the bitwise interpretation of *Raw Correlation Counts* described in Section 3.6.1 is already an efficient representation for most platforms, there are several architectural features that will significantly impact ultimate performance. In particular, the optimal width K of the triples to be considered simultaneously will depend upon the register widths and supported instruction sets. In addition, population count instructions (i.e., a single instruction that returns the number of set bits in some bit-field) are particularly useful. Fortunately, such instructions are available on most modern CPUs and GPUs.

In this section, we consider *Raw Correlation Counts* on several modern CPUs, including Intel Nehalem and Sandy Bridge processors and AMD K10 and Bulldozer processors. We also discuss optimizations for low-power systems including Intel Atom and ARM Cortex-A9 processors.

As noted above, we split the base case computation into two phases: the calculation of all pairwise distances between points and the population counts based on the logical expression in Eqn. 3.6.1. These two phases, one consisting of floating-point instructions and the other logical operations and population counts, are largely independent and can be optimized separately. To achieve optimal performance we have implemented several kernels for each phase and selected the fastest by running on actual hardware.

3.6.3.1 Floating Point Phase

For phase one of the base case calculation, we have implemented kernels using scalar SSE2 instructions, 128-bit vector SSE2, SSE3, and FMA4 instructions, and 256-bit AVX instructions. With two exceptions, using the highest available instruction set delivered optimal performance. The exceptions are Intel Atom and AMD Bulldozer processors. Although Atom supports the SSE3 instruction set and Bulldozer supports the AVX instruction set, both handle the widest vector cases by either issuing multiple narrower microoperations or fusing narrower vector resources.

The ARM architecture does not support double-precision SIMD instructions, so double-precision computations are performed on a standard VFPv3 floating-point unit. However, the ARMv7-A ISA uses the floating point status register for double precision comparison results rather than writing a bitmask to a floating point register. In order to avoid transfers of comparison results to general purpose registers, we exploit the fact that the NEON SIMD unit uses the same registers as the floating-point unit and construct a comparison bitmask using available NEON instructions.

Table 2: Processors used in the experiments shown in Fig. 10.

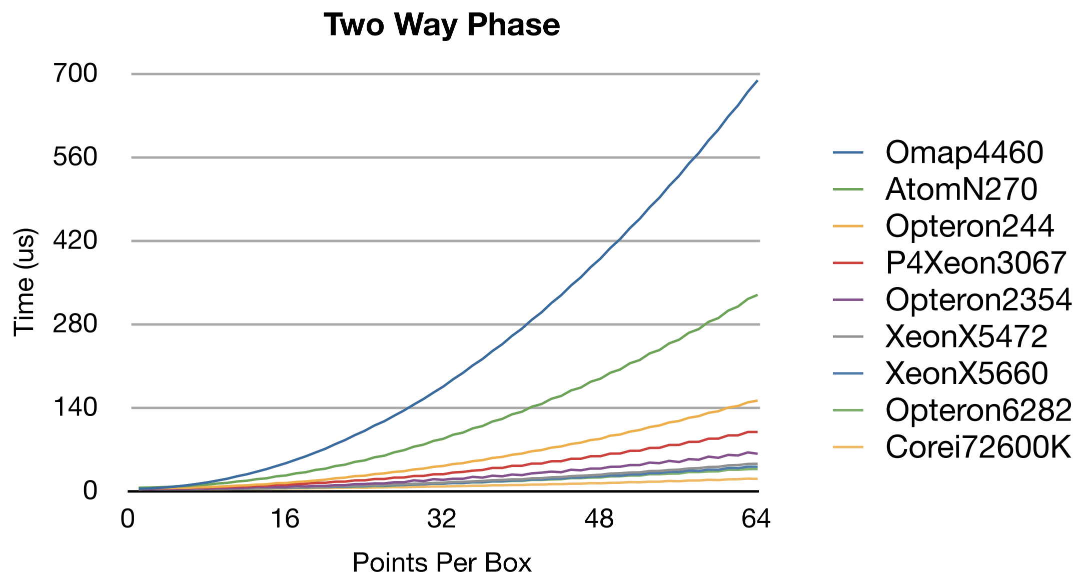
Processor	Clock	ISA	Microarchitecture
OMAP 4460	1.2 GHz	ARMv7	Cortex-A9
Atom N270	1.6 GHz	x86	Bonnell
Opteron 244	1.8 GHz	x86-64	K8
P4 Xeon 3067	3.1 GHz	x86	NetBurst
Opteron 2354	2.2 GHz	x86-64	K10
Xeon X5472	3.0 GHz	x86-64	Harpertown
Opteron 6282	2.6 GHz	x86-64	Bulldozer
Xeon X5660	2.8 GHz	x86-64	Nehalem
Core i7 2600K	3.4 GHz	x86-64	Sandy Bridge

NEON is a widespread SIMD extension to the ARMv7 ISA. This approach is faster than the naïve approach of transfers from the status registers to general-purpose registers. Figure 10 illustrates the performance of the floating-point intensive phase of the base case calculation on the CPUs listed in Table 2.

3.6.3.2 Logical operations phase

For the second phase of the base case computation, we leverage 64-bit instructions available on modern CPUs and vector representations of the bit-fields to achieve optimal performance. We restrict the sizes of sets A, B, and C to be at most 64. This restriction is acceptable for our tree-based implementation because very few leaf nodes have more than 64 points in an optimal tree (see Section 3.8.1.2). We handle base cases with nodes containing more than 64 points by splitting the nodes and making multiple smaller kernel calls. Thus, we can reduce the phase two inner loop to a small number of logical operations on 64-bit binary vectors and a final population count computation on a 64-bit binary vector. It is worth noting that 64-bit instructions are available on all modern x86 CPUs.

Since logical operations are fast on all CPUs, relative performance of the kernel is determined by the speed of the population count computation. Intel Nehalem and Sandy Bridge and AMD K10 and Bulldozer CPUs all have hardware support for this



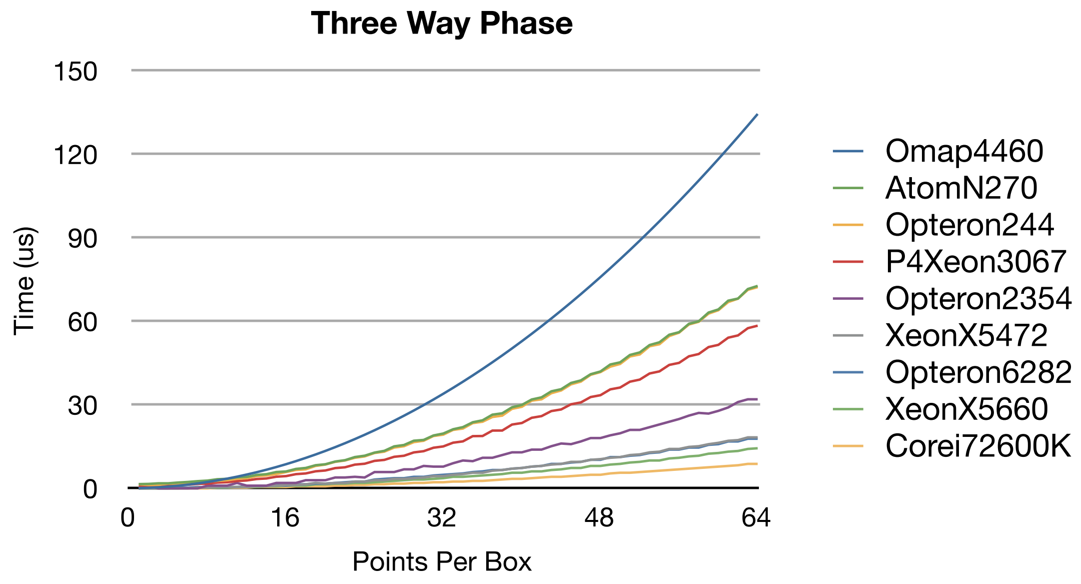


Figure 10: Execution times for the first phase (two way) and second phase (three way) of the base case calculations. Reported times correspond to the minimum time over 100,000 kernel executions. All kernels were compiled with gcc 4.6.3 using architecture-specific optimization flags.

operation via a POPCNT instruction. We found that on all of these CPUs using POPCNT instructions results in optimal performance or near optimal performance. For older CPUs we tried several methods of computing population counts. First, we considered using 8-bit, 11-bit and 16-bit lookup tables. We also tried a simple divide-and-conquer implementation, its more sophisticated variant in Alg. 5.1 from [173], the algorithm from HAKMEM [15], and its variant with 4-bit fields from [173]. For the last algorithm, we also designed SIMD versions using MMX and SSE2 instructions. Additionally, we implemented SSSE3 and XOP versions of the SIMD-oriented population count algorithm suggested by W. Mula[122].

The basic ARMv7 instruction set does not contain a population count instruction, but NEON provides a SIMD instruction VCNT that performs population counts on SIMD vectors. Cortex-A9 CPUs have only 64-bit execution units, so using 128-bit instructions does not yield performance benefits. Moreover, Cortex-A9 can issue only one NEON SIMD instruction per clock cycle compared to two scalar operations. Despite these limitations, in our tests the NEON version is about 50% faster than our fastest scalar version. Figure 10 illustrates the performance of phase two of the base case computation on various CPUs.

Additionally, we can see further performance benefits by short-circuiting the loop over the third element of the tuple. When considering a tuple (a, b, c), if we find that the distance between a and b will not satisfy any constraint in the matcher, then we can stop testing possible values of c.

3.7 Implementations

We now discuss implementations of our new n-point correlation function estimation algorithms.

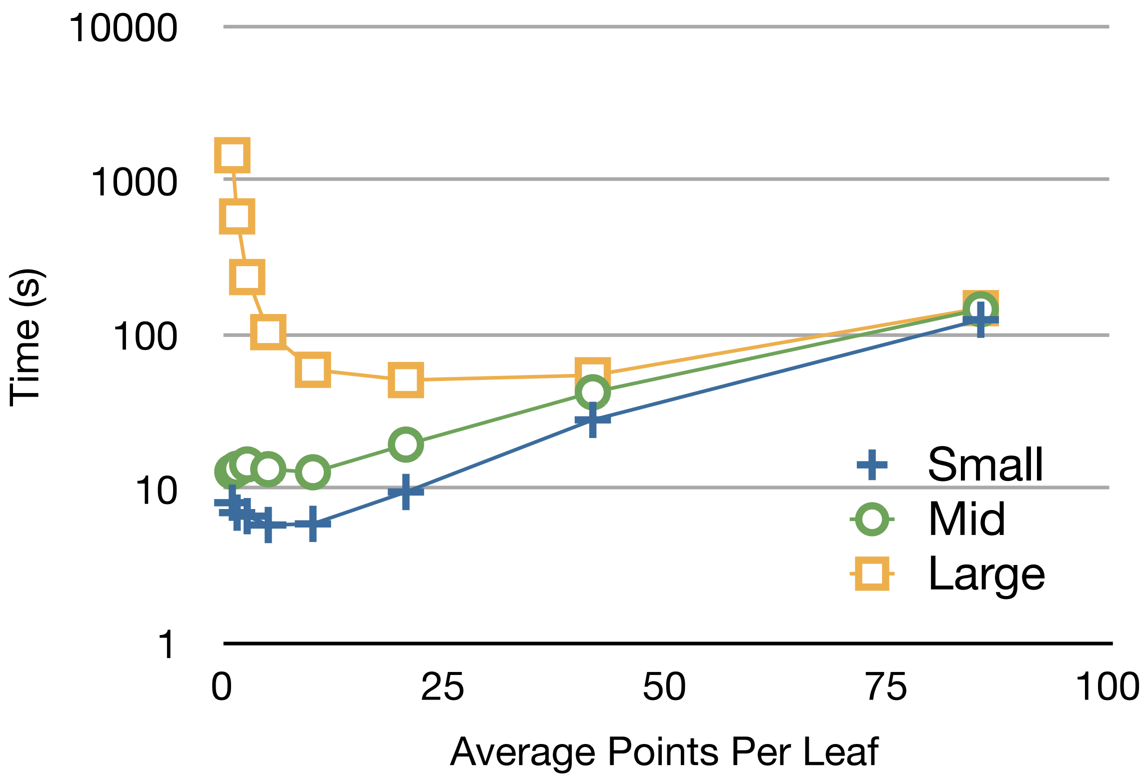


Figure 11: Runtime vs. average number of points per leaf node in the optimized kd-tree algorithm. The data are 1.1×10^6 points from an N-body simulation of galaxy positions in a cube with side length 1000 Mpc h^{-1}. The three series correspond to a matcher with side lengths $3, 6, 3.1$ (small), $10, 10, 10$ (Mid), and $9, 27, 35$ (Large), with all measurements in units of Mpc h^{-1}.

3.7.1 Parallel Implementation

The naive npcf estimation algorithm can be trivially parallelized. Each n-tuple of points can be checked independently using Algorithm 3.5.3. Although this method is embarrassingly parallel, it suffers from performing mostly unnecessary work. We therefore turn to parallelizing our more efficient algorithms.

In general, different parts of the multi-tree traversal are independent. Therefore, we can perform these computations in parallel. In other words, the two recursive calls in Algorithm 3.5.2 can be done independently. In a shared memory setting, this can be done by allocating space for the results of subtasks and combining them when the computation finishes.

In the pairwise traversal (Algorithm 3.5.5), the formation of interaction lists can be parallelized as described above. The loop over leaves can be done in parallel, again by allocating separate space for the results for each thread and reducing over these partial lists after the loop completes.

The above ideas can be carried to the distributed setting as well. However, we can make further use of the observation that underlies all our fast algorithms: a point will only belong to a tuple that satisfies the matcher with points that are "close" to it.

Parallelizing tree traversals across distributed compute nodes may require us to send large blocks of data during the computation. Instead, for p processors, we can simply divide the space into p equal-sized regions and allocate the points in each region to the corresponding processor. We then "ghost" the points along the boundaries.

Let $\hat{d}$ be the maximum upper bound distance among all entries in all matchers. Then, in order to count all matching tuples that include points it owns, each process only needs points that lie within $\hat{d}$ of its region. Therefore, each process needs ghost copies of all points in other processors' regions satisfying this distance constraint. Process i sends copies of its points that lie within distance $\hat{d}$ of its boundaries to all processes j where $j < i$. The last restriction is necessary to avoid overcounting.

Once each process has it's ghost points, it can perform its computations independently of the other processors. Note that we must avoid counting any tuples consisting entirely of ghost points, since these will be handled by the process which owns the points.

We simply assume that the points are loaded into some arbitrary subset of the nodes. The processes also need to know the dimensions of the overall input state, the number of other processes, and the partitioning of space among processes, all of which are easy for nodes to compute for themselves or a master node to broadcast. The nodes then perform a distributed shuffle so that each node owns all the points in its region. Processors send ghosts to their neighbors as described above. Each process performs its own counts, using the multi-tree algorithms described above. The final result is then collected using a distributed reduce.

3.7.2 Open-Source Code

We have implemented all of the above ideas in an open-source library, NPOINT. The code is implemented in C++. It uses space-partitioning trees from the MLPACK open-source machine learning library [39].

The code consists of an MPI layer which handles the parallel computation described above. The efficient resampling code knows how to assign owned and ghost points to resampling regions and handles the calculations over individual regions and combines them into the final counts. It calls the multi-tree algorithm (single or multi-matcher as needed). As noted above, the base cases and pruning checks are contained within matcher classes, which are selected using templates. The CPU optimizations are contained within specialized matcher classes. All of these options can be selected using command line parameters.

The library currently supports *kd*-trees, with octrees being replaced after a major refactoring. It supports single matchers for general values of n, multi-matchers using evenly spaced bins for general n, and angle matchers for 3pcf calculations. It also includes CPU optimized calculations for 3 and 4 point, both single and multi-matcher. Additional methods and options will be added based on demand from users.

3.7.3 Specifying Multiple Matchers

Previously, we discussed algorithms to efficiently compute raw correlation counts for multiple matchers by sharing work between computations over different matchers. We now turn to the question of how the user will specify these matchers.

A single matcher is simple to specify. The user inputs a matrix (as in Equation 3.3.1) and thickness dr (or an upper and lower bound matrix). Multiple matchers can take a variety of forms.

The most straightforward method for multiple matchers is to simply input a list of matrices. We can apply the multi-matcher algorithms above by simply finding the minimum lower bound and maximum upper bound in each entry of the matcher over all matchers. However, the base case is less straightforward. For M matchers, we can iterate through all of them when checking a pairwise distance. In general, we cannot do better than this, since a given n-tuple of points may satisfy many (or all) of the matchers.

It is possible to obtain additional performance improvements by restricting the class of matchers available. One possibility is to specify a set of matchers by giving an upper and lower bound in each dimension along with a number of bins to divide the dimension into. In this setting, we can make the prune checks as before. However, in the base case computations,

Implementations are governed by what practitioners really want. Currently, the code includes the fixed bins implementation above. It also has a specialized implementation for 3-point correlations based on angles, described in Section 3.5.2. Three point correlations in astronomy are often studied by fixing the lengths of two sides of a triangle and varying the angle between them [117]. The library includes an implementation of a specialized “angle matcher” which shares work between these different matchers.

In addition to the existing implementations, it is easy to add new methods for specifying multi-matchers to the current code base. Currently, the pruning checks (Algorithms 3.5.4 and 3.5.7) and base case (Algorithm 3.5.3) are contained in a matcher class. The tree traversals, jackknife resampling, and distributed parallel codes can all take a matcher class as a template. As long as a new matcher class uses a simple interface, it can be simply inserted into the multi-tree algorithm and run in parallel on distributed or shared memory architectures.

3.8 Results

We now discuss and evaluate our new algorithms and compare against the previously existing approaches on data of real scientific interest. We also present a theoretical runtime analysis of our algorithms.

3.8.1 Empirical Results

We present empirical results on our new algorithms. Many of these results are from two previous papers: KDD 2012 [113] and Supercomputing 2012 [114]. As noted previously, all these experiments used our new C++ implementation based on MLPACK [39].

3.8.1.1 Multi-Matcher and Efficient Resampling

We begin by examining the performance increases from our multi-matcher and efficient resampling algorithms. We compare these to a naive resampling algorithm, which removes subregions one at a time and repeats the entire computation. We also compare to a naive multi-matcher algorithm which loops over each matcher and calls

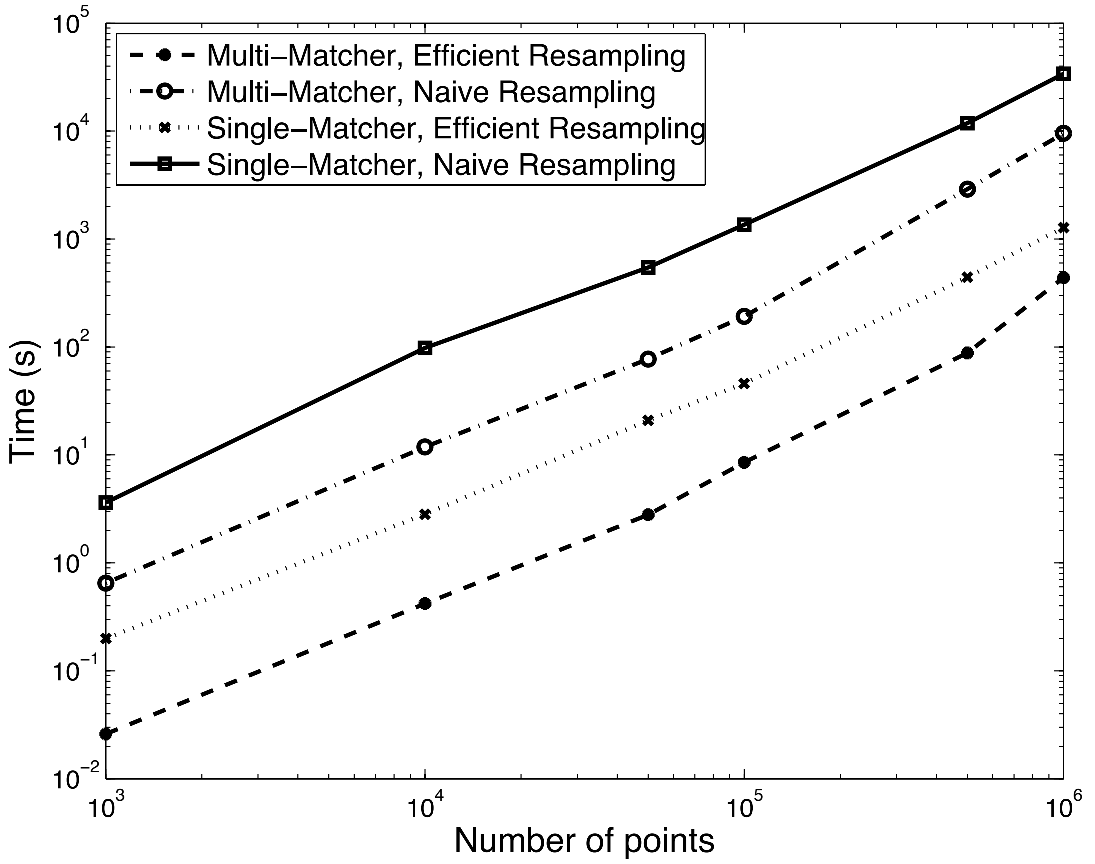


Figure 12: Runtimes for two-point correlation estimation on uniformly distributed data with 20 matchers and 30 equally-sized resampling regions.

