Astronomy and Computing 25 (2018) 159–167

Contents lists available at ScienceDirect

Astronomy and Computing

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

Full length article

Tree-less 3d friends-of-friends using spatial hashing

P. Creasey

Department of Physics and Astronomy, University of California, Riverside, CA 92507, USA

ARTICLE INFO

Article history:
Received 12 May 2018
Accepted 21 September 2018
Available online 29 September 2018

Keywords:
Methods: N-body simulations
Methods: Data analysis
Cosmology: Dark matter
Cosmology: Large-scale structure of universe
Methods numerical

ABSTRACT

I describe a fast algorithm for the identification of connected sets of points where the point-wise connections are determined by a fixed spatial distance — a task commonly referred to in the cosmological simulation community as Friends-of-Friends (FOF) group finding. This technique sorts particles into fine cells sufficiently compact to guarantee their cohabitants are linked, and uses locality sensitive hashing to search for neighbouring (blocks of) cells. Tests on N-body simulations of up to a billion particles exhibit speed increases of factors up to 20× compared with FOF via trees (a factor around 8 is typical), and are consistently complete in less than the time of a *k*-d tree construction, giving it an intrinsic advantage over tree-based methods. The code is open-source and available online at https://github.com/pec27/hfof.

1. Introduction

A way to identify dense groups of points in $\mathbb{R}^k$ is to construct connected components of points where direct connections are given for all pairs of points whose Euclidean separation is less than a 'linking length' b. This task is particularly common when processing cosmological simulations of the Λ cold-dark matter model to find the statistics of halos, which are virialised objects with a mean density of approximately 200× the critical density of the universe (Gunn and Gott, 1972; Bertschinger, 1985; Eke et al., 1996). Simulations of these objects discretise the (primarily dark) matter distribution into N bodies (Davis et al., 1985), and at any given time-scale of interest a catalogue of the connected components (or 'friends-of-friends' groups) in $\mathbb{R}^3$ is constructed (e.g. Jenkins et al., 2001; Reed et al., 2003, 2007; Crocce et al., 2010; Courtin et al., 2011; Angulo et al., 2012, although other alternatives exist, see Knebe et al., 2011 for an overview). The data sets of these simulations have grown from 32,768 particles (Davis et al., 1985) to the trillions of particles this decade (Skillman et al., 2014), making the production of these group catalogues challenging.

The ubiquitous algorithm for finding these friends-of-friends (hereafter FOF) groups is to perform a breadth-first search (e.g. Huchra and Geller, 1982). In this algorithm, finding connected components proceeds in the following manner: a stack of boundary points is maintained (initialised with a single point), and at each step a point is removed (marked as linked) and replaced by all its (unlinked) neighbours within the linking length, and this proceeds until the stack is empty, and the component is complete. This fixed-radius neighbour search is performed via organisation of the points into a *k*-d tree, a binary space partitioning structure where neighbour searches can be performed in $O(\log n)$ operations, n being the total number of points. Examples of such codes include Behroozi et al. (2013), the FOF code from the NbodyShop,[1] which is the almost unmodified ancestor of more recent codes such as Cola (Koda et al., 2016; Carter et al., 2018), yt (Turk et al., 2011) and probably many others unknown to this author. As far as I am aware *k*-d trees are used to perform the neighbour finding step in the non-public codes also, such as Kwon et al. (2010) and Fu et al. (2010) and Arepo (Springel, 2010, and also the non-public version of its predecessor Gadget-2). Some of these codes have been designed to create the group catalogue in parallel (often on the same cluster as the simulation), to mitigate the analysis problems.

Recently Feng and Modi (2017) have released an open source (*k*-dimensional) FOF algorithm kdcount[2] that is used in nbodykit (Hand et al., 2017). This algorithm uses the dual tree method (e.g. Moore et al., 2001) which exploits the fact that the searching points are hierarchically organised, allowing neighbour calculations (either inclusions or exclusions) to be calculated (typically excluded) for entire branches of the search tree. Their algorithm is not strictly breadth-first, a consequence of which is the need to merge components using a (customised) disjoint-set algorithm (Tarjan, 1975).

An alternative method for neighbour searches is the mapping of points on a fixed-grid, for example in the 'chaining mesh' method of Hockney and Eastwood (1988, sec 8.4.1) for a short-range component of the Coulomb force, and in the correlation function code Corrfunc (ascl:1703.003). By choosing a cell width greater than the

[1] http://faculty.washington.edu/trq/hpcc/, see also https://github.com/N-BodyShop/fof for the code.

[2] see http://rainwoodman.github.io/kdcount.

E-mail address: peter.creasey@ucr.edu.

https://doi.org/10.1016/j.ascom.2018.09.010
2213-1337/

search radius, one guarantees that all neighbours are within the 26 adjacent cells (in 3-d). Since the extent of the short range force is generally a multiple of the interparticle separation, this mesh is coarse w.r.t. the particles, corresponding to a modest memory footprint. Unfortunately, in the application FOF one is generally interested in linking lengths of $0.2\times$ the interparticle spacing (e.g. Davis et al., 1985), implying meshes of (at least) $125\times$ the particle count, and correspondingly a prohibitively large memory footprint.

A method to avoid such large data structures is to store only the filled cells, mapping them into a 1-d hash-table (Yuval, 1975; Bentley and Friedman, 1979) such that neighbouring cells can be (speculatively) searched at the map (hash) of their location. Such a method has been employed for fixed-radius neighbour searches (e.g. Teschner et al., 2003; Hastings et al., 2005, sometimes referred to as locality sensitive hashing). This is $O(1)$ for look-ups, though limitations include the expense of the hash function, the cost of resolving collisions (cells mapped to the same index) and the decreased coherence of memory accesses.

