Astronomy and Computing 20 (2017) 44–51

Contents lists available at ScienceDirect

Astronomy and Computing

journal homepage: www.elsevier.com/locate/ascom

Full length article

A fast algorithm for identifying friends-of-friends halos

Y. Feng [a,b,*], C. Modi [a]

[a] Berkeley Center for Cosmological Physics Campbell Hall 341, University of California, Berkeley CA 94720, United States
[b] Berkeley Institute for Data Science, Doe Library 140, University of California, Berkeley CA 94720, United States

ARTICLE INFO

Article history:
Received 12 July 2016
Accepted 12 May 2017
Available online 19 May 2017

Keywords:
Cosmology
Halo
Simulation
Algorithm
Feature identification

ABSTRACT

We describe a simple and fast algorithm for identifying friends-of-friends features and prove its correctness. The algorithm avoids unnecessary expensive neighbor queries, uses minimal memory overhead, and rejects slowdown in high over-density regions. We define our algorithm formally based on pair enumeration, a problem that has been heavily studied in fast 2-point correlation codes and our reference implementation employs a dual KD-tree correlation function code. We construct features in a hierarchical tree structure, and use a splay operation to reduce the average cost of identifying the root of a feature from $O[\log L]$ to $O[1]$ (L is the size of a feature) without additional memory costs. This reduces the overall time complexity of merging trees from $O[L \log L]$ to $O[L]$, reducing the number of operations per splay by orders of magnitude. We next introduce a pruning operation that skips merge operations between two fully self-connected KD-tree nodes. This improves the robustness of the algorithm, reducing the number of merge operations in high density peaks from $O[\delta^2]$ to $O[\delta]$. We show that for cosmological data set the algorithm eliminates more than half of merge operations for typically used linking lengths $b \sim 0.2$ (relative to mean separation). Furthermore, our algorithm is extremely simple and easy to implement on top of an existing pair enumeration code, reusing the optimization effort that has been invested in fast correlation function codes.

1. Introduction

Friends-of-Friends clustering (FOF) is a common problem in cosmology for identifying features (clusters, usually called halos or groups) in density fields. Three common uses are (1) to find halos from N-body computer simulations in the 3-dimensional configuration space (Davis et al., 1985); (2) to find sub structures inside halos from N-body computer simulations in the 6-dimensional phase space (White et al., 2010; Behroozi et al., 2013); (3) and galaxy clusters from observational catalogs (Murphy et al., 2012) in the red-shifted configuration space. To assemble a physical catalog based on the feature catalog from the FOF algorithm, it is typical to prune the features (with some dynamical infall model), and to compute and associate additional physical attributes (e.g. spherical over-density parameters).

FOF algorithms identify features (or clusters) of points that are (spatially) separated by a distance that is less than a threshold (linking length b, typically given in units of mean separation between points) and assigns them a common label. A typical algorithm that solves this involves a breadth-first-search (henceforth BFS). During each visit of BFS, a neighbor query returns all of the particles within the linking length of a given particle. The feature label of these neighbors are examined and updated, and the neighbors whose labels are modified are appended to the search queue for a revisit. The first description of the friends-of-friends algorithm with breadth-first-search in the context of astrophysics following this paradigm is by Geller and Huchra (1983). A popular implementation is by Nbody-Shop (b), and more recently by Koda et al. (2016). A naive BFS algorithm queries perform neighbor queries on a point for multiple times, which is a target for optimization. For example, Kwon et al. (2010) reduce the number of queries by skipping visited branches of the tree.

Another widely used algorithm creates the friends-of-friends features by hierarchical merging (e.g. Springel, 2005). This was originally used for parallelization on large distributed computer architectures, as it allows a very large concurrency with a simple decomposition of the problem onto spatially disjoint domains. The algorithm is implemented in the popular simulation software GADGET,[1] but probably existed long before. It has been adopted in many codes, including a publicly available version in the AMR code ENZO (Bryan et al., 2014). To improve upon spatial queries, GADGET incrementally increase the linking length with

* Corresponding author at: Berkeley Center for Cosmological Physics Campbell Hall 341, University of California, Berkeley CA 94720, United States.
E-mail addresses: yfeng1@berkeley.edu (Y. Feng), modichirag@berkeley.edu (C. Modi).

[1] Though not available in the public version.

http://dx.doi.org/10.1016/j.ascom.2017.05.004
2213-1337/© 2017 Elsevier B.V. All rights reserved.

multiple iterations. During each iteration, the algorithm performs a neighbor query on a selected set of points, and merges the proto-features(proto-clusters) hosting these points by updating the labels of all constituent points of these two proto-features. The iterations are repeated till no additional merging is possible, and as a result, multiple neighbor queries on a data point are performed.

In the GADGET implementation, the proto-features are maintained as a forest of threaded trees, where the leaves (points) of any tree are connected by a linked list (hence the name threaded). During a merge operation, two link lists are joined, by traversing to the tail of the shorter linked list and connecting it to the head of the longer linked list. Two additional storage spaces of $O[N]$ are required to keep track of the size of proto-features and the threading linked list. The traverse increases the cost to merge a feature of length L to $O[L \log L]$, which can be a factor of a few more than optimal in terms of wall clock time. This short-coming of a linked list representation is discussed in detail in Section 21 of Cormen et al. (2009).