Algorithm 3.5.2. These experiments were all performed in serial on a six core, AMD Phenom 3.3 GHz machine.

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

- *single-naive* – The original multi-tree algorithm [74] inside loops over resampling regions and matchers.
- *single-efficient* – The original multi-tree algorithm [74] inside a loop over matchers but using our improved resampling algorithm (Alg. 3.5.8).
- *multi-naive* – Our improved multi-matcher algorithm inside a loop over resampling regions.
- *multi-efficient* – Both new algorithms used together.

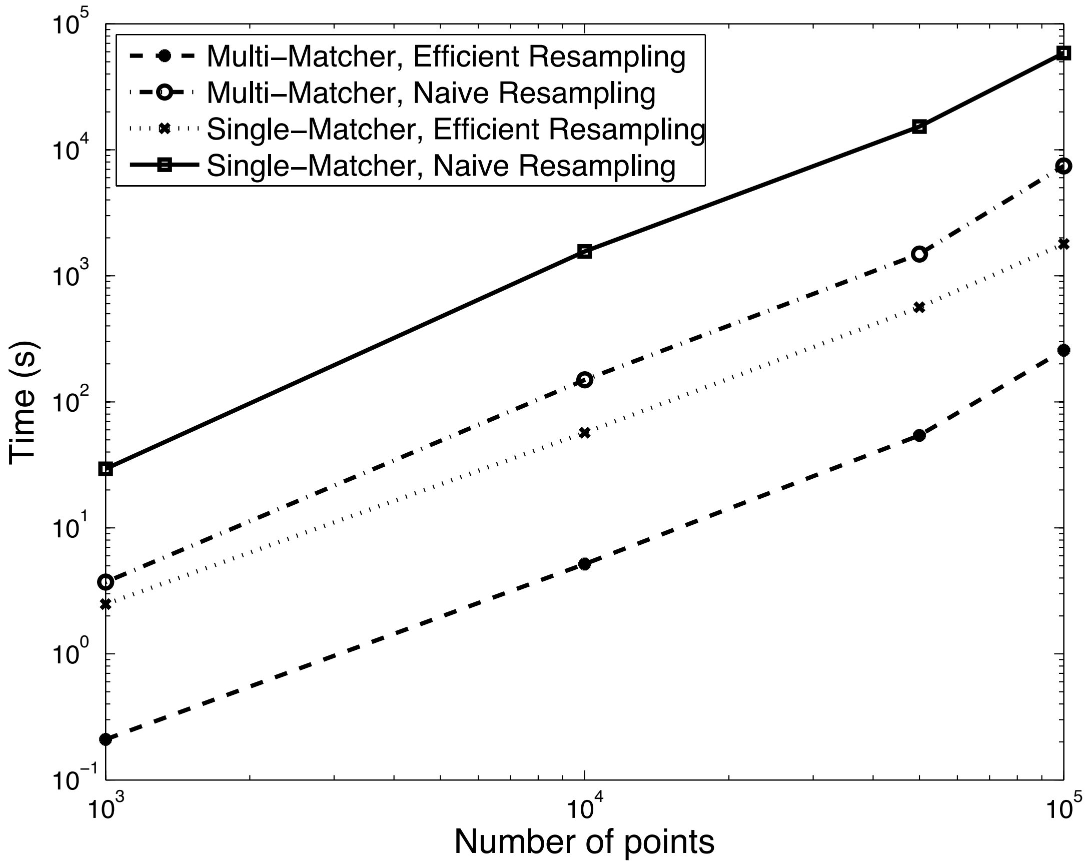


Figure 13: Three-point correlation estimation on uniformly distributed data with 50 matchers and 30 equally-sized resampling regions.

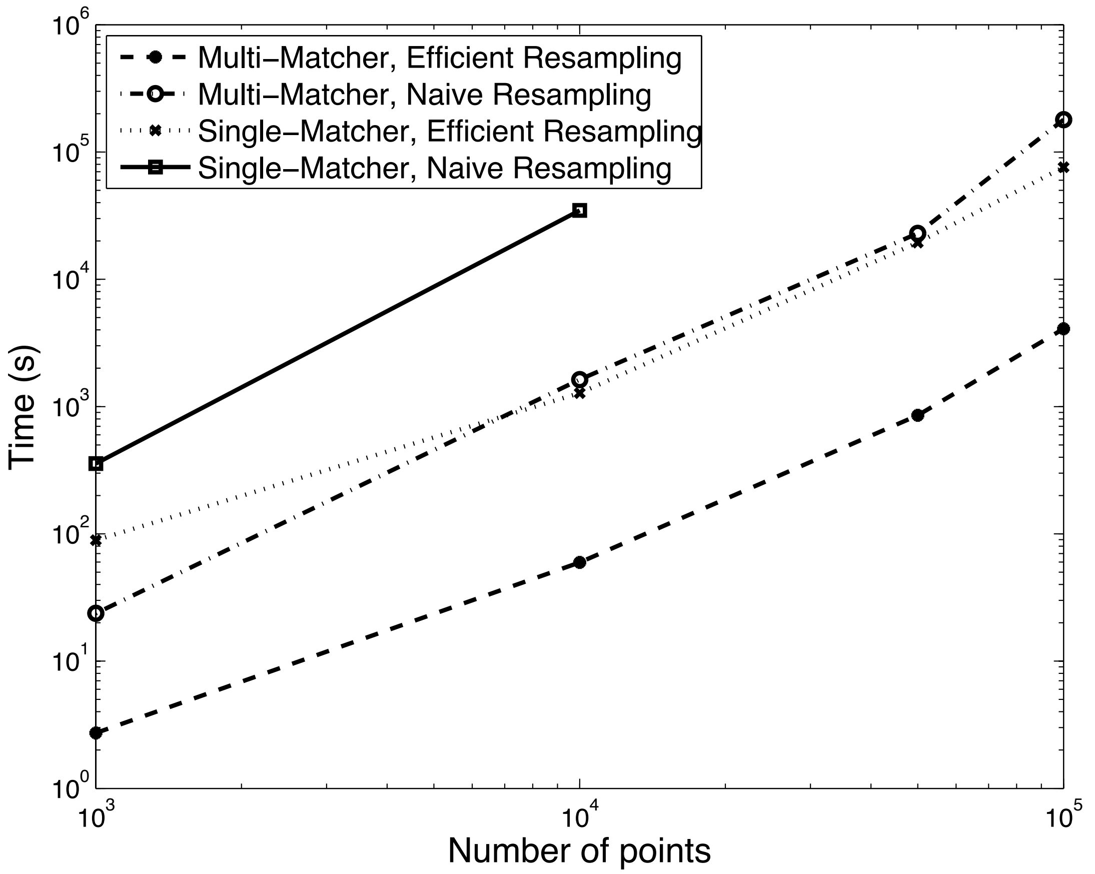


Figure 14: Four-point correlation estimation on uniformly distributed data with 216 matchers and 30 equally-sized resampling regions.

Table 3: Mock catalog experimental results new multi-matcher and efficient resampling algorithms for 2-point correlations. The multi or single refers to the method used to for matchers, naive and efficient refer to the methods used for resampling. All experiments used 10^6 points, 30 equally-sized resampling regions, and 20 matchers.

Algorithm	Time (s)	Speedup (over brute-force)	Speedup (over [74])
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	–

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 3.2.3, represent a significant fraction of the total computational overhead.

In Figures 12, 13, and 14, 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 5. 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. 14). These experiments used more matchers than the others, suggesting that the savings due to the multi-matcher algorithm will increase with more matchers.

Table 4: Mock catalog experimental results new multi-matcher and efficient resampling algorithms for 2-point correlations. The multi or single refers to the method used to for matchers, naive and efficient refer to the methods used for resampling. All experiments used 10^6 points, 12 equally-sized resampling regions, and 18 matchers. Speedups are over the brute-force algorithm. The naive-naive algorithm was omitted because of time constraints.

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 5: Speedups of our new multi-matcher and efficient resampling algorithms over the brute-force algorithm and current state-of-the-art. Timings are from the largest values of N in Figures 12, 13, and 14.

n	Algorithm	Speedup (brute-force)	Speedup (over [74])
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

3.8.1.2 Optimized Kernel

We now turn to some performance results for our optimized solution for the *Raw Correlation Count* problem. We compare the runtimes against an efficient implementation of a tree-based npcf estimation algorithm in the Ntropy framework [65]. This implementation was used for the previously largest 3pcf computation for studying large scale structure in the galaxy distribution, using 106,824 galaxies at 45 scales [119].

We plot runtimes for the galaxy set and a uniformly distributed set of 10^6 points, both of which are contained in a cube of side length 1,000 megaparsecs (Mpc). We specify matchers in terms of three lengths and a tolerance parameter f; the upper

/ lower bounds of the matcher are then $r_{ij} \pm \frac{f}{2} r_{ij}$. We show computations for three matchers with $r = 0.25$: a small matcher with side lengths 3, 6, and 3.09 Mpc, a mid matcher consisting of an equilateral triangle with side length 10 Mpc, and a large matcher with side lengths 9, 18, and 27 Mpc. These parameters represent the smallest and largest matchers used in the previously largest galaxy catalog 3pcf computation mentioned above [119].

Experiments were run on a dual-socket node with two quad-core Intel X5550 2.67 GHz Nehalem processors, 24GB of DDR3 host memory, running Red Hat Enterprise Linux 5.

In Fig. 17, we compare the runtime for our optimized implementation with the Ntropy framework. We show both the tree traversal time and the base case time. Note that our implementation almost exclusively spends time in the base case, while Ntropy tends more heavily toward tree traversal. For both methods, we used the overall optimal tuning parameter settings. For Ntropy, this meant a bucket size of 2 for all cases. In our algorithms, we used *kd*-trees with small leaf sizes for the smaller matchers and octrees for the large one.

In Fig. 11, we show the runtime of our *kd*-tree implementation as a function of the average number of points per leaf on the three matchers mentioned above. Note that the optimal leaf size depends on the size of the matcher. As the scale of the correlation being computed increases, a slightly larger leaf size becomes optimal.

3.8.1.3 Scaling Behavior

We begin by examining the empirical scaling behavior of the multi-tree algorithm with increasing numbers of points. As we discuss further in Section 3.8.2 below, we expect the runtime to depend on both the number of points and density of the distribution the data are drawn from. We examine the runtimes of our algorithm as the number of points increases for both 2-point and 3-point computations. In one set

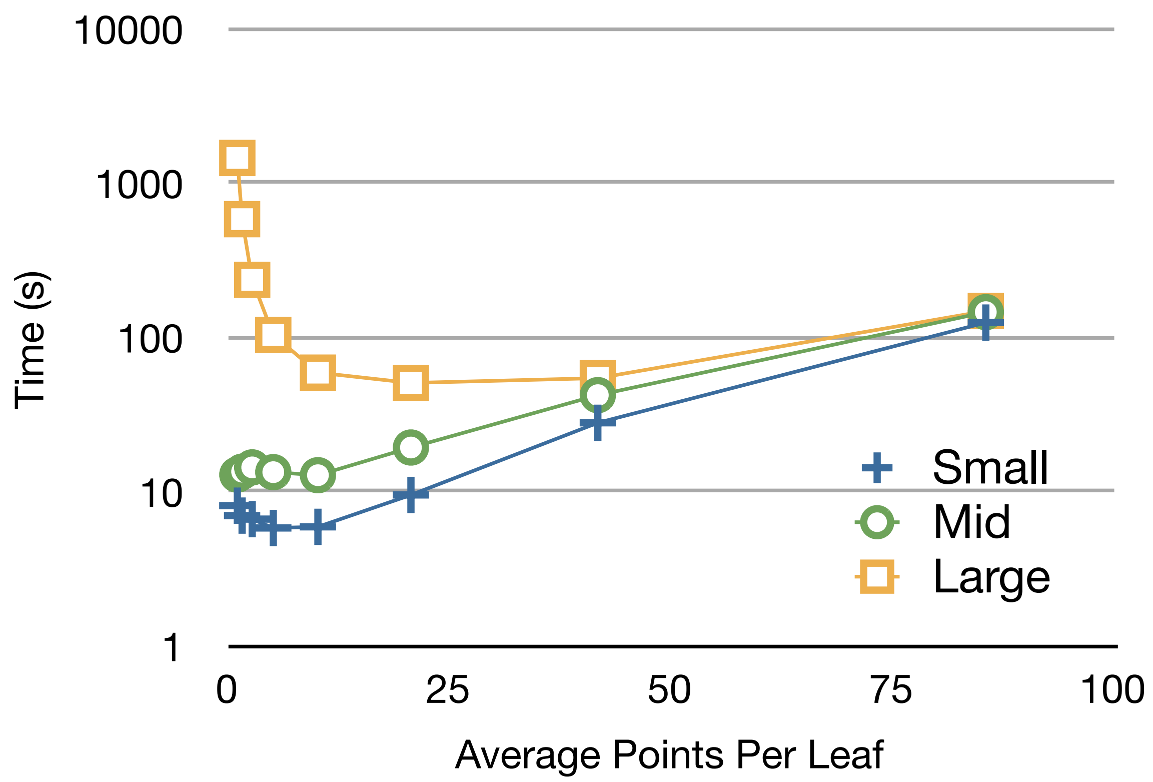


Figure 15: Runtime vs. average number of points per leaf node in our optimized *kd*-tree algorithm. The series show results for the small, mid, and large-sized matchers.

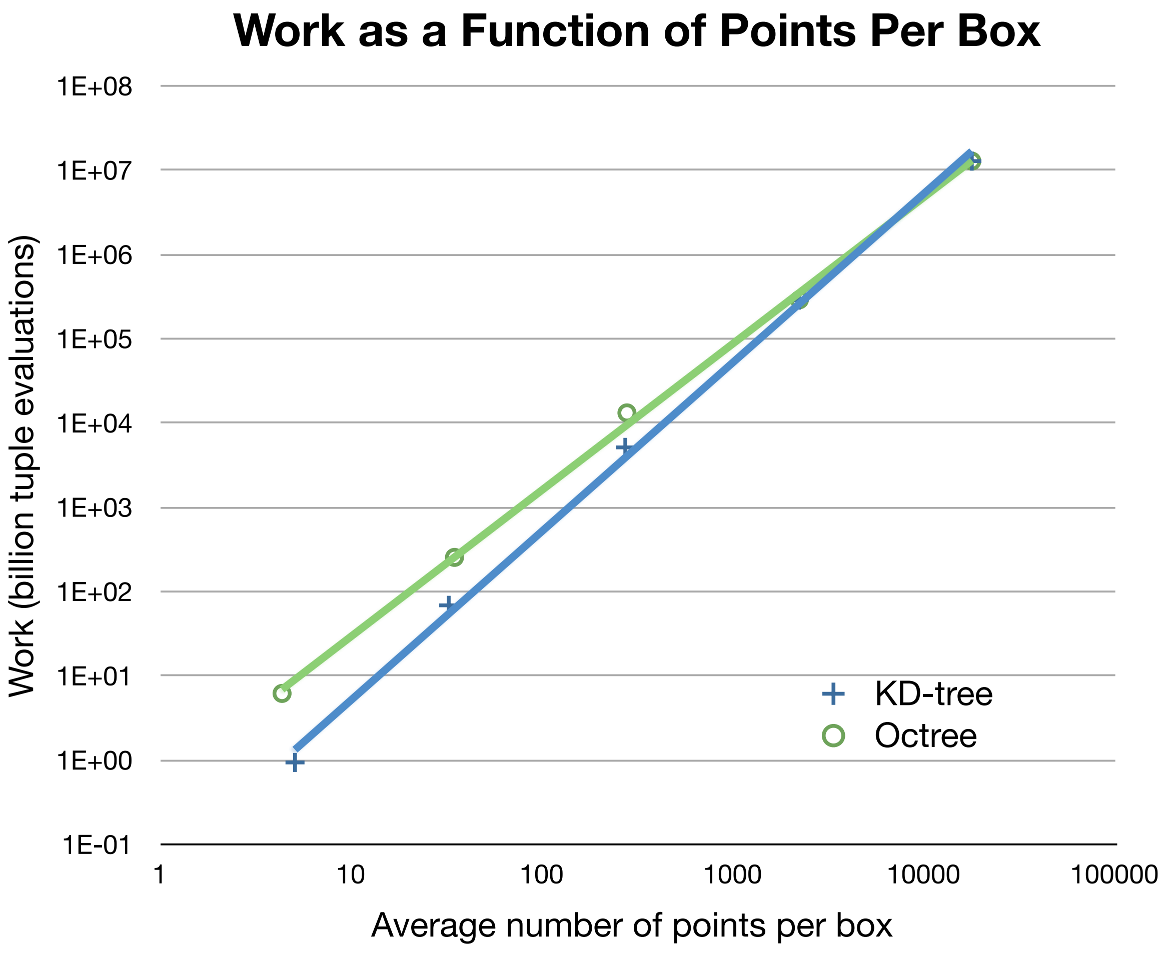


Figure 16: A plot of the relationship between the average number of points per box and the work. The reported values are derived empirically using our galaxy data with mid-sized matchers.

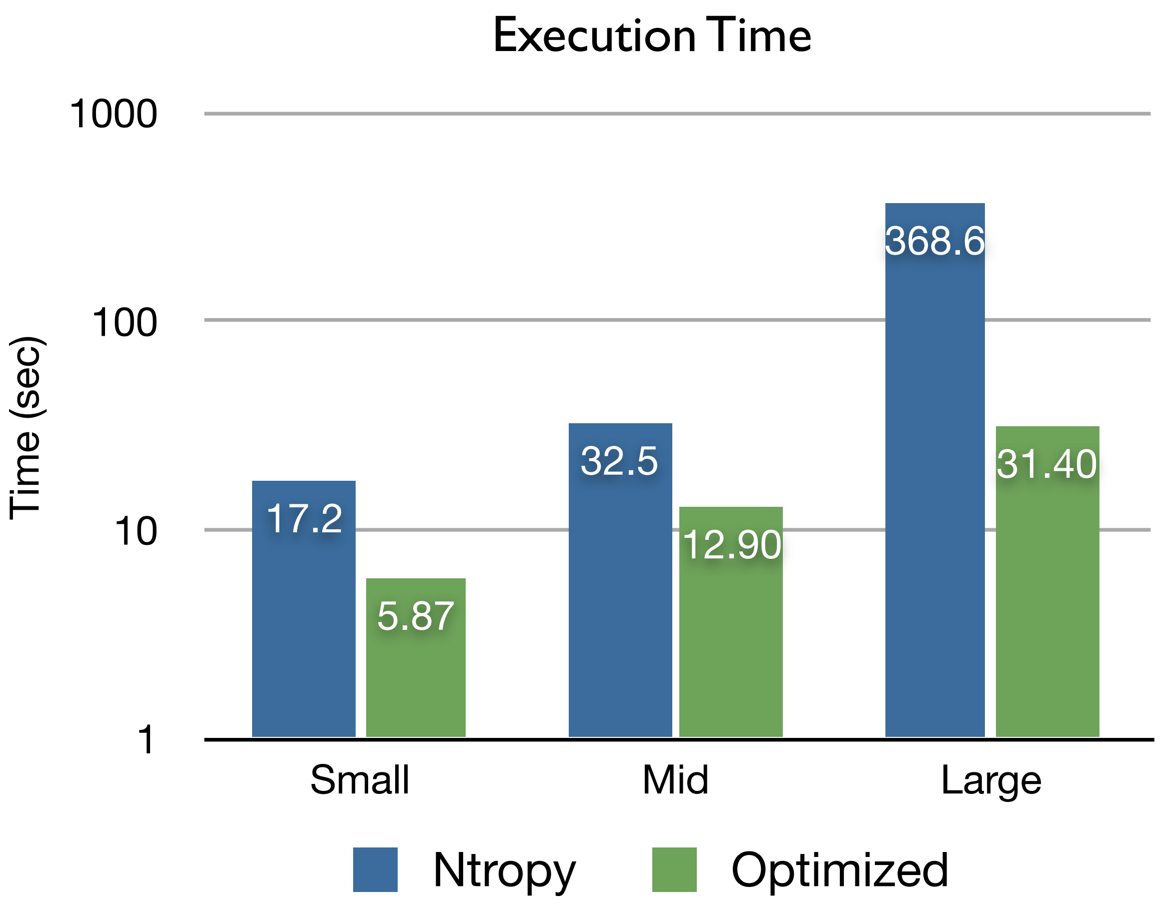


Figure 17: A plot of the execution time of our implementation compared with the Ntropy code. Results include timing values for three different sizes of matchers.