Spatial hashing has been successfully implemented by Wu et al. (2007) and Vijayalaksmi and Punithavalli (2012) for the related clustering algorithm DBSCAN, which is a generalisation of FOF to connecting components only about a subset of 'core' points (Ester et al., 1996).

In practice these codes are not applied to FOF calculations, possibly because they have not been optimised for this specialised use-case. Spatial hashing appears to be less common in computational physics, with exceptions such as the parallelisation scheme of Warren and Salmon (1993) and in the level set tracking methods of Brun et al. (2012).

This paper describes a novel algorithm for performing FOF in 3-d by grouping points into fine mesh whose cells are sufficiently compact to guarantee their points will be connected. These filled cells are grouped into 4^3 blocks which are stored via spatial hashing, the use of blocks decreasing the average number of hash-look-ups per filled cell. The merging of cells happens 'on the fly' as the blocks are inserted in a raster order, i.e. neighbours queries are only performed over blocks previously inserted in the table, and then the components are connected via the disjoint-sets algorithm, in a manner similar to Feng and Modi (2017). An example implementation is provided at https://github.com/pec27/hfof.

This paper is organised as follows. Section 2 describes the spatial hashing and linking algorithm, optimisations, and the method applied for periodic domains. Section 3 describes the comparison codes and test sets. Section 4 analyses the performance and compares with other codes and Section 5 concludes.

2. Spatial hashing for fixed-distance neighbour linking

In this section a methodology for FOF group finding via spatial hashing is described. Whilst this algorithm is not limited to cosmological simulations, these are the motivation, and some consideration of their features for this purpose as described in Section 2.1. Section 2.2 describes the arrangement of points into cells compact enough to guarantee connectivity, and their aggregation into blocks to reduce the number of lookups. Section 2.3 describes the hash function and 2.4 describes the adjustments to account for periodic domains.

2.1. Matter distribution in cosmological simulations

In the cosmological context, the clustering of matter produces halos which at the low-mass regime have a mass function approximating a power law

$$\frac{\mathrm{d}N}{\mathrm{d}M} \propto M^{-\alpha} \tag{1}$$

with $\alpha \approx 1.9$ (e.g. Reed et al., 2007), and notably $\alpha > 1$ implies a divergent low-mass tail, i.e. there should be an infinite number density of low-mass clusters, our discrimination of them limited only by our finite mass resolution (this is not strictly speaking true of the real universe, where diffusion damping terms will limit very low mass halos, but these are rarely resolved in cosmological simulations). A corollary of this is that the groups found are likely to be dominated (by number) by single particle groups,[3] and also that the number of groups is a significant fraction of the total number of particles (typically around one-third for cosmological simulations). As such a FOF algorithm needs to be efficient in the cases where the neighbourhood within a linking length is empty.

At the other extreme is that of high mass groups. Given the previous paragraph it may be tempting to think that most points are in small groups, however this is not the case. This can also be seen from Eq. (1), since $\int M\mathrm{d}M / \int \mathrm{d}M$ (i.e. the mass-weighted average halo mass) would have a divergent high-mass contribution, i.e. the average particle is in a group of $\gg 1$ particles, the exact number depending upon the mass function to higher masses (which in discrete simulations often depends upon artificial limitations such as the box size). As such the linking component of a FOF algorithm needs to scale well, in order to handle the connection of points to large groups.

Whilst both of these extremes need to be handled by group finding algorithms, I find in general the former seems more demanding, in that a significant fraction of the particles have zero neighbours within the linking length, and the majority of the computational time is spent confirming that these particles are truly isolated (see for example the 2nd panel of Fig. 1). It is helpful to keep this in mind during the following section.

2.2. Cell and block organisation

At the finest level, each particle is assigned to a cell according to its position in a lattice with cell-width

$$c = \frac{b}{\sqrt{3}} \tag{2}$$

where b is the linking length. Since the maximum distance between vertices in a unit hypercube in $\mathbb{R}^k$ is $\sqrt{k}$, this guarantees that any points in the same cell must belong to the same FOF group, which essentially reduces the problem of linking points to one of linking cells, and hereafter I will almost exclusively talk in terms of cells. The filled cells are sorted in raster order (i.e. sorted by z then y then x), which immediately places a bound on the complexity of the algorithm to be at least $O(n \log n)$, similar to that of the k-d tree construction.

This relationship of cell size to linking length is illustrated in Fig. 1 (first panel), where the locus of potential neighbours for positions in the central cell is highlighted. This lattice size guarantees that any neighbouring particle within a distance $< b$ must be within a 'stencil' of the 116 adjacent cells (Fig. 1 rightmost panel). This can be reduced by a factor of 2, to 58 neighbouring cells, by assuming that the lattice is built in raster order and using the symmetry of the distance metric,[4] however 58 turns out to be a rather large number of neighbour searches per cell (see discussion in Section 2.3 about optimisation of hash-table look-ups). As such, the cells are grouped into *blocks* of $4\times\ 4\times\ 4$ (i.e. 64) cells, i.e. the block width is

$$\Delta = 4c \tag{3}$$

[3] in the analysis of cosmological simulations groups with small (e.g. < 20) particles are generally ignored, but at the stage of constructing FOF groups these have yet to be filtered.

[4] i.e. 58 subsequent cells will be connected when they search for the current cell.