Detection of galaxies superclusters in simulated cosmological structures

Sergio Daniel Hernández Charpak
200922618

Advisor: Jaime E. Forero-Romero

April 24, 2016

Contents

1 Abstract **2**

2 Introduction **2**

3 Background **2**
3.1 Cosmic flows and peculiar velocities 2
3.2 Boundaries of Laniakea 3

4 Methods **3**
4.1 Simulations 3
4.2 Approach and Algorithms 3

5 Results **5**

6 Future Work **10**

A Region Growing Algorithm **10**

1 Abstract

Recently Tully et al. (2014) [**?**] used local cosmic flow information to define our local supercluster, Laniakea. In this work we present a study on large cosmological N-body simulations aimed at establishing the significance of Laniakea in a cosmological context. We explore different algorithms to define superclusters from the dark matter velocity field in the simulation. We summarize the properties of the supercluster population by their abundance at a given total mass and its shape distributions. We find that superclusters similar in size and structure to Laniakea are relatively uncommon on a broader cosmological context. We finalize by discussing the possible sources of systematics (both in our methods and in observations) leading to this discrepancy.

2 Introduction

In the Universe scene at large scales the galaxies group themselves in structures similar to a filament web which go through large voids regions and cross in regions called superclusters. Even though these structures can be easily detected at simple view, there are different possibilities to delimited them from physical criteria [**?**].

A proposal to define a supercluster is to use the flow of galaxies in this region of space. Within a supercluster, galaxies tend to flow to the most dense region as a consequence of the gravitational attraction process. In this way, spacial regions where the galaxy flow is convergent represent galaxies superclusters.

Recently a team build a galaxies velocities flow map of the local group in a scale of hundreds of light years [**?**]. In this map converging points were found and this team identified Laniakea, the galaxies supercluster which includes our galaxy, the milky way [**?**].

We propose to develop a method to detect a statistical significant number of galaxies supercluster in cosmological simulations. With this we aim to quantified if Laniakea can be considered as an atypical structure in the Universe.

3 Background

3.1 Cosmic flows and peculiar velocities

The peculiar velocity refers to the velocity relative to a rest frame. In the case of the Cosmicflows-2 map, the rest frame is the earth. The Laniakea supercluster of galaxies was mainly identified following an analysis of the peculiar velocities flows. A Wiener

filtering method was apply to obtained a better Signal/Noise proportion and separate the peculiar velocities from the cosmic expansion.
This is to be explained in further detail.

3.2 Boundaries of Laniakea

The contour of the region were reconstructed with the V-web Algorithm. This algorithm is deeply connected to the velocities as its core is the shear velocity tensor.

This is to be explained in further detail.

4 Methods

As we support the open-source community, we used the N-Body simulation open software Gadget-2 [**?**] to generate the simulations and source code in C and Python to treat the data obtained from the simulations. The development of the code will be in Github `https://github.com/sercharpak/Monografia/`.

4.1 Simulations

We used the N-Body simulation software Gadget-2 [**?**] widely used in the scientific community to generate a box of size 500 Mpc/h with 512^3 dark matter particles (DM). The initial conditions were generated with Springel's N-Genic software. The simulations ran on the HPC cluster at UNIANDES with 48 processors and took approximately 10 hours to run.

By DM particle we mean a particle which only interacts through gravity interaction. In the simulation each particle represents a galaxy. The DM particles are placed in a 3D grid layout first and then are perturbed following Poisson distribution before the simulation start, during the initial conditions generation.

Once the simulation starts, the particles interact only with gravity as time goes by following the cosmological constants. Here these were not modified, the default values were used.

We then use different algorithms to identify the superclusters within the simulation.

4.2 Approach and Algorithms

So far we have used a naive way to approach to the problem.

1. We calculate the magnitude of the velocity (the speed) of each particle.

2. We look for regions where the center has the highest speed and from it the speed decreases while the particle is further away from the center.