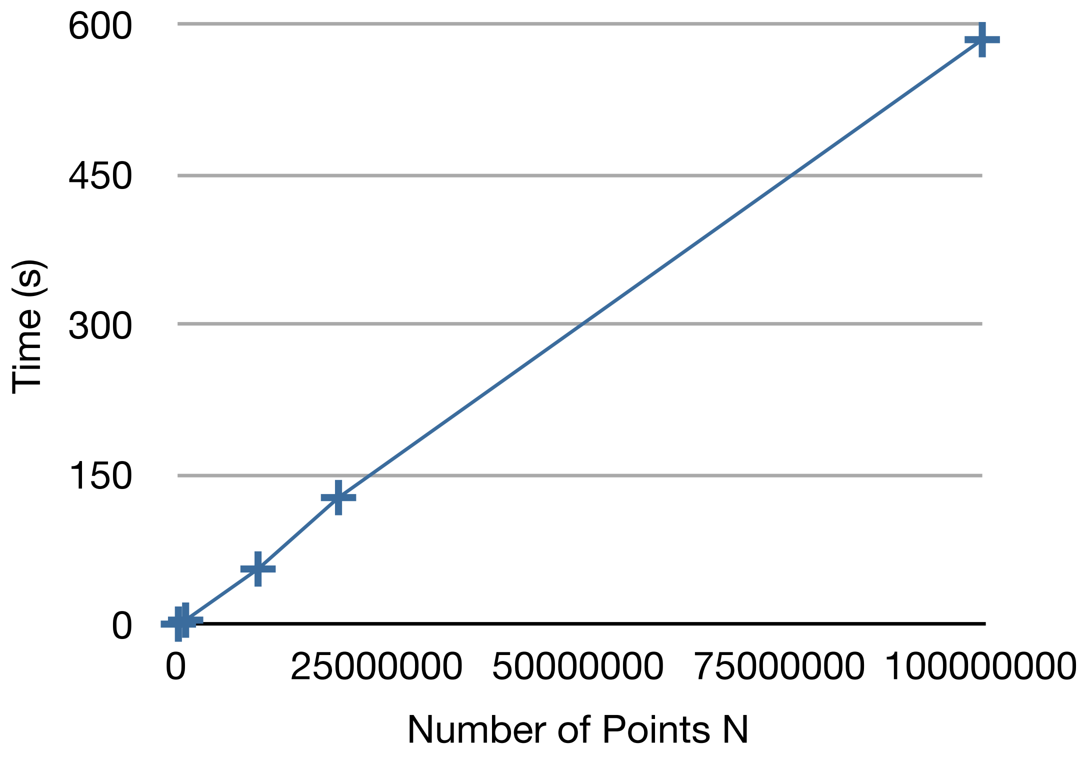


Figure 18: Execution time for 2-point correlation counts on Poisson data with density $2 \times 10^{-2}(h/\mathrm{Mpc})^3$. The data are linear with $R^2 = 0.9996$.

of experiments, we fix the density for all computations, in the other, we fix the box size containing all of the data and increase N, thus increasing the density as N. As shown in Figures 18 and 19, for fixed density, both our 2-point and 3-point algorithms show linear scaling with increasing N.

For the fixed density experiments, we used a matcher with length 8 Mpc/h and a thickness of 25%. The 3pt matcher used an equilateral triangle with all three sides of this length. The data are Poisson in the 2-point case and from a mock galaxy catalog for the 3-point experiments. The different sizes are the points contained in sub-cubes of the 1000 Mpc/h cube containing the whole data set.

We also show results for increasing density. We draw samples from a Poisson distribution with a fixed sample window with sides of length 1000Mpc/h. Therefore,

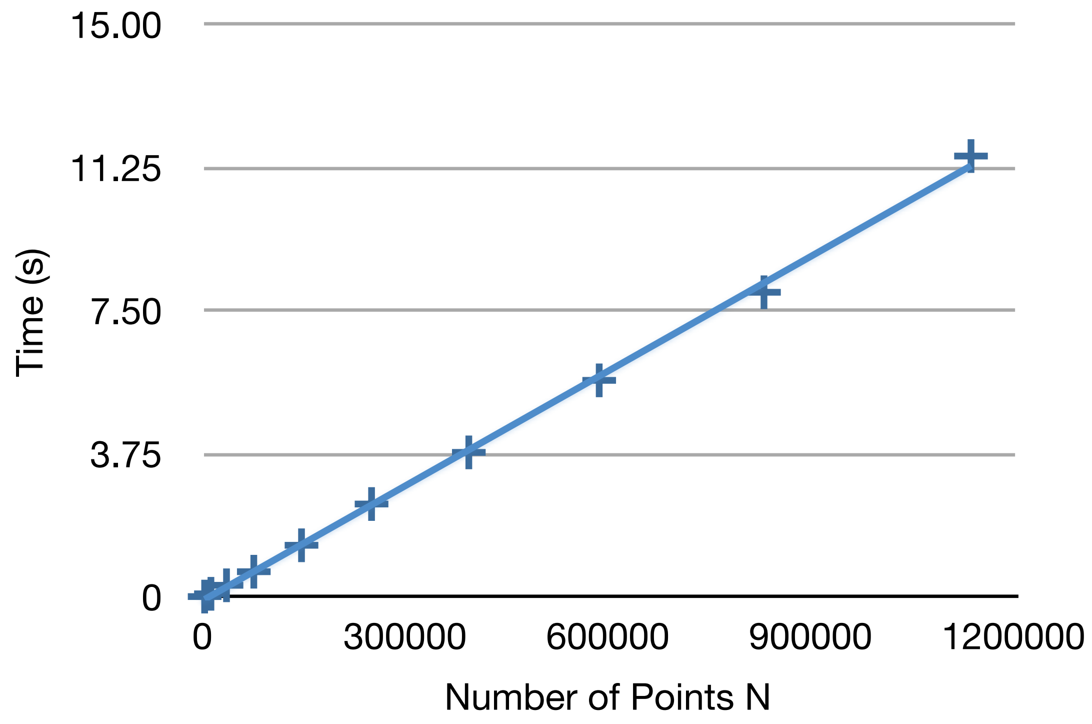


Figure 19: Execution time for 3-point correlation counts on a mock galaxy catalog. The data sets are subregions of a single, large mock galaxy catalog with sample density $1.2 \times 10^{-3}(h/\mathrm{Mpc})^3$. The data are linear with $R^2 = 0.9989$.

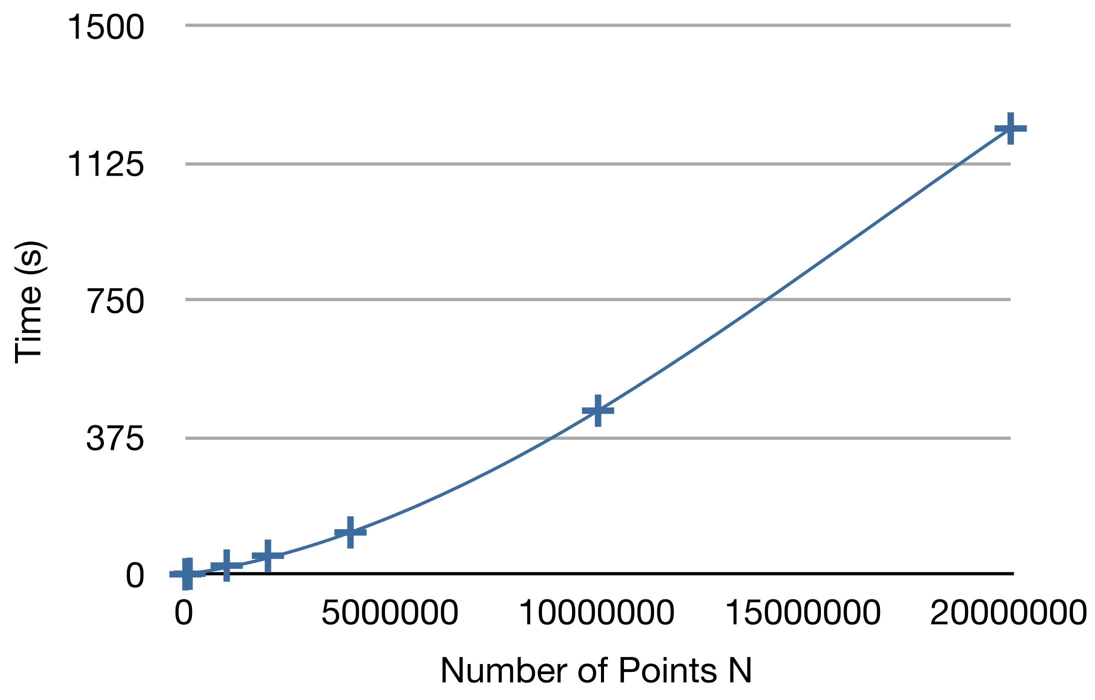


Figure 20: Execution time for 2-point correlation counts on Poisson data with density that increases with N. The data sets are contained in a cube of side length 1000Mpc/h. The density is thus $N \times 10^{-9}$ points per Mpc$^3h^{-3}$. The data fit a parabola with $R^2 = 0.9996$.

the density grows as N. As shown in Figures 20 and 21, these runtimes scale super-linearly. Both these results support our theoretical bounds in Section 3.8.2.

3.8.1.4 Overall Results

To the best of our knowledge, the previous largest 3pcf calculation of real scientific interest used 106,824 galaxies, 30 jackknife subsamples, and 45 different matchers. This required approximately 30,000 cpu-hours [117, 119].

By combining our efficient resampling multi-matcher, and optimized base case implementations, we were able to compute the 3pcf of 1.1 million galaxy positions at the same scales and using 27 resampling regions in 112 cpu-hours using a commodity cluster. Note that this calculation requires computing the *Raw Correlation Counts* on a random set of 20 million points. To our knowledge, this represents the largest

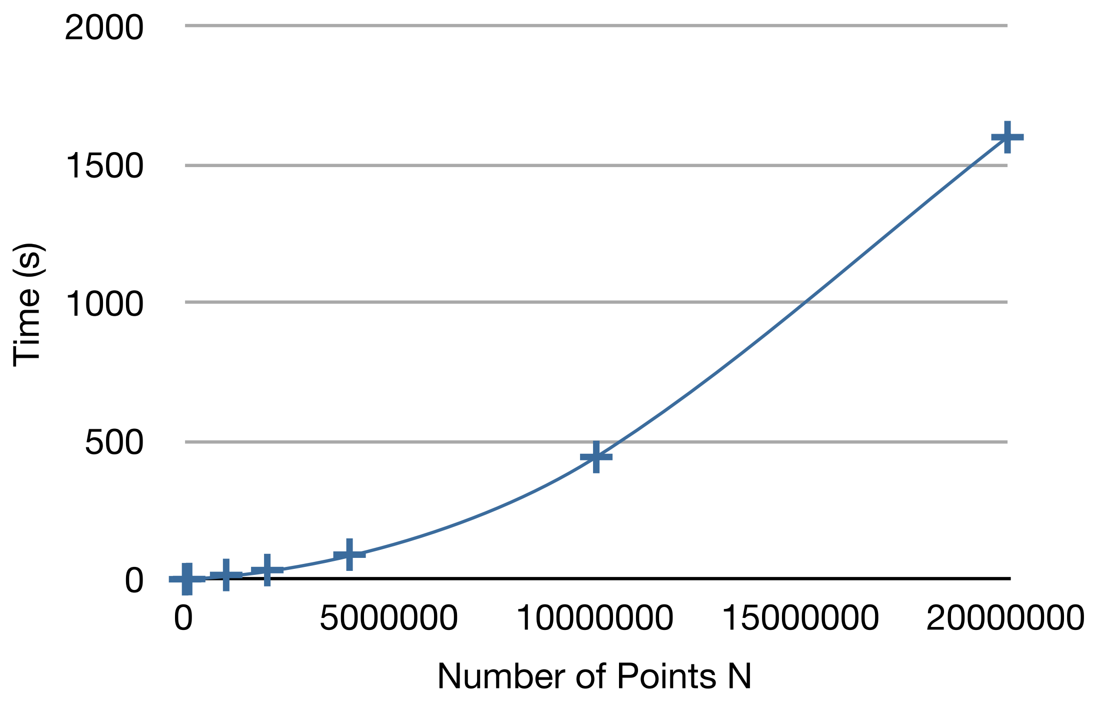


Figure 21: Execution time for 3-point correlation counts on Poisson data with density that increases with N. The data sets are contained in a cube of side length 1000Mpc$/h$. The density is thus $N \times 10^{-9}$ points per Mpc$^3h^{-3}$. The data fit a degree 3 polynomial with $R^2 = 0.9999$.

3pcf calculation of large scale structure at scientifically relevant scales.

3.8.2 Runtime Theory

We now turn to a theoretical analysis of the runtime for our algorithms. As was the case for the EMST problem discussed previously, worst-case analysis will be too blunt a tool for this analysis. An adversary can easily construct an input for which our algorithms take the worst case $O(N^n)$ time. For example, consider the raw correlation counts for 2pcf estimation for a matcher $(0, r)$. Let all N points lie in a ball of diameter r and let the rest of the sample window be empty. Then, our algorithm cannot prune any pair of points, and will have to consider all $O(N^2)$ pairs. Even for the uniformly distributed data, worst-case analysis is not suitable. For a point set drawn from a Poisson ensemble, there is still a non-zero probability that a realization like the one described above will be chosen.

Since we are dealing with random processes, expected runtime is a natural choice for the analysis. We will give expected runtime results in terms of intrinsic properties of the data. Since we are dealing with point processes, the natural properties of the data are the n-point correlations themselves. The most important quantity is the density (or 1-point correlation) ρ. We will show that this is sufficient to characterize the runtimes of our algorithms on Poisson data. This result is useful on its own, since in practice the random set used in Monte Carlo npcf estimators is much larger than the data, so the computations on the random set often dominate the total time. We will also characterize the runtimes on structured data using correlation integrals – integrals over the n-point correlation functions. The n-point correlation functions can be considered as perturbations on a Poisson distribution. These integrals show up in our analysis in a similar fashion – as the amount of extra work required (in expectation) over a Poisson set of the same size.

Throughout, we consider only data-data computations for simplicity. These results generalize to data-random computations in a natural way. We denote the size of the data set by N, the order of the correlation being estimated by n, the (sample) density of the data by ρ, and the largest upper bound distance in the matcher by $\hat{r}$. Throughout, we work in three dimensional space.

In this section, we show the following:

Claim 3.8.1. *Raw correlation counts for npcf estimation can be computed in $O(N)$ time after $O(N \log N)$-time tree construction.*

The space partitioning trees considered here require $O(N \log N)$ time for tree construction. This step can be done as a pre-computation, and in general, this accounts for a small fraction of the overall runtime. We therefore focus on the actual computation of the raw correlation counts, since this will dominate the runtime.

Intuitively, as the sample window and number of points grow, the number of points close to a given point will remain small. Therefore, the number of tuples which satisfy a given matcher (which does not grow with the size of the data) and which include a given point will remain constant. Since all points in a tuple must be close, the total number of tuples is at most $O(N)$.

In order to turn the above claim into a proof, we require several steps. First, we must make precise what we mean by asymptotic growth in the number of points. If the sample window remains fixed while more points are contained within it, then the density of points will increase. Our analysis will make use of this density, and analogs in the higher-order correlation functions, as runtime parameters.

We then give a formal proof that the number of tuples satisfying a matcher grows as $O(N)$ under reasonable assumptions. Combined with a result that our interaction list-based tree traversal algorithm runs in $O(N)$ time, we arrive at a proof of the claim above.

We bound the runtime in terms of the number of points N and the density of the point process ρ. The size of the data is a natural parameter. The density ρ gives us the expected number of points in a unit volume in a Poisson set. Intuitively, a set with a high density will have more points satisfying a given matcher and will thus require more time to compute. Also, although the number of points can affect the density, it does not directly control it. As N increases, the size of the sample window may grow as well. Depending on these relative rates of growth, the density may increase, decrease, or remain constant. Therefore, the runtime will depend on the density as well as the overall number of points.

There are two sources of work in the algorithm: the tree traversals (and pruning checks) and the base case computations. If we can bound the amount of work in each, then we can bound the total work required.

3.8.2.1 Computing Interaction Lists

We begin by examining the formation of interaction lists in the pairwise traversal algorithm (Algorithm 3.5.5).

Lemma 3.8.2. *The formation of interaction lists in Algorithm 3.5.6 requires $O(N \cdot \rho)$ work for any order of correlation using an octree on Poisson data.*

Proof. We begin by rescaling our data to fit within a unit cube. When we do this, we must also rescale the distances in the matcher by the same amount. Let $\hat{r}$ be the largest upper bound distance in the matcher. Then, assume we scale the data in each dimension by a factor b. Then, $r = \hat{r} \cdot b^{-1}$ is the maximum upper bound distance in the rescaled matcher.

A node at level i has sides of length 2^{-i}. There are 2^{3i} nodes at level i. Since the data are uniformly distributed, the expected depth of our octree is $\log_8 N$.

A node at level i is contained within a ball of radius $\sqrt{3}2^{-(i+1)}$ at the center of the node. This node can prune any other node except those that lie within a radius of:

$$r_b = r + \sqrt{3}2^{-i} + \sqrt{3}2^{-(i+1)} \tag{3.8.1}$$

where the second term comes from the diameter of a box.

The number of boxes that intersect a ball of radius $r + \sqrt{3}\left(2^{-i} + 2^{-(i+1)}\right)$ at level i is bounded by the number that can be packed into a ball of radius $r + \sqrt{3}2^{-(i-1)}$. This number B_i is bounded by the ratio of these two volumes:

$$B_i \leq \frac{\frac{4}{3}\pi\left[r + \sqrt{3}\left(2^{-(i-1)}\right)\right]^3}{2^{-3i}} \tag{3.8.2}$$

Since it takes constant work to check for the ability to prune and then split a node, the total amount of work done at level i of the tree can be bounded by

$$\begin{aligned} W_i &= 2^{3i}B_i && (3.8.3)\\ &\leq \frac{4}{3}\pi 2^{6i}\left[r + \sqrt{3}2^{-(i-1)}\right]^3 && (3.8.4)\\ &= \frac{4}{3}\pi 2^{6i}\left[r^3 + 3\sqrt{3}r^2 2^{-(i-1)} + 9r2^{-2(i-1)} + 3\sqrt{3}\cdot 2^{-3(i-1)}\right] && (3.8.5)\end{aligned}$$

We can sum this over all levels of the tree.

Breaking this down term-by-term, the first term gives us

$$\begin{aligned} \sum_{i=0}^{\log_8 N}\frac{4}{3}\pi 2^{6i}r^3 &= \frac{4}{3}\pi r^3\sum_{i=0}^{\log_8 N}(8^i)^2 && (3.8.6)\\ &= r^3\sum_{i=0}^{\log_8 N}64^i && (3.8.7)\\ &= r^3\frac{1-64^{\log_8 N}}{1-64} && (3.8.8)\\ &= O(r^3N^2) && (3.8.9)\end{aligned}$$

Similarly, the second term is $O(r^2N^{\frac{5}{3}})$, the third is $O(rN^{\frac{4}{3}})$, and the last is $O(N)$. So, the total amount of work required is

$$W = O(r^3N^2 + r^2N^{\frac{5}{3}} + rN^{\frac{4}{3}} + N) \tag{3.8.10}$$

Recall that r is the scaled upper bound distance in the matcher. So, this bound on the total work depends on how the data were scaled. We assumed that the data originally were contained in a cube of side length b, which we rescaled so that $r = \hat{r} \cdot b^{-1}$. Therefore, the density of the data is

$$\rho = \frac{N}{b^3} \tag{3.8.11}$$

From this, we see that

$$\rho = \frac{N r^3}{\hat{r}^3} \tag{3.8.12}$$

Assuming that the matcher distance of interest $\hat{r}^3$ is fixed as N increases, then by substituting into Equation 3.8.10, we have

$$W = O(N \cdot \rho) \tag{3.8.13}$$

□

3.8.2.2 Base Case Work

We now discuss the amount of work required in the base cases.

Lemma 3.8.3. *The work required in the base case of Algorithm 3.5.5 is bounded by $O(\rho^{n-1}N)$ for an n-point correlation computation on a Poisson distributed set.*

Proof. Using the proof of lemma 3.8.2 above, we know that for a given node, we have pruned all other nodes within a given distance. From this, we can obtain the number of nodes in the interaction list of a given node. The expected total work in the base case is bounded by the number of n-tuples of points that were not pruned in the formation of the interaction lists.