Due to these multiple iterations of the data, each making many expensive spatial queries (that slows down significantly as over-density grows) required in the existing algorithms, FOF has been generally considered a slow algorithm. As a result, algorithms that leads to an exact solution are rarely discussed in any detail in the literature of cosmology and astrophysics, while numerous approximated FOFs have been proposed as better alternatives to trade the speed with accuracy, some with more desirable physical characters (e.g. avoid bridging — counting nearby halos as one). The general idea of these approximated methods is that accurately tracking the outskirts of halos (features) is not important as it is already dominated by shot-noise in the numerical scheme of solvers. A few examples are improving the speed by using density information (Eisenstein and Hut, 1998), stochastic sub-sampling (Liu et al., 2008), and a relaxed linking length (Nbody-Shop, a).

Conceptually the FOF problem of cosmology is the same as a well known problem of computer science — that of identifying the maximum connected components (MCC) from a graph, where the graph is induced from the data set with an adjacent matrix

$$A(i,j) = \begin{cases} 0, & \mathit{Dist}(i,j) > b \\ 1, & \mathit{Dist}(i,j) \le b, \end{cases}$$

where b is the linking length. Put differently, if there is a path between two points, then they belong to the same feature, which is represented by a disjoint set. This problem is well studied and has a wide range of applications beyond the field of astrophysics. Numerous example implementations are freely available and integrated into machine learning packages (e.g. Shun and Blelloch, 2013).

In this paper we apply well known data structures and algorithms from computer science to derive a fast exact friends-of-friends algorithm that avoids expensive neighbor queries, uses minimal memory overhead, and rejects over-density slow down.

Our main inspiration is from the dual-tree algorithm introduced by Moore et al. (2001). The dual-tree algorithm efficiently calculates correlation functions by walking two spatial index trees simultaneously and avoids expensive and unnecessary neighbor queries. We use KD-Tree in the example implementation, though this can be replaced with a ball-tree for higher dimensional data and a chaining mesh for low dimensional data to achieve better performance (for the latter, see Sinha, 2016). Most importantly, the dual-tree algorithm calculates the correlation function with a single pass, enumerating each pair of neighboring points exactly once. Rewriting the FOF algorithm with pair enumeration avoids the repeated neighbor queries in breadth-first-search (BFS) algorithms and the GADGET hierarchical algorithm.

The main issue in the hierarchical merging algorithm, as pointed above, is the costly hierarchical merging of proto-features. We address this by representing the proto-features with a tree/forest data structure, and apply a splay operation in the merge procedure, which moves recently accessed nodes closer to the root, accelerating root finding operations in the average case (Cormen et al., 2009, Section 21). The splay operation was original introduced by Sleator and Tarjan (1985) to balance binary tree structures. In our case, splay reduces the average case complexity to construct final features of length L to $O[L]$ (as compared to $O[L \log L]$ with a linked list, as implemented in GADGET). It also eliminates the need to use additional $O[N]$ storage space for threading and balancing, resulting an extremely simple implementation. For completeness, we give an intuitive proof of finding correct solution with a single pass of pair enumeration with the splay tree data structure.

To further speed up our algorithm, especially in case of heavily over dense region where spatial queries become increasingly expensive (scaling as $O[((1+\delta)b^3)^2]$ where b is the linking length and δ is the over density), we implement another important optimization. We show that if two KD-Tree nodes (proto-features) are known to be fully-connected, the nodes need not be further opened and their respective hosting proto-features can be directly merged. This optimization eliminates most of merge operations in dense region and is particularly relevant in high resolution simulations that resolves kpc scale structures and over-density peaks of $\delta \gg 10^3$ (if we push high resolution simulations such as Hopkins et al., 2014, to a cosmological volume), though even for current generation of simulations it already reduces the number of merge operations by 20% to 50%.

The algorithm can be directly applied as the local section of a parallel friend of friend halo finding routine. Our implementation of the algorithm is available at https://github.com/rainwoodman/kdcount/blob/master/kdcount/kd_fof.c. We note that our reference dual tree pair enumeration code is not particularly optimized for performance, and hence we rather focus on the theoretical aspects of the algorithm and optimizations in this work. One can easily re-implement our algorithms with existing highly optimized fast correlation function codes to further improve the performance of FOF halo identification on actually problems.

The paper is organized as the following: in Section 2, we define the plain dual-tree friends-for-friends algorithm and prove its correctness; in Section 3, we will discuss the optimization; in Section 4, we perform scaling tests of the algorithm on two realistic cosmological simulation data sets.

2. Dual tree friends-of-friends algorithm

In this section, we describe our main algorithm, which is based on walking simultaneously two KD-trees that spatially indexes the data set being analyzed.

Definition 1. We define a KD-Tree with M nodes as a tuple of $(L[0:M], R[0:M], P[0:M])$, where $L[m]$ is the left child of m, $R[m]$ is the right child of m, and $P[m]$ is the list of points contained by m. We follow the convention that 0-th node is the root node. Several operations are also defined:

- $\mathit{Dist}(i,j) \equiv$ distance between ith and jth point in the dataset. Every time a pair of points are enumerated a $\mathit{Dist}(i,j)$ operation is performed.
- $\mathit{MinDist}(m,n)/\mathit{MaxDist}(m,n)$, the minimal/maximal distance of pairs between mth and nth node;
- $\mathit{MinDistB}(m,n)/\mathit{MaxDistB}(m,n)$, the bounds of minimal/maximal distance between mth and nth node.

The bounds are quickly computed from the bounding geometry of the KDTree nodes, as in a pair counting algorithm. We use the bound properties

$\mathit{MinDistB}(m,n) < \mathit{MinDist}(m,n)$