3. We define the limits of this region the regions where the speed begins to increase with the distance from the center.

We used a region growing algorithm, a simple image segmentation algorithm used to identify different regions in a image [**?**].
$f(x, y, z)$ denotes the input data. $S(x, y, z)$ denotes a seed array, with value 1 where the particle can be define as a seed and 0 where not. It is the same size of $f(x, y, z)$. Q denotes a predicate which is to be apply to the input data and determines if the region which starts at the seeds grows or not. Here Q denotes: "if the speed of the close particle is lower than the marked one but greater than a threshold, mark that particle and continue."

1. We choose seeds to begin the growth. In this case we choose the particles with high velocities. Here
$$|v| > th_{high}$$

2. We open a window of particles to look for 2 close particles from the seed s which do not for part of $S(x, y, z)$. Here:
$$window = 20000$$

3. Once we found the 2 close particles c we apply the predicate Q. Here:
$$Q := |v_s| > |v_c| > |th_{low}|$$

4. If the particle c satisfies Q, it is marked, $S(c) = 1$

5. Apply the algorithm to c, and so on (Recursion).

Here we defined empirically different thresholds in order to observer the results.
First we used:
$th_{low} = v_{min} + \frac{\sigma_v}{2}$ and $th_{high} = v_{max} - \sigma_v$.
Secondly we used:
$th_{low} = \bar{v} + 2\sigma_v$ and $th_{high} = \bar{v} + 8\sigma_v$.

The first version of the algorithm was written in python using the module pyGadgetReader [**?**] to read the data and transform it to NumPy arrays in Python. In Appendix A we attach the source code used.

5 Results

The simulation ran correctly on 48 processors in approximately 10 hours. We first visualized the simulation and its distribution of velocities.

We first visualize the speed histogram.

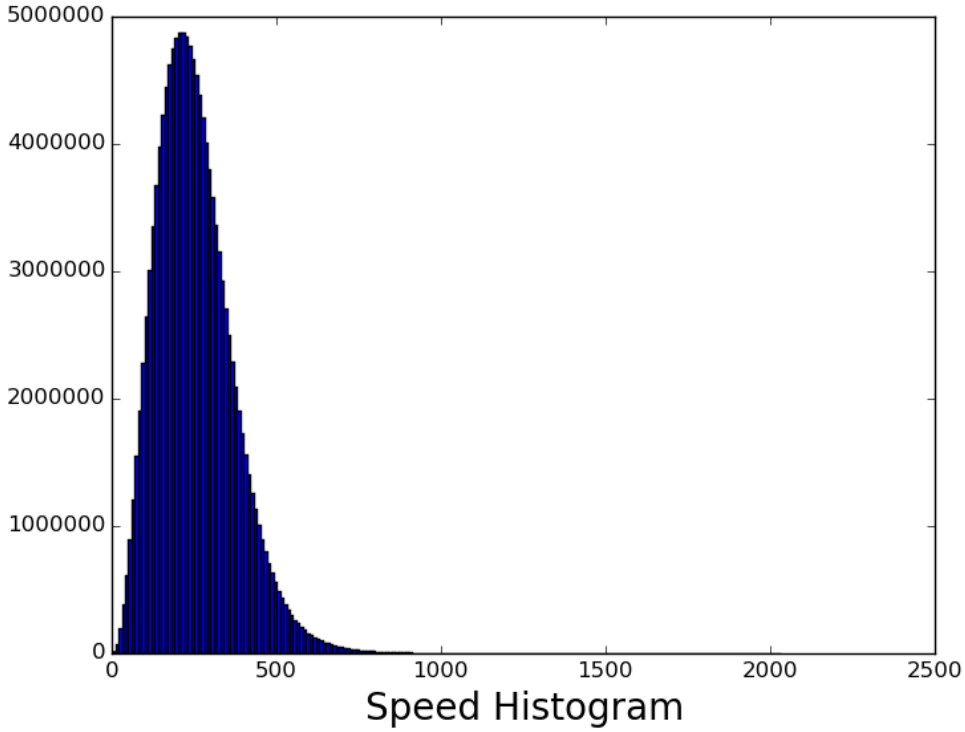

Figure 1: Speed Histogram of the simulation

The histogram resemble a Poisson distribution. This is a consequence of the perturbations generated in the initial conditions generation.

v_{max}	2023.87
v_{min}	0.34
σ_v	117.75

Table 1: Properties of the speed distribution

Our hypothesis is based on the most direct implication of gravity. The lower speed particles will mainly be in the border regions and the higher speed particles will be in

the center regions.