From Equation 3.8.2, we know that for a given box, the number of boxes in it's interaction list is bounded by

$$\begin{aligned} B_{\log_8(N)} &\leq \frac{\frac{4}{3}\pi \left[r + \sqrt{3}\left(2^{-(\log_8(N)-1)}\right)\right]^3}{2^{-3\log_8(N)}} && (3.8.14)\\ &= \frac{4}{3}\pi N \left[r + 2\sqrt{3}N^{-\frac{1}{3}}\right]^3 && (3.8.15)\\ &= \frac{4}{3}\pi \left(r^3 N + 6\sqrt{3}r^2 N^{\frac{2}{3}} + 18 r N^{\frac{1}{3}} + 24\sqrt{3}\right) && (3.8.16) \end{aligned}$$

In expectation, since we have taken the tree to depth $\log_8(N)$, each box has a single point. The total work is bounded by the number of tuples that can lie within this distance. For each node, there are at most $O(B_i^{n-1})$ tuples to consider. Since there are N nodes, the work is bounded by

$$W \leq N \cdot O\left[\left(r^3 N + r^2 N^{\frac{2}{3}} + r N^{\frac{1}{3}} + N\right)^{n-1}\right] \tag{3.8.17}$$

Note that the true amount of work will be reduced through symmetry; however, this only reduces the constant pre factor.

As before, we can relate r to ρ through Equation 3.8.12. Therefore, we see that

$$W = O\left(\rho^{n-1} N\right) \tag{3.8.18}$$

□

3.8.2.3 Complete Results

Theorem 3.8.4. *Using Algorithm 3.5.5, the raw correlation counts for npcf estimation can be computed in $O(\rho^{n-1}N)$ time for Poisson data and any value of n after tree construction.*

Proof. After tree construction, the algorithm simply constructs the interaction lists. It then, in the case that none of the base cases can be pruned in line 4, considers every n-tuple of leaf nodes and computes a base case on them. Therefore, the runtime is the combination of the times needed for these two steps. The proof follows immediately from the lemmas 3.8.2 and 3.8.3. □

Corollary 3.8.5. *The multi-matcher algorithm (Algorithm 3.5.7) using the pairwise traversal runs in $O(\rho^{n-1}N)$ time after $O(N \log N)$ tree construction time.*

Proof. The proofs used here only refer to the largest upper bound distance in the matcher . We can apply the same reasoning to the largest upper bound distance among all the matchers considered to obtain the result. □

We have shown that for Poisson data for which the density does not grow with N (or in other words, if the sample window grows along with the number of data points), then the raw correlation counts can be obtained in $O(N)$ time for any order of correlation. This restricted case is still of considerable interest, since much of the time in a real npcf calculation is spent obtaining raw correlation counts for the Poisson set. However, the natural follow-up problem is to bound the runtime of the algorithm on more general data sets.

The key difficulty comes in quantifying the departure of the real data set from a Poisson set. If we could do so, then it may be possible to modify the proofs above to take into account these perturbations. Unfortunately, the natural way to quantify a distributions distance from Poisson is through the n-point correlation functions. This suggests an output-sensitive analysis – one which shows the dependence of the runtime on the size of the correlation functions estimated.

The proofs above rely on the Poisson distribution in two fundamental ways: in counting the number of points (or boxes) that lie within a given radius of a fixed point and in ensuring that the octree is shallow and points are evenly distributed throughout it. In the case of counting points within a given volume, this can be done through the n-point correlations. As noted previously, the npcf completely determines the distribution of counts in fixed volumes. Given this distribution, it should be possible to bound the number of points to be considered in base cases.

As for the tree construction, octrees were used in this analysis for simplicity. More sophisticated trees can guarantee logarithmic depth for less than perfect input distributions. Given these two factors, it seems possible, although challenging, to generalize our bounds to general input distributions.

CHAPTER IV

FOCK MATRIX CONSTRUCTION IN HARTREE-FOCK THEORY

We now turn to a radically different problem: estimating electronic wavefunctions in Hartree-Fock theory. Given a molecule (or complex of many molecules), we would like to be able to accurately predict its behavior under a variety of circumstances. Classical molecular dynamics simulations are effective for some tasks, but are unable to handle fundamentally quantum mechanical problems, such as bond breaking and formation. Instead, we would like a fully rigorous, quantum theory that will allow us to handle general molecular systems.

Hartree-Fock theory provides an upper bound on the ground-state energy of a system of nuclei and electrons. This bound can be tightened with more accurate theories, such as configuration interaction and coupled cluster theories. These methods use the Hartree-Fock ground state as an input. Additionally, the bound can be improved by increasing the size of the *basis set* (described below).

Unfortunately, the rate-limiting step in a Hartree-Fock calculation scales as $O(N^4)$. This step is the construction of the *Fock matrix*, an $N \times N$ symmetric (possibly dense) matrix, each entry of which is composed of two nested sums, each over N terms. Here, we describe the computational task in Hartree-Fock theory in more detail. We also discuss some existing approaches for overcoming the $O(N^4)$ bottleneck.

We then examine the application of multi-tree algorithms to this fundamental computational problem. We present several new contributions:

- We unify existing efficient algorithms for this problem with the common framework provided by multi-tree algorithms.

- We explore several avenues for the application of general multi-tree ideas to this problem and demonstrate some preliminary empirical results.

- We present the first real theoretical runtime analyses of both the most commonly used algorithms in the literature and our own algorithms.

We begin by examining the underlying computational task in Hartree-Fock theory. After highlighting the problem, we discuss existing methods from the literature. We then turn to an in-depth discussion of two of these algorithms, the Continuous Fast Multipole Method and the Linear K algorithm, and show how these fit within our multi-tree framework. We then introduce two new multi-tree algorithms. We discuss rigorous runtime guarantees for each of these algorithms and show some preliminary empirical results. We conclude by discussing future directions for this body of work.

4.1 Hartree-Fock Theory

Hartree-Fock theory is a used to approximate the total energy of a given configuration of atomic nuclei by computing an approximate wavefunction to describe the distribution of electrons around those nuclei.

We have M atomic nuclei (of known charges Z_A and masses M_A) indexed by A and B at and K electrons, indexed by i, j, k, l. Our task is to solve the time-independent Schrodinger equation (given here in atomic units) to determine the minimum energy configuration of this system:

$$H\Psi(\mathbf{r}_i : i = 1, \ldots, K; \mathbf{r}_A : A = 1, \ldots M) = E\Psi(\mathbf{r}_i : i = 1, \ldots, K; \mathbf{r}_A : A = 1, \ldots M) \tag{4.1.1}$$

where the wavefunction Ψ is a function of the K electronic coordinates $\mathbf{r}_i$ and the M nuclear coordinates $\mathbf{r}_A$, and the Hamiltonian operator is given by:

$$H = -\frac{1}{2}\sum_i \nabla_i^2 - \frac{1}{2}\sum_A M_A \nabla_A^2 - \sum_i^K \sum_A^M \frac{Z_A}{\mathbf{r}_{iA}} + \sum_{i,j}^K \frac{1}{\mathbf{r}_{ij}} + \sum_{A,B}^M \frac{Z_A Z_B}{\mathbf{r}_{AB}} \tag{4.1.2}$$

Following standard practice, we invoke a series of approximations to make the above problem tractable. First, we decouple the electronic and nuclear interactions (the *Born-Oppenheimer approximation*) and consider the nuclei to be fixed, classical point charges. Under this assumption, we take the positions of the nuclei as input and use a wavefunction which depends only on the positions of the electrons.

We further assume that the electronic wavefunction factors:

$$\Psi(\mathbf{r}_i : i = 1, \ldots, K) = \prod_{i=1}^{K} \psi_i(\mathbf{r}_i) \tag{4.1.3}$$

This is a powerful assumption which eliminates electron correlation effects. However, it is necessary for tractability, and correlation effects can be replaced later. We must also enforce the quantum mechanical postulate that the wavefunction be antisymmetric in its arguments. Therefore, rather than the simple form in Equation 4.1.3, we use a *Slater determinant*, written $[\psi_1(\mathbf{r}_1) \cdots \psi_K(\mathbf{r}_K]$. This is the following (symbolic) determinant:

$$\begin{vmatrix} \psi_1(\mathbf{r}_1) & \psi_1(\mathbf{r}_2) & \cdot & \psi_1(\mathbf{r}_K) \\ \psi_2(\mathbf{r}_1) & \psi_2(\mathbf{r}_2) & \cdot & \psi_2(\mathbf{r}_K) \\ \vdots & \vdots & \vdots & \vdots \\ \psi_K(\mathbf{r}_1) & \psi_K(\mathbf{r}_2) & \cdot & \psi_K(\mathbf{r}_K) \end{vmatrix} \tag{4.1.4}$$

It can be easily verified that Slater determinants have the necessary anti-symmetry property.

Furthermore, for this discussion, we will assume that there are no open molecular orbitals – all electrons belong to spin-antispin pairs. This leads to *restricted* Hartree-Fock theory. Generalizing this discussion to open-shell or unrestricted HF is straightforward, but is eliminated for simplicity.

In order to arrive at a computationally tractable formulation, we require one more important approximation. We assume each of the single-electron wavefunctions ψ_i

are a linear combination of a set of *basis functions*:

$$\psi_i(\mathbf{r}) = \sum_{j=1}^{N} c_{ij}\phi_j(\mathbf{r}) \tag{4.1.5}$$

By making these assumptions, our task has been reduced from solving the full Schrodinger equation to determining the matrix of coefficients C.

These basis functions are generally chosen to approximate the known solutions to the Schrodinger equation for single atoms. These are commonly known as the s, p, d etc. shells.

The most commonly used basis functions (and those considered in this chapter) consist of atom-centered Gaussians. A "primitive" basis function is given by:

$$\phi_j(\mathbf{r}) = N_A(x - A_x)^{l_j}(y - A_y)^{m_j}(z - A_z)^{n_j} \exp\left(-\alpha_j\|\mathbf{r} - \mathbf{A}\|^2\right) \tag{4.1.6}$$

where the vector $\mathbf{A}$ is the position of a nucleus on which the function is centered, the angular momentum is $L = (l_j + m_j + n_j)$, α_j is the exponent or bandwidth of the function, and N_A is a dimensionless normalization factor. The function is defined for any $\mathbf{r}$ in $\mathbb{R}^3$.

A basis set is generally specified by atom. Typically, a basis set consists of a set of functions for each different atomic number. When performing a calculation, we use a basis consisting of the functions for each atomic type centered on the nucleus of each atom of that type. In practice, basis sets generally consist of "contracted" basis functions. Each basis function is itself a linear combination of primitive basis functions.

The names of basis sets often include the number of functions in the contractions. For instance, the commonly used 6-31G basis consists of a single contracted function, comprised of six primitive functions, for inner shells and two contracted functions, with three and one primitive, for valence shells. The "G" stands for Gaussians, the form of all the primitive functions. In a calculation using the 6-31G basis, each hydrogen atom will have the same two primitives to represent its s shell. Each oxygen atom will have a single function comprised of six primitives for its s shell and two functions with three and one primitive for the sp shell. The momenta of the functions correspond to the momenta of the shells they represent – so s functions have $L = 0$, p functions have $L = 1$, and so on.

Basis functions are grouped into shells. So, a p shell of functions consists of three separate functions, all with the same center and exponent, and exactly one of l_j, m_j, n_j equal to one for each function.

A full Hartree-Fock calculation takes as inputs:

- A set of nuclear coordinates A and atomic numbers Z_A.
- A number of electrons – K.
- A basis set consisting of N contracted functions.

and outputs an N approximate electronic wavefunctions and corresponding energies.

4.1.1 The Roothan-Hall Equations

By inserting the approximations listed above to the Schrodinger equation, we obtain the following linear algebraic formulation [162]. In place of the Schrodinger equation, we obtain the Roothan-Hall equation:

$$FC = SCE \tag{4.1.7}$$

where

- S is the $N \times N$ overlap matrix: $S_{ij} = \int d\mathbf{r}\ \phi_i(r)\phi_j(r)$ for basis functions i and j.
- E is a $N \times N$ diagonal matrix of eigenvalues.
- C is the $N \times N$ matrix of basis weights, with entries given in Equation 4.1.5.

- F is the $N \times N$ *Fock matrix*, the Hamiltonian in the basis given by our Gaussian functions and defined below.

The Fock matrix can be decomposed as:

$$F = H^{\text{core}} + J - \frac{1}{2}K \tag{4.1.8}$$

where

- H^{core} contains the kinetic and nuclear attraction terms and can be precomputed.
- J is the Coulomb matrix, representing the electrostatic repulsion between electrons.
- K is the exchange matrix, a term which arises from anti-symmetry.

The elements of J and K are given by:

$$J_{ij} - \frac{1}{2}K_{ij} = \sum_k \sum_l D_{kl} \left[(ij|kl) - \frac{1}{2}(ik|jl) \right] \tag{4.1.9}$$

where $(ij|kl)$ is a two-electron integral (defined below). D is the density matrix, $D_{kl} = \sum_m^{\text{occ}} C_{km}C_{lm}$, where the sum is over the columns of C corresponding to the $K/2$ smallest eigenvalues in E (the "occupied" orbitals).

The key quantity in the Fock matrix definition is the two-electron integral:

$$(ij|kl) = \int d\mathbf{r}_1 d\mathbf{r}_2 \phi_i(\mathbf{r}_1)\phi_j(\mathbf{r}_1) \frac{1}{\|\mathbf{r}_1 - \mathbf{r}_2\|} \phi_k(\mathbf{r}_2)\phi_l(\mathbf{r}_2) \tag{4.1.10}$$

where $\mathbf{r}_1$ and $\mathbf{r}_2$ are dummy variables integrated over all of $\mathbb{R}^3$. If the basis functions are Gaussians (or linear combinations of Gaussians) then this integral has a closed form solution.

In the formulation above (Eqn. 4.1.7), the Fock matrix depends on the density matrix, which depends on the coefficient matrix C, which is in turn the solution to the equation. Therefore, we must iteratively compute a solution self-consistently, shown in Algorithm 4.1.1. The generalized eigenvalue problem can be solved in $O(N^3)$ time.

Algorithm 4.1.1 SCF (Basis Set, Nuclear coordinates, Number of electrons K)

```
   Precompute $S$ and $H_{ij}^{\text{core}}$
   Construct guess density $D^{(0)}$
3: **while** $\|D^{(n)} - D^{(n-1)}\| > \epsilon$ **do**
     Compute $F^{(n)}$ from $D^{(n)}$ using Eqns. 4.1.8 and 4.1.9
     Solve $FC = SCE$ to get $C^{(n)}$ and $E^{(n)}$
6:   Sum over the $K/2$ lowest energy levels to get $D^{(n+1)}$
     $n \leftarrow n + 1$
   **end while**
```

The computation of the new Fock matrix (Line 4) is the rate-limiting step, since it requires $O(N^2)$ operations for each of the $O(N^2)$ entries, for a total complexity of $O(N^4)$.

Each step of this algorithm is subject to considerable attention in the literature. The construction of a guess density in line 2 is crucial to efficiently finding an accurate solution. The density update and convergence checks can be performed by more sophisticated methods such as the direct inversion of the iterative subspace (DIIS) [140]. However, in this work, we restrict our attention to the asymptotic rate limiting step: the construction of the Fock matrix in line 4.

Since the H^{core} contribution to F can be precomputed before the SCF iterations, we focus on the construction of the Coulomb and exchange matrices. The construction of these matrices is the focus of the remainder of this chapter.

Problem 4.1.1. ***Fock Matrix Construction.*** *Given a set of N (Gaussian) basis functions, M nuclei, and a density matrix D, compute the Coulomb and exchange matrices:*

$$J_{ij} = \sum_{k=1}^{N} \sum_{l=1}^{N} D_{kl} \, (ij|kl) \tag{4.1.11}$$

$$K_{ij} = \sum_{k=1}^{N} \sum_{l=1}^{N} D_{jl} \, (ik|jl) \tag{4.1.12}$$

for $1 \leq i, j \leq N$.

4.1.2 Two-Electron Integrals

Problem 4.1.1 requires repeated evaluation of the two-electron integrals in Equation 4.1.10. We briefly discuss the computational challenges involved in evaluating this integral.

Gaussian basis functions allow a closed-form solution to Equation 4.1.10. The key to evaluating this integral lies in application of the Gaussian product theorem. In Equation 4.1.10, ϕ_i and ϕ_j are Gaussian basis functions over a common variable with centers at r_i and r_j. Applying the GPT to two zero-momentum basis functions, we have

$$\phi_i(r)\phi_j(r) = N_i N_j \exp(-\beta) r_{ij}^2) \times \exp\left(-\beta \|r - r_{ij}\|^2\right) \tag{4.1.13}$$

Where

$$\begin{aligned} \beta &= \alpha_i \alpha_j / (\alpha_i + \alpha_j) & (4.1.14) \\ r_{ij} &= \frac{\alpha_i r_i + \alpha_j r_j}{\alpha_i + \alpha_j} & (4.1.15) \end{aligned}$$

Note that all terms except the last one are independent of the dummy variable r. Similarly, we can apply the GPT to functions of arbitrary angular momentum. This again results in a term of the form $\exp(-\beta r_{ij}^2)$ times a summation of length equal to the total angular momentum.

We obtain the full two-electron integral for four zero-momentum primitive Gaussians by applying the Gaussian product theorem to both pairs of functions.

$$\begin{aligned} (ij|kl) &= M\left[(\alpha_i + \alpha_j)(\alpha_k + \alpha_l)\sqrt{\alpha_i + \alpha_j + \alpha_k + \alpha_l}\right]^{-1} \\ &\quad \times \exp\left(-\tfrac{\alpha_i \alpha_j}{\alpha_i + \alpha_j}\left\|\tfrac{\alpha_i A_i - \alpha_j A_j}{\alpha_i + \alpha_j}\right\|^2\right) \times \exp\left(-\tfrac{\alpha_k \alpha_l}{\alpha_k + \alpha_l}\left\|\tfrac{\alpha_k A_k - \alpha_l A_l}{\alpha_k + \alpha_l}\right\|^2\right) \\ &\quad \times \operatorname{erf}\left(\left\|\tfrac{\alpha_i A_i - \alpha_j A_j}{\alpha_i + \alpha_j} - \tfrac{\alpha_k A_k - \alpha_l A_l}{\alpha_k + \alpha_l}\right\| \times \sqrt{\tfrac{(\alpha_i + \alpha_j)(\alpha_k + \alpha_l)}{\alpha_i + \alpha_j + \alpha_k + \alpha_l}}\right) \end{aligned} \tag{4.1.16}$$

where erf is the error function and M is a constant times the product of the normalization factors. Integrals between higher momentum basis functions can be derived from taking derivatives of the zero-momentum integral. This leads to a sum of terms

[47].

$$\begin{aligned}(ij|kl) &= M \exp\left(-\tfrac{\alpha_i\alpha_j}{\alpha_i+\alpha_j}\left\|\tfrac{\alpha_i A_i-\alpha_j A_j}{\alpha_i+\alpha_j}\right\|^2\right)\\ &\quad\times\exp\left(-\tfrac{\alpha_k\alpha_l}{\alpha_k+\alpha_l}\left\|\tfrac{\alpha_k A_k-\alpha_l A_l}{\alpha_k+\alpha_l}\right\|^2\right)\\ &\quad\times\textstyle\sum_{m=0}^{L} C_m F_m(\|r_{ij}-r_{kl}\|)\end{aligned} \tag{4.1.17}$$

where L is the sum of the momenta of the four functions, the coefficients C_m depend on the exponents and distances between the functions and

$$F_m(x) = \int_0^1 dt\ t^{2m}\exp(-xt^2) \tag{4.1.18}$$

Contracted integrals are just linear combinations of primitive integrals due to the linearity of the integral operator.

The form of Equation 4.1.17 suggests that there are many common terms between different integrals. For instance, let i, j, and k be zero-momentum functions and let l have momentum 1. As noted above, in all commonly used basis sets, there are three functions l with the same center and exponent but with different angular momentum directions. Since many of the terms in Equation 4.1.17 do not depend on the angular momentum, it makes sense to compute them once. Furthermore, in practice, higher-momentum integrals are constructed from lower momentum intermediate quantities through recurrence relations [81, 126].

Both of these observations suggest the importance of caching and reusing intermediate quantities in integral construction. In particular, integrals involving functions in the same shell (e.g. the three p functions or six d functions) are computed together in a batch. A set of four shells (or its corresponding integrals) is referred to as a *shell quartet*. Throughout, we will always compute integrals between four shells, in keeping with standard integral evaluation codes.

In keeping with standard notation in quantum mechanics, we sometimes refer to the pair of shells i and j and the "bra" pair and k, l as the "ket" pair. We will also commonly refer to such pairs as *shell pairs*, and two shell pairs (four shells) as a *shell quartet*.

Efficient evaluation of integrals is a complicated and demanding topic. For the remainder of this chapter, we assume access to an efficient integral evaluation engine. Both the related work and new algorithms discussed below will make calls to this engine for particular integrals. A key measure of speedup will be the number of integral computations avoided through these methods.

4.1.2.1 Integral Symmetry

From the definition of the two-electron integral (Equation 4.1.10), we can see that the integral $(ij|jk)$ has an eight-fold symmetry. The functions in the bra and ket pair can each be swapped, since multiplication is commutative. Also, the bra and ket pairs themselves can be swapped by renaming the dummy variables of integration.

Since the computation of integrals is the most intensive step in Fock matrix construction, it is important to take advantage of these symmetries. For instance, in the Coulomb matrix, the integral $(ij|kl)$ is multiplied by the density entries D_{kl} and D_{lk}, and each of these results are added to J_{ij} and J_{kl}. When we compute an integral, we can obtain a significant (though constant as N grows) improvement by contracting it with all necessary density entires at once. The algorithms discussed below will make extensive use of the symmetry within the bra and ket pairs.

4.1.2.2 Bounding Two-Electron Integrals

The two-electron integrals are generally very expensive to compute exactly. Therefore, the key to efficient Fock matrix construction algorithms lies in efficiently computing bounds or approximations on two-electron integrals. Here, we quickly survey some of the widely used techniques.

The most common bound on the two electron integral comes from application of the Schwarz inequality [80]. The two-electron integral satisfies the axioms for an inner product between shell pairs. Therefore, we can apply the Schwarz inequality to get

$$(ij|kl) \leq (ij|ij)^{\frac{1}{2}} (kl|kl)^{\frac{1}{2}} \tag{4.1.19}$$

We frequently write Q_{ij} for $(ij|ij)^{\frac{1}{2}}$. Note that we have decomposed a four-index quantity into a product of two two-index quantities. This is an important potential source of efficiency improvements, since it is much easier to precompute and store two-index quantities.

Other bounds build on this foundation. Gill, Johnson, and Pople [67] proposed another bound using the Holder inequality applied to integrals over charge distributions from the bra or ket pair. Gadre *et al.* focused on approximations that avoid computing the more expensive parts of a two-electron integral [64]. However, these bounds often require additional computational overhead to use, thus limiting their applicability for efficient approximation algorithms. Furthermore, these methods only provide approximations of the integral, rather than rigorous bounds. Since these make the overall error more difficult to control, they often require tighter convergence criteria and lower approximation thresholds. These requirements can easily eliminate any computational savings the bounds provide.

The Schwarz inequality-based bounds correctly capture the exponential decay of the integrals as the distance between the two functions in the bra or ket pair increases. However, the full integral also decreases linearly as the distance between the two contracted centers increases, while the Schwarz bound does not. More sophisticated bounds attempt to capture this behavior. The Multipole Based Integral Estimates (MBIE) [102], provide a rigorous bound on integral behavior. However, these suffer front he expensive multipole pre computations required to use them, thus limiting their effectiveness.

The QQR bounds [116] attempt to overcome this problem. They approximate an integral as the Coulomb interaction between a pair of classical charge distributions. The charge distributions are obtained from the Gaussian product theorem on the bra and ket pair.

$$(ij|kl) \approx \frac{Q_{ij}Q_{kl}}{r - \mathrm{ext}'_{ij} - \mathrm{ext}'_{kl}} \tag{4.1.20}$$

where R is the distance between the center of the bra and ket shell pairs and

$$\text{ext}'_{ij} = \sqrt{\frac{2}{\alpha_i + \alpha_j}}\text{erfc}^{-1}(\theta) \tag{4.1.21}$$

for an error threshold θ. While the QQR approximation can be computed extremely efficiently, it is not strictly an upper bound for the true integral.

4.2 Fock Matrix Construction Algorithms

Since the creation of the new Fock matrix in each iteration is the bottleneck, we focus on methods to efficiently approximate it. In order to exactly (up to numerical precision) compute the Fock matrix at each iteration, we must exactly (again up to machine precision) compute all $O(N^4)$ two-electron integrals. Since the integrals only depend on the (fixed) basis functions, they can be computed once, stored, and re-used in each SCF iteration. With this method, it is only necessary to contract the two-electron integrals with the density matrix in each step. However, this naive method is doomed by it's $O(N^4)$ storage requirement and computation time.

Depending on the amount of time required to access disk, it may be faster to simply recompute integrals as needed. The *Direct SCF* method, first proposed by Almlof, Faegri, and Korsell [5], attempts to overcome the $O(N^4)$ bottleneck. In Direct SCF, rather than computing and storing all two-electron integrals, we will only compute those that are "significant" for a particular Fock matrix and neglect the rest. If we can avoid computing enough of the integrals in each step, this will be significantly faster than the naive method.

The original Direct SCF paper determined if an integral is significant by computing an estimate which neglects angular momentum. A more effective and widely-used bound uses the Schwarz inequality bound from Equation 4.1.19. As a preprocessing step, we find all the "significant" bra or ket shell pairs (Algorithm 4.2.1). We then iterate over all pairs of significant shell pairs, check if the integral is large enough to be

Algorithm 4.2.1 FormSignificantShellPairs(Basis Set B, error threshold ϵ)

```
List $SP = \emptyset$
for all pairs of shells $i, j \in B$ do
    Compute $Q_{ij} = \sqrt{(ij|ij)}$
    if $Q_{ij} > \epsilon_1$ then
        Add $(i, j)$ to $SP$.
    end if
end for
Return $SP$
```

Algorithm 4.2.2 DirectSCF(Basis Set B, error thresholds ϵ_1, ϵ_2)

```
$SP$ = FormSignificantShellPairs($B$, $\epsilon_1$)
Given a new density matrix $D$
for all Shell pairs $(i, j) \in SP$ do
    for all Shell pairs $(k, l) \in SP$ do
        if $Q_{ij} \cdot Q_{kl} > \epsilon_2$ then
            Compute $(ij|kl)$ and contract with $D_{kl}, D_{lk}, D_{ij}, D_{ji}$
        end if
    end for
end for
```

computed, and if so, compute it and contract it with the density matrix (Algorithm 4.2.2). Different bounds can also be substituted for the Schwarz bound.

In the literature, direct screening methods are generally claimed to scale as $O(N^2)$ for N basis functions[48]. The first part of Algorithm 4.2.2 clearly takes $O(N^2)$ work for N shells. The second part requires $O(T^2)$ work, where T is the number of significant shell pairs obtained from the first part. T generally claimed to scale as $O(N)$. However, it is straightforward to construct a counter example where T grows quadratically, for example, by placing all N shells on a common center. Below, we give the first rigorous adaptive analysis of the direct SCF method, and show that T does scale as $O(N)$ under assumptions satisfied by real computations of interest.

Direct computation of new Fock matrices is a widely used method. However, for very large molecules, the quadratic scaling of the method will still be prohibitively expensive. If the Coulomb and exchange matrices are sparse (with only $O(N)$ non-zero entries), it may be possible to compute them in $O(N)$ time. This would make truly large scale computations possible. We now turn to algorithms which exploit properties of the Coulomb and exchange matrices separately and attempt to scale linearly.

4.2.1 Coulomb Matrix Construction

We start by considering the Coulomb matrix. We can consider the terms $\phi_i(\mathbf{r}_1)\phi_j(\mathbf{r}_1)$ to be charge distributions $\Phi_{ij}(\mathbf{r}_1)$. Under this interpretation, the Coulomb matrix entry J_{ij} is simply the total electrostatic potential at the charge distribution Φ_{ij} due to all other charges Φ_{kl}. This is the N-body problem from Chapter 2 applied to charge distributions rather than point charges.

In the case of point charges, we saw that the *Fast Multipole Method* [76] can compute these potentials for N charges in $O(N)$ time. Several methods have proposed applying this approach for the Coulomb matrix entries [176, 100, 174, 160].

Essentially, these methods approximate the potential collections of distant charge distributions with multipole expansions. For example, the Continuous Fast Multipole Method (CFMM) [176] treats distant charge distributions as point charges or higher-order multipoles (for higher momentum distributions), then uses the same multipole translation operators as the FMM. This FMM-base approach leads to a faster algorithm. However, the error due to the point-charge approximation is not bounded formally. Also, one must first apply a prescreening method (such as the one mentioned above) to eliminate most of the $O(N^2)$ possible charge distributions.

The J-matrix engine optimizes the evaluation of integrals for the Coulomb case [175]. Rather than looping over independent calls to the integral construction engine (as in line 6 of Algorithm 4.2.2), common quantities are extracted from inner loops. The quantum chemical tree code (QCTC) combines the tree-based FMM approach for distant shell pairs with the J-matrix engine for nearby pairs for the best currently available speedups [29].

4.2.2 Exchange Matrix Computation

All of the Coulomb matrix methods discussed above rely on the fact that the density matrix entry only depends on the ket shell pair. This allows us to treat the computation as a variant of the N-body problem for FMM-based methods and is the key to the optimizations in the J-matrix engine. The exchange matrix does not have this simple form. Instead, the density matrix entry contracted with each integral depends on a shell from both the bra and the ket pair.

Efficient exchange matrix algorithms modify the direct SCF algorithm in Algorithm 4.2.2. The near-field exchange (NFX) method [23] only computes those integrals that are determined to be in the near-field of an FMM-based method. The order-N exchange (ONX) method [148] attempts to account for the $1/r$ dependence of the two-electron integral on the distance between the bra and ket pairs. The linear K (LinK) method [127] adds bounds on the density matrix for additional speedups for sparse systems with localized densities. We discuss this method in further detail below.

4.3 Multi-Tree Algorithms for Fock Matrix Construction

Computing the Fock matrix directly requires $O(N^4)$ work, and is thus completely intractable. Faster methods exist, but suffer from several limitations. Prescreening is still at best quadratic, even for systems where the density of basis functions in space does not increase with N. Coulomb and exchange matrix methods require separate implementations, lack rigorous error guarantees, and suffer from the same lack of formal runtime analysis and hidden dependence on how properties of the system scale with N. Thus, the field is still lacking a single, unified fast method for Fock matrix construction with both rigorous error bounds and runtime guarantees.

In this section, we discuss some preliminary steps toward overcoming these problems through the unifying framework of multi-tree algorithms. We present two new

Algorithm 4.3.1 MultiTree(Tree nodes I, J, K, L)

```
    if $I, J, K, L$ are leaves then
        Compute all integrals and contract with densities
    else
        Compute bounds $(IJ|KL)^{\min}, (IJ|KL)^{\max}$
5:      if $(IJ|KL)^{\max} | < \epsilon$ then
            Prune
        else if $| (IJ|KL)^{\max} - (IJ|KL)^{\min} | < \epsilon$ then
            $(IJ|KL)^* = \Big((IJ|KL)^{\max} + (IJ|KL)^{\min}\Big) /2$
            Contract $(IJ|KL)^*$ with densities
10:     else
            Split largest node and recurse
        end if
    end if
```

algorithms for Fock matrix construction and discuss the ongoing effort to make them competitive with existing methods. We then show how two of the most successful algorithms from the literature can be considered as multi-tree algorithms.

4.3.1 A New Multi-Tree Algorithm

A multi-tree approach suggests itself immediately. We build a tree on the centers of the shells. In addition to the bounding information on the centers of the shells, we store bounds on the exponents. We perform a four-way recursion. When considering nodes (I, J, K, L), we compute upper and lower bounds on distances. Combining these with bounds on the exponents, we can obtain lower and upper bounds on the integrals.

We make use of the following notation:

$$(IJ|KL)^{\min} = \min_{i \in I, j \in J, k \in K, l \in L} (ij|kl) \tag{4.3.1}$$

$$(IJ|KL)^{\max} = \max_{i \in I, j \in J, k \in K, l \in L} (ij|kl) \tag{4.3.2}$$

This algorithm can be used to compute the Coulomb and exchange matrices in a single pass, or separate passes can be used for each. This algorithm is straightforward to implement. However, there are several impediments to using it for very large problems.

The most important part of this algorithm is the computation of the bounds in line 4. Useful upper and lower bounds are difficult to obtain. Integrals can be negative or positive. Furthermore, many of the available bounds are not ideal. The kernel is not monotonic with respect to all of its arguments. In particular, the integral may change sign as the distance between the common centers of the bra and ket pairs changes. Most of the literature is interested in upper bounds or more generic approximations. Our method requires both upper and lower bounds. While the method can use approximations, doing so runs the risk of greater error in the matrix.

Simple bounds, such as the Schwarz bound in Equation 4.1.19, do not account for all of the distance dependencies of the integral. In order to achieve maximum speedups, we need a bound that correctly captures the exponential decay as the distance between functions in the bra or ket increases along with the linear decay in the distance between the two pairs. In particular, obtaining speedups from approximations in line 9 requires very tight bounds.

Our ultimate goal in Hartree-Fock computations is accuracy in the error of the final energy. However, this energy is the final result of many iterations of Fock matrix construction and diagonalization, making it difficult to control directly. The existing methods discussed above have several sources of error. First, sufficiently distant shell pairs are neglected entirely. Second, integrals which fall below some threshold are also neglected. Third, FMM-based methods incur some error in their treatment of far-field interactions.[1]

Algorithm 4.3.1 provides an additional source of error. We approximate entire collections of integrals if the error incurred by doing so is small (by some user defined parameter). However, we can improve on this algorithm. Past work on dual-tree

[1]The LinK method does not incur any additional error compared to the DirectSCF method since it only prunes the integrals that the direct method would have.

kernel summations [107] guarantees absolute or relative error in the final result.

We sketch how this can be done for absolute error Conceptually, we allow some total error in each entry of the computed Coulomb or exchange matrix, or equivalently, we allow the error for each bra shell pair. Then, in the prune check (line 4, we see if the total possible error in the integrals is less than the total error allocated to the bra shell pairs in nodes I and J. If it is, then we make the prune. We then deduct the maximum error we can have incurred from the error available to bra shell pairs.

It is not clear in Algorithm 4.3.1 how to efficiently obtain bounds on the density matrix. While it is possible to iterate through the density matrix to obtain these bounds whenever needed, this can be unnecessarily expensive. Furthermore, we cannot expect the density matrix structure to correspond to the spatial structure captured by the tree.

Note that both of these points, the enforcement of rigorous error guarantees and efficient storage and indexing of bounds on the density matrix, require us to handle quantities that depend on more than one shell. It is not clear how to efficiently store and sort such quantities.

4.3.2 Dual-Tree Algorithm

Inspired by our pairwise traversal algorithm in the n-point correlation context, we can take a more direct approach. We first call **FormSignificantShellPairs**. We then build a tree directly on shell pairs. We perform a dual tree algorithm on this tree.

Note that this algorithm is similar to the multi-tree algorithm. However, instead of a four-way recursion over trees built directly on shells from the basis, we do a two-way recursion on trees built on basis shells. This immediately lets us find many of the opportunities for pruning without the overhead of the four-way recursion.

As with the multi-tree algorithm, this algorithm can be used to compute both

Algorithm 4.3.2 DualTreeFockMatrix(ShellPair Tree Nodes Q, R)

```
    if $Q$ and $R$ are leaves then
        Compute all integrals directly and contract with density
    else
        Compute bounds $(Q|R)^{\min}, (Q|R)^{\max}$
5:      if $|(Q|R)^{\max}| < \epsilon$ then
            Prune
        else if $|(Q|R)^{\max} - (Q|R)^{\min}| < \epsilon$ then
            $(Q|R)^* = \left((Q|R)^{\max} + (Q|R)^{\min}\right)/2$
            Contract $(Q|R)^*$ with densities
10:     else
            Split largest node and recurse
        end if
    end if
```

matrices, or just one at a time. Furthermore, for Coulomb matrix calculations, we can store bounds on density matrix entries in the node R and use them in pruning and approximations.

4.3.3 Related Work as Multi-Tree Algorithms

We now turn to two seemingly unrelated algorithms from the literature – the CFMM [176] and LinK [127] methods for Coulomb and exchange matrix construction, respectively. We show that each of these fits within our dual- and multi-tree framework.

4.3.3.1 The Continuous Fast Multipole Method

As noted above, the most efficient methods for Coulomb matrix construction are based on generalizations of the Fast Multipole Method. If we view the bra and ket pairs as charge distributions, then the Coulomb matrix construction problem can be viewed as computing a Coulomb potential. The target points are bra pairs, and the source points are ket pairs. The charges on source points are given by the density matrix D. If the source and target distributions are sufficiently distant, they can be approximated by an interaction between point charges.

Since we are now dealing with charge distributions with infinite extent, some modifications to the standard FMM are required. The well-separatedness condition needs to be modified. In general, the accuracy of the FMM is controlled by fixing the distance at which an interaction can be considered far-field. This well-separated index, denoted WS, is 1 if nearest neighbors are considered far-field, two if next nearest neighbors are, and so on. For the standard FMM, a global value of WS is fixed before computation (generally 2).

For Gaussian basis sets, the charge distributions will themselves be Gaussians. Since Gaussians have infinite support, we must truncate them in order to have any far-field interactions. The CFMM uses the following definition of extent:

Definition 4.3.1. *Let A and B be Gaussian shells. Let α be the exponent of the contracted shell (obtained from the Gaussian Product Theorem. Then, for a given error tolerance ϵ, the extent of the charge distribution AB is*

$$r_{ext} = \frac{1}{2}\sqrt{\frac{2}{\alpha}\ln(\epsilon) - r_{AB}} \tag{4.3.3}$$

The error caused treating the interaction between two charge distributions as far field will increase as the extents of the distributions grow. We therefore define a well separated parameter for each distribution

$$WS = \max(2\lceil r_{\text{ext}}/l, WS_{\text{ref}}) \tag{4.3.4}$$

where l is the box size and WS_{ref} is the well-separated criterion chosen for point charges.

We then modify the octree data structure to account for differing values of WS for different distributions. Each box is divided into several separate nodes. Each node contains all distributions with a given WS index for the box. We can then treat interactions between nodes at a given level as near-field or far-field independently.

Algorithm 4.3.3 CFMM(Basis Set B, density D, error thresholds ϵ_1, ϵ_2)

```
    $T$ = **FormSignificantShellPairs**($B, \epsilon_1$)
    Build modified octree on centers of pairs in $T$
    Given a density $D$
    // Upward Pass
5:  Form multipole expansions at each leaf node
    Pass multipole expansions upward to form expansions for internal nodes
    // Downward Pass
    For each node, starting from the root, find well-separated nodes at the same level
    that have not been previously accounted for
    Translate multipoles to well-separated nodes to handle far-field interactions
10: At the leaves, account for any remaining near-field interactions directly
```

We can now describe the full CFMM algorithm. We begin by screening for significant charge distributions as before. We then build the modified octree on the charge distribution. We then perform the usual FMM by first building multipole expansions from the bottom of the tree up, then passing down the tree, performing translations to account for far-field interactions. We finally perform any remaining near field interactions directly.

The Fast Multipole Method is one of our prototypical dual-tree algorithms from Chapter 2. Clearly, the CFMM fits within the same framework. It again makes use of space-partitioning trees and prunes parts of the computation by approximating the interactions between pairs of tree nodes. Although it uses the fixed pattern of near- and far-field interactions common to the FMM, this can be viewed as a more efficient version of the recursive traversal pattern used in dual-tree algorithms.

4.3.3.2 The Linear K Algorithm

We now turn to the most efficient existing method for exchange matrix construction – the Linear K (LinK) method [127]. The Linear K method modifies the Direct SCF algorithm (Algorithm 4.2.2). In Direct SCF, we screen shell pairs individually, without sharing information between any parts of the computation. LinK sorts lists of possible integrals in decreasing order of the estimate Q_{ij}. By doing so, once we have determined that we have reached the point in the list where integrals are no longer significant, we can stop computation entirely.

The two key efficiency improvements come at lines 19 and 23. Essentially, the necessary integrals are prescreened and sorted by both the size of the bound and the size of the largest density entry to be contracted with the integral. The sorting allows one to leave the loop over shells early, since the lists are sorted in order of decreasing integral estimate.

We compute a bound on the maximum possible value Q_{ij} for each shell i:

$$((_{\max}|(_{\max})^{\frac{1}{2}}\, i) = \max_j\,(ij|ij)^{\frac{1}{2}} = \max_j Q_{ij} \tag{4.3.5}$$

This algorithm seems to be completely different from any dual- or multi-tree algorithms examined so far. However, we claim it resembles our pairwise traversal npcf algorithm (Algorithm 3.5.5). The first step in LinK is identifying the significant shell pairs. Although this step is neglected in previous analyses, the version given in Algorithm 4.2.1 clearly requires $O(N^2)$ work. We will show that this step can be performed in linear time using a dual-tree algorithm.

The major improvements in the LinK method come from sorting the lists of shell pairs so that the loops can be exited early. We claim that sorting is a one-dimensional analog of space-partitioning trees. The efficiencies in dual-tree algorithms come from the indexing (sorting) provided by the tree and the corresponding ability to prune. The LinK method simply precomputes all of the bounds and sorts (or indexes) by them. Thus, all the pruning is done at once, rather than piecemeal as it is encountered during the algorithm. A space partitioning tree that takes into account the integral and density bounds used by LinK may be able to replicate LinK's success in a more obviously dual-tree or multi-tree fashion.