For this we must choose a threshold for low velocities and a threshold for high velocities. We obtain: $th_{low} = v_{min} + \frac{\sigma_v}{2} = 59.2$ and $th_{high} = v_{max} - \sigma_v = 1906.12$

We first have a look at the particle distribution of particles with speed higher than the threshold for low velocities.

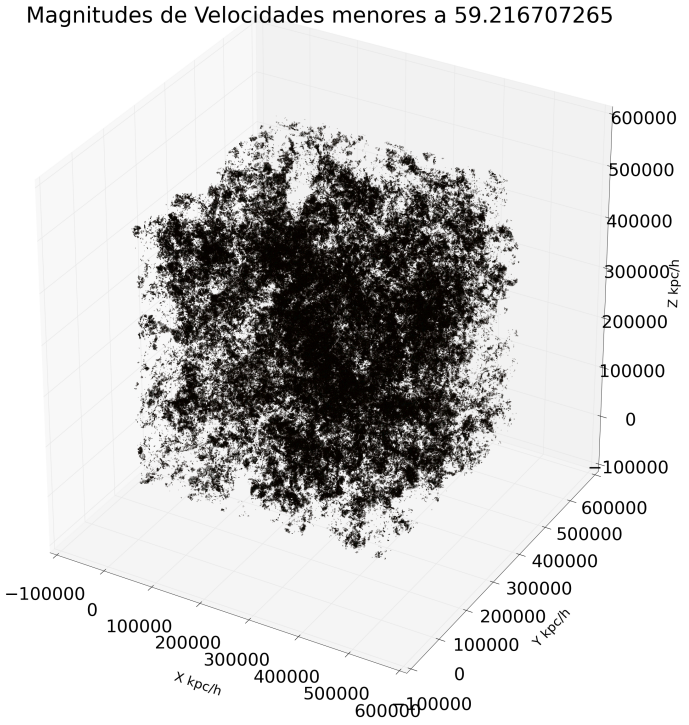

Figure 2: DM particles with $|v| < 59.2$

There seem to be some structures, but it is not clear enough. This is due to the high number of DM particles used in the simulation (512^3).

We produce cuts in the z-direction to visualize in 2-D the speed distribution.