Algorithm 4.3.4 LinK(Basis Set B, density D, error thresholds ϵ_1, ϵ_2)

```
$T$ = **FormSignificantShellPairs**($B, \epsilon_1$)
Create array of lists $S$
**for all** shells $i$ such that $(i, j) \in T$ for some $j$ **do**
  $S[i] = \emptyset$
  **for all** shells $k$ such that $(k, l) \in T$ for some $l$ **do**
    **if** $D_{ik} (i_{\max}|i_{\max})^{\frac{1}{2}} (k_{\max}|k_{\max})^{\frac{1}{2}} > \epsilon$ **then**
      Insert $k$ into $S[I]$
    **end if**
  **end for**
  Sort $S[i]$ by decreasing $D_{ik} (k_{\max}|k_{\max})^{\frac{1}{2}}$
**end for**
**for all** Shell pairs $(i, j) \in T$ **do**
  $ML_i = \emptyset; ML_j = \emptyset$
  **for all** Shells $k \in S[i]$ **do**
    **for all** Shells $l$ such that $(k, l) \in T$ **do**
      **if** $D_{ik}Q_{ij}Q_{kl} > \epsilon$ **then**
        Insert $(k, l)$ in $ML_i$
      **else**
        Leave loop over $l$
      **end if**
    **end for**
    **if** $ML_i = \emptyset$ **then**
      Leave loop over $k$
    **end if**
  **end for**
  Fill in $ML_j$ in the same fashion
  Merge $ML_i$ and $ML_j$ to form $ML$
  **for all** pairs $(k, l) \in ML$ **do**
    Compute $(ij|kl)$ and contract with $D_{ik}, D_{il}, D_{jk}, D_{jl}$
  **end for**
**end for**
```

4.4 Results

4.4.1 Theoretical Results

We now turn to an adaptive analysis of the Fock matrix construction problem and the algorithms we have discussed for its solution. As before, we require a more nuanced idea than simply the size of the data set, or in this case, the number of basis functions. An adversarial distribution of basis functions can easily cause any of the algorithms discussed above to take the full $O(N^4)$ time of the naive algorithm.

To see this, consider a distribution in which the basis functions all share a common center. In this case, every shell pair is significant. Furthermore, no integrals can be pruned through Schwarz criteria, the LinK pruning steps, or treated as far-field interactions in the CFMM. Therefore, all of these algorithms will require $O(N^4)$ time.

As before, we can take the distribution of the data into account. In the npcf setting, we were able to prove results about uniform distributions. In the HF setting, we can also restrict the possible input distributions. Our distributions of basis functions will consist of functions centered on atoms of real molecules. This immediately gives us several powerful conditions.

Most importantly for our analysis, we assume that atomic nuclei are not arbitrarily close. We assume that any two nuclei are separated by a distance at least b. This assumption is physically motivated, since in chemical processes, nuclei will remain separated by the repulsive forces between them.

In the npcf setting, as N grows, we differentiated between the case where the sample window grows as well and where the density increases. Here, we have a similar condition. The number of basis functions can grow for two reasons: either the number of atoms grows, or the number of basis functions on each nucleus grows. Once again, our analysis will distinguish between these possibilities.

Our analyses make use of the following notations:

- N – The total number of basis functions.

- b – The minimum distance between nuclei.
- c – The maximum number of basis functions on a single nucleus.
- A – The total number of atoms.
- $F^{\max}$ – In Equation 4.1.17, we showed that the two electron integral can be decomposed into exponential terms and a term over incomplete gamma functions, denoted $\sum_m C_m F_m(r)$. We will use the term $F^{\max} = \max_{i,j} \sum_m C_m F_m(0)$ where the summation comes from integrals of the form $(ij|ij)$.

We can immediately note the following inequality from these definitions:

$$N \leq A \cdot c \tag{4.4.1}$$

Our analyses bound runtimes in terms of A and c, thus allowing us to highlight the different scaling behavior that occurs as the basis set becomes more complete and as the molecule grows but the basis set remains fixed.

We also take into account the number of integrals the algorithms actually need to compute.

4.4.1.1 Preliminaries

Lemma 4.4.1. *There are at most $O(c(r/b)^3)$ basis shells within a radius r of a given function.*

Proof. If there is a basis shell at a given point, then there are at most $c-1$ other shells at that point and no other shells within a ball of radius b.

We can therefore bound the number of functions within a distance r of a given function by bounding the number of balls of radius b that can be packed within a ball of radius of $r + b/2$. Therefore, the maximum number of spheres that can be packed is:

$$S \leq \left(\frac{r}{b} + 1\right)^3 \tag{4.4.2}$$

The proof follows from the fact that there are at most c functions at the center of each of these spheres. □

Lemma 4.4.2. *There are at most $O(A \cdot c^2)$ significant shell pairs.*

Proof. We fix a single basis shell, and count the number of possible shells that can be paired with it to form a significant shell pair. Recall that we prune a shell pair (i, j) if

$$(ij|ij) < \epsilon \tag{4.4.3}$$

We wish to solve this inequality for a radius r^* beyond which all pairs will be pruned. We fix shell i at the origin (without loss of generality) and let r be the distance to the center of shell j.

Then, using a standard shell quartet decomposition [22, 47]

$$(ij|ij) = 2\pi^{\frac{5}{2}} \exp\left(-\alpha^2 r^4\right) \sum_{m=0}^{L} F_m(0) \tag{4.4.4}$$

Where

$$\alpha = \frac{\alpha_i \alpha_j}{\alpha_i + \alpha_j} \tag{4.4.5}$$

and $L = 2m_i + 2m_j$. The terms F_m are all evaluated at zero because the bra and ket pairs are identical.

Solving for r, we have

$$r^* = \frac{\left[-\log\left(\frac{\epsilon}{2\pi^{\frac{5}{2}} F^{\max}}\right)\right]^{\frac{1}{4}}}{\alpha^{\frac{1}{2}}} \tag{4.4.6}$$

where $F^{\max}$ is defined above.

Given this r^*, we can bound the number of other shells that lie within it. Since we have assumed the minimum distance between atoms to be bounded from below, we have that the maximum number of shells S within distance r^* of a fixed shell is bounded by the number of balls of radius b that can be packed into a ball of radius r^*.

$$S \leq c \cdot \left(\frac{r^*}{b}\right)^3 \tag{4.4.7}$$

Therefore, the total number of shell pairs T is bounded by

$$\begin{aligned} T &\leq N \cdot \left(\frac{r^*}{b}\right)^3 && (4.4.8) \\ &\leq A \cdot c^2 \cdot b^{-3} \cdot O\left(\left[-\log\left(\frac{\epsilon}{F^{\max}}\right)\right]^{\frac{3}{4}}\right) && (4.4.9) \end{aligned}$$

□

Lemma 4.4.3. *The list of significant shell pairs can be computed in time proportional to the number of pairs, or $O(A \cdot c^2)$ time.*

Proof. (Sketch.) The **FormSignificantShellPairs** algorithm (Alg. 4.2.1) clearly runs in $O(N^2)$ time since it loops over all pairs of shells. However, a dual-tree algorithm can do it in $O(N)$ time.

Our goal is to construct a list of all pairs satisfying the integral cutoff criterion. This list is contained in a list of all shell pairs within the distance r^* given above.

Constructing this list is similar to the construction of the interaction list for npcf estimation described in Chapter 3. We can apply a similar dual-tree algorithm, and the analysis closely follows that of the previous algorithm. □

Note that both of these results exhibit different scaling behavior based on what we mean by increasing N. If the number of atoms grows while the basis used is fixed, then we see linear growth in the number of shell pairs. However, as we move to a more complete basis, we see quadratic growth, since this increases the number of shells per atom c.

Definition 4.4.4. ***Localized Density Assumption.*** *Let r be the distance between the centers of shells i and j. We say the density is* localized *if there exist constants c', d, and $\hat{r}$ such that for all i and j such that $r > \hat{r}$,*

$$D_{ij} < \frac{c'}{r^d} \qquad (4.4.10)$$

Previous work on linear scaling methods for exchange matrix construction generally makes an informal assumption on the structure of the density matrix. For insulating systems, the density is expected to decay exponentially with distance. On the other hand, for conductors, the density may be non-localized even on large scales. Our proofs below show that a weaker assumption than exponential decay is sufficient.

With these preliminary results, we can turn to the main contributions of this section.

4.4.1.2 Runtime Theorems

Theorem 4.4.5. *The DirectSCF algorithm (Algorithm 4.2.2) runs in at most $O(A^2 \cdot c^4)$ time.*

Proof. The computation of the significant shell pairs takes $O(A \cdot c^2)$ time and finds the same number of pairs by lemma 4.4.3. After this step, the algorithm consists of two nested loops over all shell pairs. Since there are $O(A^2 \cdot c^4)$ such pairs, the entire algorithm takes $O(A^2 \cdot c^4)$ time. □

Theorem 4.4.6. *The LinK algorithm (Alg. 4.3.4) runs in at most $O(A^2 \cdot c^4 \cdot b^{-6} \cdot |\log(\epsilon)^{\frac{3}{2}}|)$ time and computes integrals of at most $O\left(A \cdot c^3 \cdot b^{-6} \cdot \epsilon^{-\frac{3}{d}}\right)$ shell quartets under the assumption that $D(r) \leq c' r^{-d}$ for some constants c' and d and r large enough.*

Proof. We assume throughout that there are T significant shell pairs, where T is bounded by Lemma 4.4.2.

We first construct the list of significant shell pairs. Note that the max integrals $(i_{\max}|i_{\max})^{\frac{1}{2}}$ can be computed while forming the list of significant shell pairs at no additional asymptotic cost.

The next step constructs the lists $S[i]$ for each shell i. A shell is placed in this list if both integral bounds $(i_{\max}|i_{\max})^{\frac{1}{2}}$ and $(k_{\max}|k_{\max})^{\frac{1}{2}}$ are large and if the density matrix entries D_{ik} are large. Assuming that all the $(i_{\max}|i_{\max})^{\frac{1}{2}}$ are bounded from above by some constant, then, using our assumption on the structure of the density matrix, we have that the number of shells in list $S[i]$ is bounded by the number of shells that can be packed in a radius r':

$$r' \leq \left(\frac{\epsilon}{c' \left(i_{\max} | i_{\max} \right)^{\frac{1}{2}} \max_k \left(k_{\max} | k_{\max} \right)^{\frac{1}{2}}} \right)^{-\frac{1}{d}} \tag{4.4.11}$$

Therefore, the size of the list is bounded by

$$|S[i]| \leq cb^{-3} \left(\frac{\epsilon}{c' \left(i_{\max} | i_{\max} \right)^{\frac{1}{2}} \max_k \left(k_{\max} | k_{\max} \right)^{\frac{1}{2}}} \right)^{-\frac{3}{d}} \tag{4.4.12}$$

and sorting this list takes $O(|S[i]| \log |S[i]|)$ time. However, forming the lists $S[i]$ for all shells i requires a double loop over the list of significant shell pairs, which takes $O(T^2)$ time.

The main portion of the algorithm consists of a triply-nested loop over all shell pairs (i, j), all shells k in the list $S[i]$, and over all shells l such that (k, l) is a significant pair. This requires $O(T \cdot |S[i]| \cdot T) \leq O(T^2 \max_i |S[i]|)$ work, if all the loops run to the end. However, the loops all exit once we encounter an integral that is small enough to be neglected.

For a fixed i, the the number of possible k is bounded by Eqn. 4.4.12. Then, the number of l that are in a shell pair with k is bounded by Eqn. 4.4.7. Therefore, we have an $O(T)$ bound on the number of integrals we actually have to compute. Therefore, the total number of integrals I is bounded by:

$$I \leq O\left(A \cdot c^3 \cdot b^{-6} \cdot \epsilon^{-\frac{3}{d}} \right) \tag{4.4.13}$$

The runtime of the second part is determined by the number of integrals computed, since the loops exit early when they encounter an integral that does not need to be computed.

The runtime stated in the theorem follows from the fact that the loops to construct $S[i]$ dominate the total runtime with their $O(T^2)$ time. $\square$

So, we have an algorithm which formally runs in quadratic time in the size of the molecule, but which only computes linearly many integrals under the assumption of localized density. However, the time spent computing integrals easily dominates the total computation time on most molecules of interest. In fact, the actual crossover point may be difficult to observe with current hardware. This probably explains why the runtime is generally observed to be linear in empirical studies.

We now turn to runtimes for our new algorithms. For simplicity, we only address the runtime of our dual-tree algorithm.

Theorem 4.4.7. *If the bounds $(Q|R)^{\min}$ and $(Q|R)^{\max}$ can be computed in constant time for given nodes Q and R, then the dual-tree Fock matrix construction algorithm (Algorithm 4.3.2) runs in at most $O(A \cdot c^3 + A \cdot c^2 \log(A \cdot c^2))$ time and computes $O(A \cdot c^3 \cdot b^{-6} \epsilon^{-3})$ integrals using an octree built on the centers of shell pairs.*

Proof. We first construct the list of significant shell pairs, in $O(T)$ time and containing T pairs using lemma 4.4.3. Tree construction requires $O(T \log T)$ time.

Note that an integral of the form $(q|r)$ for some shell pairs q and r decreases as r^{-1} where r is the distance between the centers of the shell pairs. Therefore, for a given shell pair q, we will only compute integrals for shell pairs within some distance of q that depends on ϵ.

We have that

$$(a|b) = 2\pi^{\frac{5}{2}} e^{-\alpha_a r_a^2} e^{-\alpha_b r_b^2} \sum_m F_m(r) \tag{4.4.14}$$

where

$$\left| \sum_m F_m(r) \right| \leq \frac{f}{r} \tag{4.4.15}$$

for some constant f and large r [47].

From, this we can see that we can prune any shell-pairs separated by a distance greater than

$$r^* = \frac{f}{\epsilon} \left(2\pi^{\frac{5}{2}} \max_a \left\{ e^{-\alpha_a r_a^2} \right\} \max_b \left\{ e^{-\alpha_b r_b^2} \right\} \right) = \frac{2\pi^{\frac{5}{2}} f}{\epsilon} \tag{4.4.16}$$

Using the same reasoning as Equation 4.4.7 above, we see that the maximum number of integrals I that need to be computed is

$$\begin{aligned} I &\leq T \cdot c \cdot \left(\frac{r^*}{b}\right)^3 && (4.4.17) \\ &\leq T \cdot c \cdot O(b^{-3}\epsilon^{-3}) && (4.4.18) \end{aligned}$$

Combining this with the number of significant shell pairs from Lemma 4.4.2, we have that

$$I \leq O\left(A \cdot c^3 \cdot b^{-6}\epsilon^{-3}\right) \qquad (4.4.19)$$

Any nodes separated by a distance greater than r^* will be pruned in line 5 of Algorithm 4.3.2. Therefore, we can apply the same reasoning as lemma 3.8.2 in Chapter 3. As before, we only need to consider pairs of tree nodes within a fixed distance which is independent of the problem size. This distance is slightly larger than the distance r^*, so there are only a constant factor more bases case pairs to consider than the number of integrals to compute. Therefore, we perform work proportional to the number of integrals in the base case.

At each level of the tree, we prune any nodes that are more distant than r^*. We can bound the number of nodes to be considered at each level, from the root to the leaves. As before, this is proportional to the work in the base case. □

Note that the proof does not depend on the ability to prune batches of integrals in line 7. The key to potential speedups of this algorithm over existing methods lies in how well we can take advantage of this opportunity. This in turn depends on the quality of our bounds, the details of the basis set, and the structure of the molecule. We return to these factors in our discussion of the empirical results involving this molecule.

Furthermore, the LinK bound is tighter than our dual-tree bound by a factor of ϵ^{-d}, since LinK is able to take advantage of the decay in the density matrix.

Table 6: Results for exchange matrix construction algorithms. All error tolerances are set to 10^{-8} Errors are relative to direct screening with error cutoff 10^{-10}.

Molecule	Algorithm	Time (s)	Error (μhartrees)
Amylose	LinK	2407.68	2.86
	Dual-Tree	2206.91	2.58
Carbon Nanotube	LinK	2397.03	3.27
	Dual-Tree	2268.46	2.94

4.4.2 Preliminary Empirical Results

We turn now to a brief overview of some preliminary results for our multi-tree Fock matrix construction algorithms. Here, we focus on the construction of the exchange matrix as potentially lower hanging fruit. The FMM is very powerful and efficient, so we suspect it may be harder to outperform. Also, the Coulomb matrix in general has been studied more – it appears in DFT as well.

We show results for our implementation of the LinK algorithm (Alg. 4.3.4). We also show results for our dual-tree algorithm (Alg. 4.3.2), specialized for exchange matrix computation. We use the QQR bounds [116] described in Equation 4.1.20. We implement all algorithms in a development version of the Psi4 computational chemistry package [169].

We show a few preliminary results on our implementations in Table 6. We examine two molecules, amylose and a carbon nanotube. The calculations were done in a 6-31G* basis. Amylose consists of 45 atoms and 182 shells. The carbon nanotube has 30 atoms and 140 shells. The structure files were those used in evaluations of the QQR bounds [116].

The preliminary results in Table 6 show very small speedups over the LinK method. We highlight a very few results Clearly, these results are not in line with the massive empirical speedups obtained from dual and multi-tree algorithms for other problems, such as those discussed in Chapters 2 and 3. There are several possible reasons for this.

First, these molecules are relatively small. All multi-tree algorithms will have some crossover point, below which the direct iterative approach will be more efficient. While we do obtain some small improvements here, it is possible that these will grow with larger molecules.

The second possible reason is related to the first. The integral code currently used in Psi4 is not very efficient. While this should produce larger speedups sooner (by making base cases more expensive relative to pruning checks), it does severely limit the overall size of systems that we can consider. This may prevent us from getting a clear look at the actual performance of the algorithms in the large system limit.

The third possibility is that the extra complications in our dual- and multi-tree algorithms are simply not worth the extra expense. The LinK method is simple and efficient. Note that it is similar to our two-stage traversal algorithm for raw correlation counts in Chapter 3.

One significant feature of our limited experiments is the overall lack of the approximations made possible in line 7 of Algorithm 4.3.2. This may be due to insufficiently tight bounds. Another possibility is that the molecules we are able to experiment on are too small to allow for efficient pruning. Determining the answers to all of these questions will require more efficient integral code to be able to explore larger molecules.

CHAPTER V

CONCLUSION

We conclude this dissertation by discussing some overall lessons for multi-tree algorithms, along width directions for future work. We recap our central thesis and the evidence we have provided in support of it.

5.1 Practical Lessons for Multi-Tree Algorithm Development

This thesis explores the application of the principles of dual-tree algorithms to higher-order problems through efficient implementations on problems of real scientific interest. We summarize a few of the general lessons learned through this process.

Focus on the base case. In any future dual- or multi-tree algorithm, we believe that the algorithm for base case computations will be more important than the choice of space-partitioning tree or efficiency improvements in the tree traversal. Even for very simple base cases, such as nearest neighbor or the npcf, the base case accounts for the majority of the execution time.

Both our work on n-point correlation function estimation and Fock matrix construction emphasized the importance of efficient evaluation of the base cases. By optimizing the base case computation for 3-point correlations, we were able to show up to an order of magnitude speedup over an unoptimized version of our algorithm.

Furthermore, our efforts to evaluate efficient exchange matrix construction algorithms were hampered by the relative inefficiency of integral evaluation. Without fast integrals, we were unable to perform Hartree-Fock calculations on truly large molecules, thus limiting the opportunities for pruning in our new multi-tree algorithms and keeping us from truly understanding the crossover points of these algorithms.

Take on the whole problem. We were able to achieve significant speedups with our multi-matcher and efficient resampling npcf estimation algorithms by taking on the whole problem, rather than just the part most amenable to a dual- or multi- tree algorithm. In particular, it pays to look for any opportunities to share work between different parts of the computation.

This approach led to the reformulation of the jackknife sampling task for npcf estimation, previously done as an outer loop around a multi-tree algorithm, into something done more directly. We also discuss further possibilities opened by this approach in the context of Fock matrix construction.s

Try to find a fixed recursion pattern. A smaller or fixed recursion pattern is a recurring theme in the algorithms explored. We achieved significant speedups from our pairwise tree traversal in the npcf context. This improvement comes from two main sources. first, it allowed us to both cut down on the time spent identifying work for our optimized base case code. Second, we were able to better take advantage of the throughput offered by the bitwise interpretation of the matcher results.

Alternative recursion patterns appear in other tree-based methods. The CFMM reduces a problem based on four-tuples to one based on pairs, then further improves by using the fixed expansion pattern of the FMM. LinK avoids explicit recursion entirely as well by identifying and sorting all potential sources of work in advance.

5.2 Future Directions

We briefly discuss a few remaining loose ends from this dissertation and future directions in which this work can move. This is only a partial list, some items are withheld so I can publish them later without getting scooped.

5.2.1 A Formal Theory of Multi-Tree Algorithms

Our exploration of multi-tree algorithm suggests that they can provide massive speedups, tight theoretical runtime guarantees, and a unifying framework. While this work is mostly exploratory, the next logical step is a more rigorous treatment of multi-tree algorithms. Ongoing work with Ryan Riegel and others concerns a formal theory of multi-tree algorithms and the problems they solve. This work centers around the definition of a class of *generalized N-body problems*, and the decomposition of multi-tree algorithms into an abstract algorithmic framework with particular instantiations of pruning rules and base cases. A first step in this latter direction was undertaken in the context of dual-tree algorithms [40].

5.2.2 Multi-Tree Algorithms for Higher-Order Correlation Theories

The Hartree-Fock wavefunction is often used as the starting point for other methods which treat correlations between electrons. These methods, such as configuration interaction, coupled cluster theories, and various perturbation theories compute a linear combination of the HF ground state wavefunction and higher energy solutions. The bottleneck computations in these methods consist of tensor contractions – multi-index summations – over distance-dependent functions. This suggests that multi-tree algorithms may be a natural fit for these problems. In the past, the narrow focus of FMM-based algorithms (for Coulomb matrix construction) and sorting methods (for exchange matrix construction) prevented their being generalized to other problems. By providing a unifying framework for these methods, we have opened the possibility of applying techniques learned from Fock matrix construction to more accurate and computationally-demanding methods.

5.2.3 Omitting Fock Matrix Construction Entirely

One of the lessons about multi-treee algorithms mentioned above is to "take on the whole problem." Our formulation of the Fock matrix construction problem does not do this. The Fock matrix itself is not the object of interest. Instead, we diagonalize this matrix to obtain its spectrum.

Instead of approaching this task directly, ours and other previous work have simply tried to efficiently approximate the Fock matrix itself. Here, the measure of accuracy used for approximations is simply the amount of error allowed in individual entries of the matrix (or contributions to it). Instead, it may be beneficial to approximate its spectrum directly. The spectral decomposition of a matrix is generally accomplished through repeated matrix-vector multiplications, which are themselves simply summations. This entire problem may be suitable for the development of multi-tree algorithms.

5.2.4 Generalizing Applications of N-Point Correlation Functions

Our correlation function work focuses exclusively on computing well-known estimators of the npcf. As such, it is purely a computational problem, with no statistical aspect to our algorithms. This work fits with the primary motivation for our work – the use of the npcf in astronomy and cosmology. Cosmological models can be used to derive concrete predictions for the n-point correlations of real data. Therefore, evaluating the models is as simple as estimating the npcf of data and comparing it to the predictions of the models.

However, the n-point correlations are broadly applicable to any point processes. Unfortunately, in other applications, it is not always clear how to use them. Furthermore, they are limited as an exploratory data analysis tool by their combinatorial growth with n and the number of configurations considered.

One approach, used in materials science applications in particular [90], is to compute the npcf for some set of configurations, then apply a dimensionality reduction technique such as PCA to allow one to visualize the results. As in the chemistry context above, we are computing a full matrix, then extracting its spectrum. It may again be possible to combine these two steps into a single, more efficient multi-tree algorithm.

5.2.5 Runtime Analysis of N-Point Computation Using Correlation Integrals

Our adaptive analyses of our npcf estimation algorithms were restricted to Poisson data. Although this is a very important special case, we would like to generalize to arbitrary point processes. In the analysis, the Poisson assumption was necessary to bound the number of points in a given volume. However, this bound can be obtained for a general distribution using the n-point correlation functions.

Preliminary work here suggests that the final bounds obtained will depend on integrals over the entire hierarchy of correlation functions. However, it may be possible to restrict this analysis to a bound that depends on only the correlations being estimated. This could lead to a useful bound which describes the runtime as a time spent on a Poisson set, plus a perturbation which varies with the size of the departure from uniformity.

5.2.6 Subsampling Based Methods

In the past, dual-tree algorithms have been successfully combined with Monte Carlo methods for truly massive speedups [84, 83]. By subsampling, rather than fully computing each base case, this combination allows dual-tree algorithms which run in time independent of the total size of the data. In principle, this approach can be combined with multi-tree algorithms. By incorporating the error bounds from the Monte Carlo method into the error estimation obtained from the jackknife, we believe that npcf estimation is a natural candidate for this approach.

5.2.7 Better Basis Sets through Machine Learning

One of the difficulties in bounding integrals in Fock matrix construction is the different properties of the basis functions. A range of momenta and exponents, along with contracted functions, make accurate and tight bounds for groups of functions difficult to obtain. Basis sets are chosen for their accuracy in approximating true wavefunctions with the fewest functions possible. This size objective is in turn driven by the severe constraints of $O(N^2)$-sized matrices and quartic scaling Fock matrix construction.

In the statistics literature, kernel density estimation using a basis of Gaussians has been used to approximate any distribution to any desired accuracy. In principle, the same could be done for the HF wavefunction, given a large enough basis set. This approach seems undesirable, since N may be much larger in this case. However, if the benefits of improved multi-tree algorithms are substantial enough, we may be able to obtain more accurate solutions at reduced computational cost.

5.3 Recap of this Thesis

Our central thesis is that multi-tree algorithms can provide orders-of-magnitude speedups on fundamental problems while providing a unifying framework for understanding these problems and bounding the runtimes of their solutions. We have supported this thesis through two main applications: the estimation of n-point correlation functions and the construction of the Fock matrix in Hartree-Fock theory.

For the n-point correlation estimation problem, a combination of new multi-tree algorithms, a comprehensive approach to the entire problem, and optimized base case and distributed implementations led to up to three orders-of-magnitude speedup over the previous state-of-the-art. Our work has enabled much larger 3-point correlation analyses than were previously possible. We also show detailed runtime analyses for important special cases of our algorithm, and suggest methods to obtain more general bounds.

The Fock matrix construction problem proved to be more difficult. Although our explorations of multi-tree algorithms did not immediately deliver significant speedups, we were able to unify previously disparate methods under our framework. Furthermore, our theoretical analysis provides the first detailed look at the different factors in the performance of existing algorithms.

Our exploratory efforts do provide the promised speedups in some cases. Furthermore, our efforts do provide a unifying framework, both for previous and new methods in both problem contexts, and to the overall types of problems that are amenable to solution through multi-tree algorithms. Both these results point to the promise of future applications of multi-tree algorithms.

REFERENCES

[1] Adelman-McCarthy, J. K., Agüeros, M. A., Allam, S. S., Anderson, K. S., Anderson, S. F., Annis, J., Bahcall, N. A., Baldry, I. K., Barentine, J., Berlind, A., and others, "The fourth data release of the sloan digital sky survey," *The Astrophysical Journal Supplement Series*, vol. 162, no. 1, p. 38, 2006.

[2] Adelman-McCarthy, J. K., Agüeros, M. A., Allam, S. S., Prieto, C. A., Anderson, K. S., Anderson, S. F., Annis, J., Bahcall, N. A., Bailer-Jones, C., Baldry, I. K., and others, "The sixth data release of the Sloan Digital Sky Survey," *The Astrophysical Journal Supplement Series*, vol. 175, no. 2, p. 297, 2008.

[3] Agarwal, P. K., Edelsbrunner, H., Schwarzkopf, O., and Welzl, E., "Euclidean minimum spanning trees and bichromatic closest pairs," *Discrete & Computational Geometry*, vol. 6, no. 1, pp. 407–422, 1991.

[4] Alexander, K., "The RSW theorem for continuum percolation and the CLT for Euclidean minimal spanning trees," *Ann. Appl. Probab.*, vol. 6, no. 2, pp. 466–494, 1996.

[5] Almlöf, J., Faegri, K., and Korsell, K., "Principles for a direct scf approach to licao–mo ab-initio calculations," *Journal of Computational Chemistry*, vol. 3, no. 3, pp. 385–399, 1982.

[6] Aluru, S., Prabhu, G. M., and Gustafson, J., "Truly distribution-independent algorithms for the N-body problem," in *Proceedings of the 1994 conference on Supercomputing*, pp. 420–428, IEEE Computer Society Press Los Alamitos, CA, USA, 1994.

[7] Anderberg, M. R., *Cluster analysis for applications.* Probability and Mathematical Statistics, New York: Academic Press, 1973, 1973.

[8] Appel, A., "An efficient program for many-body simulation," *SIAM Journal on Scientific and Statistical Computing*, vol. 6, p. 85, 1985.

[9] Babu, G. J. and Feigelson, E. D., *Astrostatistics.* Chapman & Hall/CRC, 1996.

[10] Balcan, M., Blum, A., and Vempala, S., "A discriminative framework for clustering via similarity functions," in *Symposium on Theory of Computing*, pp. 671–680, ACM, 2008.

[11] Banfield, Jeffrey D. and Raftery, Adrian E., "Model-based gaussian and non-gaussian clustering," *Biometrics*, vol. 49, pp. 803–821, sep 1993.

[12] Bardeen, J. M., Bond, J., Kaiser, N., and Szalay, A., "The statistics of peaks of Gaussian random fields," *The Astrophysical Journal*, vol. 304, pp. 15–61, 1986.

[13] Barnes, J. and Hut, P., "A Hierarchical $O(N \log N)$ Force-Calculation Algorithm," *Nature*, vol. 324, 1986.

[14] Barrow, J. D., Bhavsar, S. P., and Sonoda, D. H., "Minimal spanning trees, filaments and galaxy clustering," *MNRAS*, vol. 216, pp. 17–35, Sept. 1985.

[15] Beeler, M., Gosper, R., and Schroeppel, R., "Hakmem," 1972.

[16] Bentley, J. and Friedman, J., "Fast Algorithms for Constructing Minimal Spanning Trees in Coordinate Spaces," *IEEE T. Comput.*, vol. 27, pp. 97–105, 1978.

[17] Bentley, J. and Yao, A., "An almost optimal algorithm for unbounded searching," *Inform. Process. Lett.*, vol. 5, no. 3, pp. 82–87, 1976.

[18] Berry, B. and Marble, D., *Spatial analysis: a reader in statistical geography.* Prentice-Hall, 1968.

[19] Beygelzimer, A., Kakade, S., and Langford, J., "Cover trees for Nearest Neighbor," 2006. http://hunch.net/~jl/projects/cover_tree/paper/paper.ps.

[20] Beygelzimer, A., Kakade, S., and Langford, J., "Cover Trees for Nearest Neighbor," *Proceedings of the 23rd International Conference on Machine learning*, pp. 97–104, 2006.

[21] Bhavsar, S. P. and Splinter, R. J., "The superiority of the minimal spanning tree in percolation analyses of cosmological data sets," *MNRAS*, vol. 282, pp. 1461–1466, 1996.

[22] Boys, S. F., "Electronic wave functions. i. a general method of calculation for the stationary states of any molecular system," *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, vol. 200, no. 1063, pp. 542–554, 1950.

[23] Burant, J., Scuseria, G., and Frisch, M., "A linear scaling method for Hartree–Fock exchange calculations of large molecules," *The Journal of chemical physics*, vol. 105, p. 8969, 1996.

[24] Callahan, P. B. and Kosaraju, S. R., "A Decomposition of Multidimensional Point Sets with Applications to k-Nearest-Neighbors and n-body Potential Fields," *J. ACM*, vol. 62, no. 1, pp. 67–90, 1995.

[25] Callahan, P. B., *Dealing with Higher Dimensions: The Well-Separated Pair Decomposition and Its Applications.* PhD thesis, The Johns Hopkins University, 1995.

[26] Callahan, P. and Kosaraju, S., "Faster algorithms for some geometric graph problems in higher dimensions," in *Fourth annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 291–300, 1993.

[27] Callahan, P. and Kosaraju, S., "A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields," *J. ACM*, vol. 42, no. 1, pp. 67–90, 1995.

[28] Castro, R., Coates, M., and Nowak, R., "Likelihood based hierarchical clustering," in *IEEE Transactions on Signal Processing*, vol. 52, pp. 2308–2321, 2004.

[29] Challacombe, M., Schwegler, E., and Almlöf, J., "Fast assembly of the coulomb matrix: A quantum chemical tree code," *The Journal of chemical physics*, vol. 104, p. 4685, 1996.

[30] Chazelle, B., "A faster deterministic algorithm for minimum spanning trees," in *Symposium on Foundations of Computer Science*, pp. 22–31, 1997.

[31] Chazelle, B., "A minimum spanning tree algorithm with inverse-ackermann type complexity," *J. ACM*, vol. 47, no. 6, pp. 1028–1047, 2000.

[32] Clarkson, K., "Fast Algorithms for the All Nearest Neighbors Problem," in *Proceedings of the Twenty-fourth Annual IEEE Symposium on the Foundations of Computer Science*, pp. 226–232, 1983.

[33] Colless, M., Dalton, G., Maddox, S., Sutherland, W., Norberg, P., Cole, S., Bland-Hawthorn, J., Bridges, T., Cannon, R., Collins, C., and others, "The 2df galaxy redshift survey: spectra and redshifts," *Monthly Notices of the Royal Astronomical Society*, vol. 328, no. 4, pp. 1039–1063, 2001.

[34] Cooper, L. and others, "Two-point correlation as a feature for histology images," in *Computer Vision and Pattern Recognition Workshops*, vol. 13, pp. 79–86, 2010.

[35] Cormen, T. T., Leiserson, C. E., and Rivest, R. L., *Introduction to algorithms.* Cambridge, MA, USA: MIT Press, 1990.

[36] Crawford, T. D., Sherrill, C. D., Valeev, E. F., Fermann, J. T., King, R. A., Leininger, M. L., Brown, S. T., Janssen, C. L., Seidl, E. T., Kenny, J. P., and others, "Psi3: An open-source ab initio electronic structure package," *Journal of computational chemistry*, vol. 28, no. 9, pp. 1610–1616, 2007.

[37] Cressie, N., *Statistics for spatial data.* John Wiley & Sons, 1991.

[38] Cuevas, A., Febrero, M., and Fraiman, R., "Estimating the number of clusters," *The Canadian Journal of Statistics/La Revue Canadienne de Statistique*, vol. 28, no. 2, pp. 367–382, 2000.

[39] Curtin, R. R., Cline, J. R., Slagle, N. P., March, W. B., Ram, P., Mehta, N. A., and Gray, A. G., "Mlpack: A scalable c++ machine learning library," *Journal of Machine Learning Research*, vol. 14, pp. 801–805, 2013.

[40] Curtin, R. R., March, W. B., Ram, P., Anderson, D. V., Gray, A. G., and Isbell Jr., C. L., "Tree independent dual-tree algorithms," in *International Conference on Machine Learning*, 2013.

[41] Davis, J., "Statistics and Data Analysis in Geology," 1986.

[42] Day, W. H. E. and Edelsbrunner, H., "Efficient algorithms for agglomerative hierarchical clustering methods," *Journal of Classification*, vol. 1, no. 1, pp. 7–24, 1984.

[43] Demaine, E., López-Ortiz, A., and Munro, J., "Adaptive set intersections, unions, and differences," in *SODA*, pp. 743–752, 2000.

[44] Deng, K. and Moore, A. W., "Multiresolution Instance-Based Learning," *International Joint Conference on Artificial Intelligence*, vol. 14, pp. 1233–1242, 1995.

[45] Dewdney, P., Hall, P., Schilizzi, R., and Lazio, T., "The square kilometre array," *Proceedings of the IEEE*, vol. 97, no. 8, pp. 1482–1496, 2009.

[46] Duda, R. O., Hart, P. E., and Stork, D. G., *Pattern Classification*. New York: John Wiley & Sons, Inc., second ed., 2001.

[47] Dupuis, M., Rys, J., and King, H. F., "Evaluation of molecular integrals over gaussian basis functions," *The Journal of Chemical Physics*, vol. 65, p. 111, 1976.

[48] Dyczmons, V., "No n 4-dependence in the calculation of large molecules," *Theoretica chimica acta*, vol. 28, no. 3, pp. 307–310, 1973.

[49] Eisen, M. B., Spellman, P. T., Brown, P. O., and Botstein, D., "Cluster analysis and display of genome-wide expression patterns," *Proceedings of the National Academy of Sciences*, vol. 95, no. 25, pp. 14863–14868, 1998.

[50] Eppstein, D., "Spanning trees and spanners," in *Handbook of Computational Geometry* (Sack, J. R. and Urrutia, J., eds.), pp. 425–461, North-Holland,Amsterdam: Elsevier Science Publishers B.V., 1999.

[51] Eppstein, D., "Fast hierarchical clustering and other applications of dynamic closest pairs," *J. Exp. Algorithmics*, vol. 5, p. 1, 2000.

[52] Eppstein, D. and Erickson, J., "Raising roofs, crashing cycles, and playing pool: applications of a data structure for finding pairwise interactions," in *SCG '98: Proceedings of the fourteenth annual symposium on Computational geometry*, (New York, NY, USA), pp. 58–67, ACM Press, 1998.

[53] Esselink, K., "The order of appel's algorithm," *Information Processing Letters*, vol. 41, no. 3, pp. 141–147, 1992.

[54] Estivill-Castro, V. and Wood, D., "A survey of adaptive sorting algorithms," *ACM Comput. Surv.*, vol. 24, no. 4, pp. 441–476, 1992.

[55] Fasulo, D., "An analysis of recent work on clustering algorithms," 1999.

[56] Fermann, J. and Valeev, E., "Libint: Machine-Generated Library for Efficient Evaluation of Molecular Integrals over Gaussians," 2003.

[57] Fisher, D., "Iterative optimization and simplification of hierarchical clusterings," *Journal of Artificial Intelligence Research*, vol. 4, pp. 147–180, 1996.

[58] Fredman, M. and Tarjan, R., "Fibonacci heaps and their uses in improved network optimization algorithms," *J. ACM*, vol. 34, no. 3, pp. 596–615, 1987.

[59] Friedman, J. H., Bentley, J. L., and Finkel, R. A., "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematical Software (TOMS)*, vol. 3, no. 3, pp. 209–226, 1977.

[60] Fry, J., "The galaxy correlation hierarchy in perturbation theory," *The Astrophysical Journal*, vol. 279, pp. 499–510, 1984.

[61] Fry, J., "Galaxy N-point correlation functions – Theoretical amplitudes for arbitrary N," *The Astrophysical Journal*, vol. 277, pp. L5–L8, 1984.

[62] Fry, J. and Peebles, P., "Statistical analysis of catalogs of extragalactic objects. IX-The four-point galaxy correlation function," *The Astrophysical Journal*, vol. 221, pp. 19–33, 1978.

[63] Gabow, H. N., Galil, Z., Spencer, T., and Tarjan, R. E., "Efficient algorithms for finding minimum spanning trees in undirected and directed graphs," *Combinatorica*, vol. 6, no. 2, pp. 109–122, 1986.

[64] Gadre, S. R., Kulkarni, S. A., and Pathak, R. K., "Rigorous bounds to molecular electron repulsion and electrostatic potential integrals," *The Journal of Chemical Physics*, vol. 91, p. 3596, 1989.

[65] Gardner, J. P., Connolly, A., and McBride, C., "A framework for analyzing massive astrophysical datasets on a distributed grid," in *Astronomical Society of the Pacific Conference Series*, vol. 376, p. 69, ASP; 1999, 2007.

[66] Giannantonio, T., Crittenden, R. G., Nichol, R. C., Scranton, R., Richards, G. T., Myers, A. D., Brunner, R. J., Gray, A. G., Connolly, A. J., and Schneider, D. P., "High redshift detection of the integrated Sachs-Wolfe effect," *Physical Review D*, vol. 74, no. 6, p. 063520, 2006.

[67] Gill, P. M., Johnson, B. G., and Pople, J. A., "A simple yet powerful upper bound for coulomb integrals," *Chemical physics letters*, vol. 217, no. 1, pp. 65–68, 1994.

[68] Gower, J. C. and Ross, G. J. S., "Minimum spanning trees and single linkage cluster analysis," *Applied Statistics*, vol. 18, no. 1, pp. 54–64, 1969.

[69] Gray, A. and Riegel, R., "Large-Scale Kernel Discriminant Analysis with Application to Quasar Discovery," *Proceedings of Computational Statistics*, 2006.

[70] Gray, A. G. and Moore, A. W., "Nonparametric Density Estimation: Toward Computational Tractability," in *SIAM International Conference on Data Mining*, 2003.

[71] Gray, A. G. and Moore, A. W., "Rapid Evaluation of Multiple Density Models," in *Artificial Intelligence and Statistics 2003*, 2003.

[72] Gray, A. G. and Moore, A. W., "Very Fast Multivariate Kernel Density Estimation via Computational Geometry," in *Joint Statistical Meeting 2003*, 2003. to be submitted to JASA.

[73] Gray, A. and Moore, A., "Rapid evaluation of multiple density models," in *Artificial Intelligence and Statistics*, vol. 2003, 2003.

[74] Gray, A. G. and Moore, A. W., "N-Body Problems in Statistical Learning," in *Advances in Neural Information Processing Systems (NIPS) 13 (Dec 2000)* (Leen, T. K., Dietterich, T. G., and Tresp, V., eds.), MIT Press, 2001.

[75] Gray, A. G. and Moore, A. W., "Nonparametric Density Estimation: Toward Computational Tractability," in *SIAM International Conference on Data Mining 2003*, 2003.

[76] Greengard, L. and Rokhlin, V., "A Fast Algorithm for Particle Simulations," *Journal of Computational Physics*, vol. 73, 1987.

[77] Greengard, L. and Rokhlin, V., "A Fast Algorithm for Particle Simulations," *Journal of Computational Physics*, vol. 73, pp. 325–248, 1987.

[78] Hamilton, A., "Toward better ways to measure the galaxy correlation function," *The Astrophysical Journal*, vol. 417, p. 19, 1993.

[79] Haque, I., Pande, V., and Walters, W., "The anatomy of high-performance 2d similarity calculations," *Journal of chemical information and modeling*, 2011.

[80] Häser, M. and Ahlrichs, R., "Improvements on the direct SCF method," *Journal of Computational Chemistry*, vol. 10, no. 1, pp. 104–111, 1989.

[81] Head-Gordon, M. and Pople, J. A., "A method for two-electron gaussian integral and integral derivative evaluation using recurrence relations," *The Journal of chemical physics*, vol. 89, no. 9, p. 5777, 1988.

[82] Held, M. and Karp, R. M., "The traveling-salesman problem and minimum spanning trees," *Operations Research*, vol. 18, no. 6, pp. 1138–1162, 1970.

[83] Holmes, M. P., Gray, A. G., and Isbell, Jr., C. L., "Fast kernel conditional density estimation: A dual-tree monte carlo approach," *Comput. Stat. Data Anal.*, vol. 54, pp. 1707–1718, July 2010.

[84] Holmes, M., Gray, A., and Isbell, C., "QUIC-SVD: Fast SVD using cosine trees," *Advances in Neural Information Processing Systems (NIPS)*, vol. 22, 2009.

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

[86] Jain, A. K., Murty, M. N., and Flynn, P. J., "Data clustering: a review," *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, 1999.

[87] Jain, A. K., Topchy, A., Law, M. H. C., and Buhmann, J. M., "Landscape of clustering algorithms," in *ICPR '04: Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 1*, (Washington, DC, USA), pp. 260–263, IEEE Computer Society, 2004.

[88] Jang, W. and Hendry, M., "Cluster analysis of massive datasets in astronomy," *Statistics and Computing*, vol. 17, no. 3, pp. 253–262, 2007.

[89] Jones, D. H., Saunders, W., Colless, M., Read, M. A., Parker, Q. A., Watson, F. G., Campbell, L. A., Burkey, D., Mauch, T., Moore, L., and others, "The 6df galaxy survey: samples, observational techniques and the first data release," *Monthly Notices of the Royal Astronomical Society*, vol. 355, no. 3, pp. 747–763, 2004.

[90] Kalidindi, S. R., Niezgoda, S. R., and Salem, A. A., "Microstructure informatics using higher-order statistics and efficient data-mining protocols," *JOM*, vol. 63, no. 4, pp. 34–41, 2011.

[91] Kamvar, S. D., Klein, D., and Manning, C. D., "Interpreting and extending classical agglomerative clustering algorithms using a model-based approach," in *ICML '02: Proceedings of the Nineteenth International Conference*

on Machine Learning, (San Francisco, CA, USA), pp. 283–290, Morgan Kaufmann Publishers Inc., 2002.

[92] Karger, D. R. and Ruhl, M., "Finding Nearest Neighbors in Growth-Restricted Metrics," *ACM Symposium on Theory of Computing*, pp. 741–750, 2002.

[93] Kerscher, M., "Statistical analysis of large-scale structure in the universe," *Statistical physics and spatial statistics*, pp. 36–71, 2000.

[94] Kershaw, K. and Looney, J., *Quantitative and dynamic plant ecology.* Edward Arnold London, 1973.

[95] Kirkpatrick, D. G. and Seidel, R., "The ultimate planar convex hull algorithm?," *SIAM J. Comput.*, vol. 15, no. 1, pp. 287–299, 1986.

[96] Kiskowski, M. A., Hancock, J. F., and Kenworthy, A. K., "On the use of Ripley's K-function and its derivatives to analyze domain size," *Biophysical journal*, vol. 97, no. 4, pp. 1095–1103, 2009.

[97] Krauthgamer, R. and Lee, J. R., "Navigating Nets: Simple Algorithms for Proximity Search," *15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 791–801, 2004.

[98] Kruskal, J. B., "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proc. Am. Math. Soc.*, vol. 7, pp. 48–50, 1956.

[99] Krzewina, L. and Saslaw, W., "Minimal spanning tree statistics for the analysis of large-scale structure," *Monthly Notices of the Royal Astronomical Society*, vol. 278, no. 3, pp. 869–876, 1996.

[100] Kutteh, R., Apra, E., and Nichols, J., "A generalized fast multipole approach for hartreefock and density functional computations," *Chemical Physics Letters*, vol. 238, no. 1, pp. 173–179, 1995.

[101] Lacey, C. and Cole, S., "Merger rates in hierarchical models of galaxy formation. II- Comparison with N-body simulations," *MNRAS*, vol. 271, no. 3, pp. 676–692, 1994.

[102] Lambrecht, D. and Ochsenfeld, C., "Multipole-based integral estimates for the rigorous description of distance dependence in two-electron integrals," *The Journal of chemical physics*, vol. 123, p. 184101, 2005.

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

[104] Lee, D. and Gray, A. G., "Faster Gaussian Summation: Theory and Experiment," in *Proceedings of the Twenty-second Conference on Uncertainty in Artificial Intelligence*, 2006.

[105] Lee, D. and Gray, A. G., “Fast High-dimensional Kernel Summations Using the Monte Carlo Multipole Method,” in *Advances in Neural Information Processing Systems 21*, 2009.

[106] Lee, D., Gray, A. G., and Moore, A. W., “Dual-Tree Fast Gauss Transforms,” in *Advances in Neural Information Processing Systems 18* (Weiss, Y., Schölkopf, B., and Platt, J., eds.), pp. 747–754, Cambridge, MA: MIT Press, 2006.

[107] Lee, D., Gray, A., and Moore, A., “Dual-tree fast gauss transforms,” *Advances in Neural Information Processing Systems*, vol. 18, p. 747, 2006.

[108] Lee, D., Vuduc, R., and Gray, A. G., “A distributed kernel summation framework for general-dimension machine learning,” in *SIAM International Conference on Data Mining*, vol. 2012, p. 5, 2012.

[109] Li, D., Baniassadi, M., Garmestani, H., Ahzi, S., Reda Taha, M., and Ruch, D., “3D reconstruction of carbon nanotube composite microstructure using correlation functions,” *Journal of Computational and Theoretical Nanoscience*, vol. 7, no. 8, pp. 1462–1468, 2010.

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

[111] Lupton, R., Gunn, J. E., Ivezic, Z., Knapp, G. R., and Kent, S., “The SDSS Imaging Pipelines,” in *Astronomical Data Analysis Software and Systems X: Proceedings of a Meeting Held at Boston, Massachusetts, USA, 12-15 November 2000*, vol. 10, p. 269, Astronomical Society of the pacific, 2001.

[112] March, W., Ram, P., and Gray, A., “Fast Euclidean minimum spanning tree: algorithm, analysis, and applications,” in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 603–612, ACM, 2010.

[113] March, W. B., Connolly, A. J., and Gray, A. G., “Fast algorithms for comprehensive n-point correlation estimates,” in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1478–1486, ACM, 2012.

[114] March, W. B., Czechowski, K., Dukhan, M., Benson, T., Lee, D., Connolly, A. J., Vuduc, R., Chow, E., and Gray, A. G., “Optimizing the computation of n-point correlations on large-scale astronomical data,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, p. 74, IEEE Computer Society Press, 2012.

[115] March, W. B., Ozakin, A., Lee, D., Riegel, R., and Gray, A. G., “Multitree algorithms for large-scale astrostatistics,” *Advances in Machine Learning and Data Mining for Astronomy, CRC Press, Taylor & Francis Group, Eds.: Michael J. Way, Jeffrey D. Scargle, Kamal M. Ali, Ashok N. Srivastava, p. 463-483*, vol. 1, pp. 463–483, 2012.

[116] Maurer, S. A., Lambrecht, D. S., Flaig, D., and Ochsenfeld, C., "Distance-dependent schwarz-based integral estimates for two-electron integrals: Reliable tightness vs. rigorous upper bounds," *The Journal of chemical physics*, vol. 136, p. 144107, 2012.

[117] McBride, C., "Our non-gaussian universe: Higher order correlation functions in galaxy surveys," in *Bulletin of the American Astronomical Society*, vol. 38, p. 957, 2007.

[118] McBride, C. K., Connolly, A. J., Gardner, J. P., Scranton, R., Scoccimarro, R., Berlind, A. A., Marín, F., and Schneider, D. P., "Three-point correlation functions of SDSS galaxies: constraining galaxy-mass bias," *The Astrophysical Journal*, vol. 739, no. 2, p. 85, 2011.

[119] McBride, C., *Our non-Gaussian universe: Higher order correlation functions in the Sloan Digitial Sky Survey.* PhD thesis, University of Pittsburgh, 2010.

[120] Moore, A. W., Connolly, A. J., Genovese, C., Gray, A., Grone, L., Kanidoris II, N., Nichol, R. C., Schneider, J., Szalay, A. S., Szapudi, I., and Wasserman, L., "Fast algorithms and efficient statistics: N-point correlation functions," in *Mining the Sky*, pp. 71–82, Springer, 2001.

[121] Mosaliganti, K., Janoos, F., Irfanoglu, O., Ridgway, R., Machiraju, R., Huang, K., Saltz, J., Leone, G., and Ostrowski, M., "Tensor classification of N-point correlation function features for histology tissue segmentation," *Medical image analysis*, vol. 13, no. 1, pp. 156–166, 2009.

[122] Mula, W., "Ssse3: fast popcount." http://wm.ite.pl/articles/sse-popcount.html, 2008. Accessed: 18 April 2012.

[123] Narasimhan, G., Zachariasen, M., and Zhu, J., "Experiments with computing geometric minimum spanning trees," in *Proceedings of ALENEX'00*, pp. 183–196, 2000.

[124] Nesetril, J., "Otakar Boruvka on minimum spanning tree problem Translation of both the 1926 papers, comments, history," *Discrete Math.*, vol. 233, pp. 3–36, 2001.

[125] Nevalainen, O., Ernvall, J., and Katajainen, J., "Finding minimal spanning trees in a Euclidean coordinate space," *BIT Numerical Mathematics*, vol. 21, no. 1, pp. 46–54, 1981.

[126] Obara, S. and Saika, A., "Efficient recursive computation of molecular integrals over Cartesian Gaussian functions," *The Journal of chemical physics*, vol. 84, p. 3963, 1986.

[127] Ochsenfeld, C., White, C., and Head-Gordon, M., "Linear and sublinear scaling formation of Hartree–Fock-type exchange matrices," *The Journal of chemical physics*, vol. 109, p. 1663, 1998.

[128] Osteen, R. E. and Lin, P. P., "Picture skeletons based on eccentricities of points of minimum spanning trees," *SIAM Journal on Computing*, vol. 3, no. 1, pp. 23–40, 1974.

[129] Pearson, R. and Coles, P., "Quantifying the geometry of large-scale structure," *Monthly Notices of the Royal Astronomical Society*, vol. 272, no. 1, p. 231, 1995.

[130] Peebles, P., *The Large-Scale Structure of the Universe.* Princeton Univ Pr, 1980.

[131] Peebles, P. and Groth, E., "Statistical analysis of catalogs of extragalactic objects. V-Three-point correlation function for the galaxy distribution in the Zwicky catalog," *The Astrophysical Journal*, vol. 196, pp. 1–11, 1975.

[132] Pelleg, D. and Moore, A., "Accelerating exact k-means algorithms with geometric reasoning," in *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 277–281, ACM Press, 1999.

[133] Pen, U., "Fast power spectrum estimation," *Monthly Notices of the Royal Astronomical Society*, vol. 346, no. 2, pp. 619–626, 2003.

[134] Pen, U., Zhang, T., Van Waerbeke, L., Mellier, Y., Zhang, P., and Dubinski, J., "Detection of dark matter skewness in the virmos-descart survey: implications for $\omega 0$," *The Astrophysical Journal*, vol. 592, p. 664, 2003.

[135] Pettie, S. and Ramachandran, V., "An optimal minimum spanning tree algorithm," *J. ACM*, vol. 49, no. 1, pp. 16–34, 2002.

[136] Pettie, S., "Finding Minimum Spanning Trees in $O(m\alpha(m, n))$ Time," tech. rep., University of Texas at Austin, Austin, TX, USA, 1999.

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

[138] Preparata, F. P. and Shamos, M. I., *Computational Geometry.* New York: Springer-Verlag, 1985.

[139] Prim, R. C., "Shortest connection networks and some generalizations," *Bell Sys. Tech. J.*, vol. 36, pp. 1389–1401, 1957.

[140] Pulay, P., "Convergence acceleration of iterative sequences. the case of scf iteration," *Chemical Physics Letters*, vol. 73, no. 2, pp. 393–398, 1980.

[141] Ram, P., Lee, D., March, W. B., and Gray, A., "Linear-time algorithms for pairwise statistical problems," *Advances in Neural Information Processing Systems*, vol. 23, 2009.

[142] Rasera, Y., Alimi, J., Courtin, J., Roy, F., Corasaniti, P., Fuzfa, A., and Boucher, V., "Introducing the dark energy universe simulation series (deuss)," *Arxiv preprint arXiv:1002.4950*, 2010.

[143] Ridgway, R., Irfanoglu, O., Machiraju, R., and Huang, K., "Image segmentation with tensor-based classification of N-point correlation functions," in *MICCAI Workshop on Medical Image Analysis with Applications in Biology*, vol. 1, 2006.

[144] Riegel, R., Gray, A., and Richards, G., "Massive-Scale Kernel Discriminant Analysis: Mining for Quasars," in *SIAM International Conference on Data Mining*, 2008.

[145] Ripley, B., "Locally finite random sets: foundations for point process theory," *The Annals of Probability*, pp. 983–994, 1976.

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

[147] Schmeja, S. and Klessen, R. S., "Evolving structures of star-forming clusters," *AAP*, vol. 449, pp. 151–159, 2006.

[148] Schwegler, E. and Challacombe, M., "Linear scaling computation of the Hartree-Fock exchange matrix," *Journal of Chemical Physics*, vol. 105, no. 7, pp. 2726–2734, 1996.

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

[150] Seife, C., "Breakthrough of the Year: Illuminating the Dark Universe," *Science*, vol. 302, pp. 2017–2172, December 19 2003.

[151] Sen, S. and Gupta, N., "Distribution-sensitive algorithms," *Nordic Journal of Computing*, vol. 6, no. 2, p. 194, 1999.

[152] Shamos, M. and Hoey, D., "Closest-point problems," in *16th Annual Symposium on Foundations of Computer Science*, pp. 151–162, 1975.

[153] Shao, J. and Tu, D., *The jackknife and bootstrap.* Springer Series in Statistics, 1995.

[154] Shectman, S. A., Landy, S. D., Oemler, A., Tucker, D. L., Lin, H., Kirshner, R. P., and Schechter, P. L., "The Las Campanas Redshift Survey," *arXiv preprint astro-ph/9604167*, 1996.

[155] Sibson, R., "Slink: An optimally efficient algorithm for the single-link cluster method," *The Computer Journal*, vol. 16, no. 1, pp. 30–34, 1973.

[156] Skeel, R. D., Tezcan, I., and Hardy, D. J., "Multiple Grid Methods for Classical Molecular Dynamics," *Journal of Computational Chemistry*, vol. 23, no. 6, pp. 673–684, 2002.

[157] Southwood, R. and Henderson, P., *Ecological methods.* Wiley-Blackwell, 2000.

[158] Springel, V., White, S. D., Jenkins, A., Frenk, C. S., Yoshida, N., Gao, L., Navarro, J., Thacker, R., Croton, D., Helly, J., and others, "Simulations of the formation, evolution and clustering of galaxies and quasars," *Nature*, vol. 435, no. 7042, pp. 629–636, 2005.

[159] Stoyan, D. and Stoyan, H., "Improving ratio estimators of second order point process characteristics," *Scandinavian Journal of Statistics*, vol. 27, no. 4, pp. 641–656, 2000.

[160] Strain, M. C., Scuseria, G. E., and Frisch, M. J., "Achieving linear scaling for the electronic quantum coulomb problem," *Science*, vol. 271, no. 5245, pp. 51–53, 1996.

[161] Subramaniam, S. and Pope, S. B., "A mixing model for turbulent reactive flows based on euclidean minimum spanning trees," *Combust. Flame*, vol. 115, no. 4, pp. 487–514, 1998.

[162] Szabo, A. and Ostlund, N. S., *Modern quantum chemistry: introduction to advanced electronic structure theory.* Courier Dover Publications, 1989.

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

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

[165] Szapudi, I., Prunet, S., Pogosyan, D., Szalay, A. S., and Bond, J. R., "Fast cosmic microwave background analyses via correlation functions," *The Astrophysical Journal Letters*, vol. 548, no. 2, p. L115, 2001.

[166] Tarjan, R., *Data Structures and Network Algorithms.* Society for industrial and Applied Mathematics, 1988.

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

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

[169] Turney, J. M., Simmonett, A. C., Parrish, R. M., Hohenstein, E. G., Evangelista, F. A., Fermann, J. T., Mintz, B. J., Burns, L. A., Wilke, J. J., Abrams, M. L., and others, "Psi4: an open-source ab initio electronic structure program," *Wiley Interdisciplinary Reviews: Computational Molecular Science*, vol. 2, no. 4, pp. 556–565, 2012.

[170] Waller, L. and Gotway, C., *Applied Spatial Statistics for Public Health Data.* Wiley-Interscience, 2004.

[171] Wan, P.-J., Calinescu, G., Li, X.-Y., and Frieder, O., "Minimum-energy broadcast routing in static ad hoc wireless networks," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2, pp. 1162–1171, IEEE, 2001.

[172] Wang, P., Lee, D., Gray, A., and Rehg, J., "Fast mean shift with accurate and stable convergence," in *Workshop on Artificial Intelligence and Statistics (AISTATS)*, 2007.

[173] Warren, H., *Hacker's delight.* Addison-Wesley Professional, 2003.

[174] White, C., Johnson, B., Gill, P., and Head-Gordon, M., "Linear scaling density functional calculations via the continuous fast multipole method," *Chemical physics letters*, vol. 253, no. 3-4, pp. 268–278, 1996.

[175] White, C. A. and Head-Gordon, M., "A j matrix engine for density functional theory calculations," *The Journal of chemical physics*, vol. 104, no. 7, pp. 2620–2629, 1996.

[176] White, C. A., Johnson, B. G., Gill, P. M., and Head-Gordon, M., "The continuous fast multipole method," *Chemical physics letters*, vol. 230, no. 1, pp. 8–16, 1994.

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

[178] Willett, P., "Recent trends in hierarchic document clustering: a critical review," *Inf. Process. Manage.*, vol. 24, no. 5, pp. 577–597, 1988.

[179] Wong, W.-K. and Moore, A., "Efficient algorithms for non-parametric clustering with clutter," in *Proceedings of the 34th Interface Symposium*, 2002.

[180] Xu, Y., Olman, V., and Xu, D., "Minimum spanning trees for gene expression data clustering," *Genome Inform.*, vol. 12, pp. 24–33, 2001.

[181] Yao, A., "An $O(|E| \log \log |V|)$ algorithm for finding minimum spanning trees," *Inf. Process. Lett.*, vol. 4, pp. 21–23, 1975.

[182] Yao, A., "On constructing minimum spanning trees in k-dimensional spaces and related problems," *SIAM J. Comput.*, vol. 11, no. 4, pp. 721–736, 1982.

[183] York, D. G., Adelman, J., Anderson Jr, J. E., Anderson, S. F., Annis, J., Bahcall, N. A., Bakken, J., Barkhouser, R., Bastian, S., Berman, E., and others, "The Sloan Digital Sky Survey: Technical Summary," *The Astronomical Journal*, vol. 120, no. 3, p. 1579, 2000.

[184] Zahn, C., "Graph-theoretical methods for detecting and describing gestalt clusters," *IEEE Trans. Comput.*, vol. 20, no. 1, pp. 68–86, 1971.

[185] Zhang, L. and Pen, U., "Fast n-point correlation functions and three-point lensing application," *New Astronomy*, vol. 10, no. 7, pp. 569–590, 2005.

[186] Zhang, T., Ramakrishnan, R., and Livny, M., "Birch: an efficient data clustering method for very large databases," in *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, (New York, NY, USA), pp. 103–114, ACM Press, 1996.

[187] Zhang, X. and Yu, C., "Fast n-point correlation function approximation with recursive convolution for scalar fields," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pp. 634–639, IEEE, 2011.

[188] Zhong, S. and Ghosh, J., "A unified framework for model-based clustering," *J. Mach. Learn. Res.*, vol. 4, pp. 1001–1037, 2003.

[189] Zhou, H., Shenoy, N., and Nicholls, W., "Efficient minimum spanning tree construction without Delaunay triangulation," *Information Processing Letters*, vol. 81, no. 5, pp. 271–276, 2002.