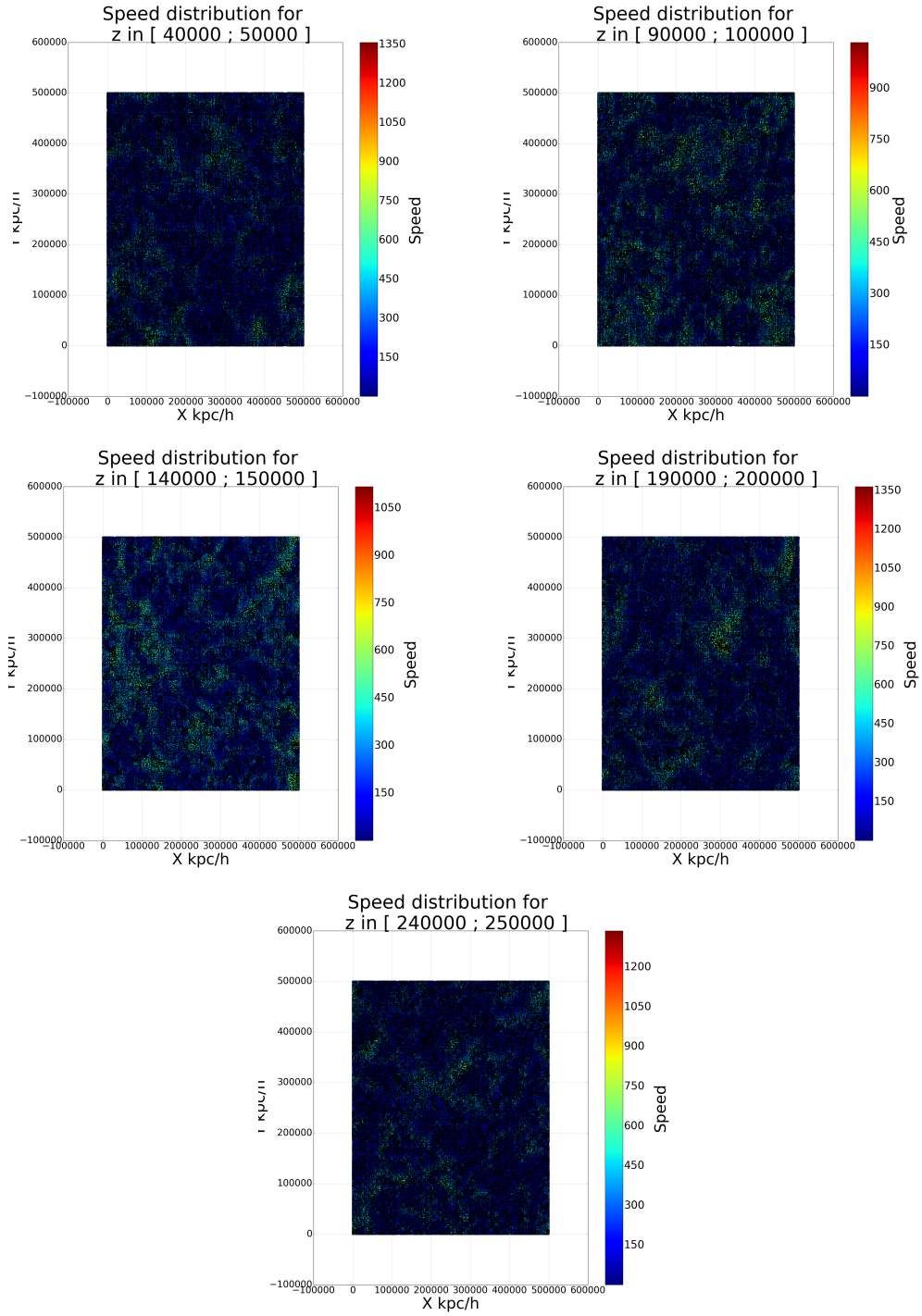

Figure 3: 2D slices of the speed distribution

We can observe more intuitive signs of structures related to the speed distribution. We can observe filament structures

It is more clear if we observe from one side the particles with speed greater than th_{high} and the particles with speed lesser than th_{low}.

Finally we apply our region growing algorithm and we obtain the following results.

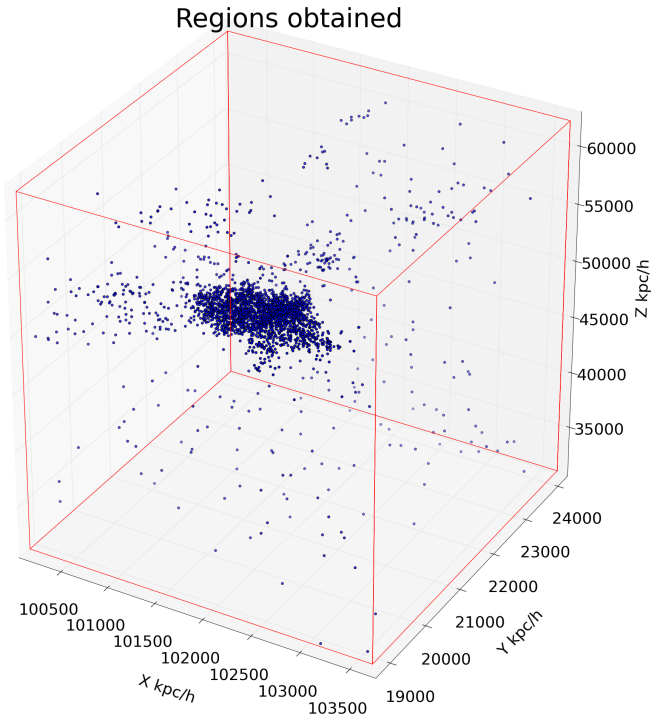

Figure 4: Region identified by the algorithm

And, changing the thresholds to: $th_{low} = \bar{v} + 2\sigma_v = 488.48$ and $th_{high} = \bar{v} + 8\sigma_v = 1195.0$
We obtained:

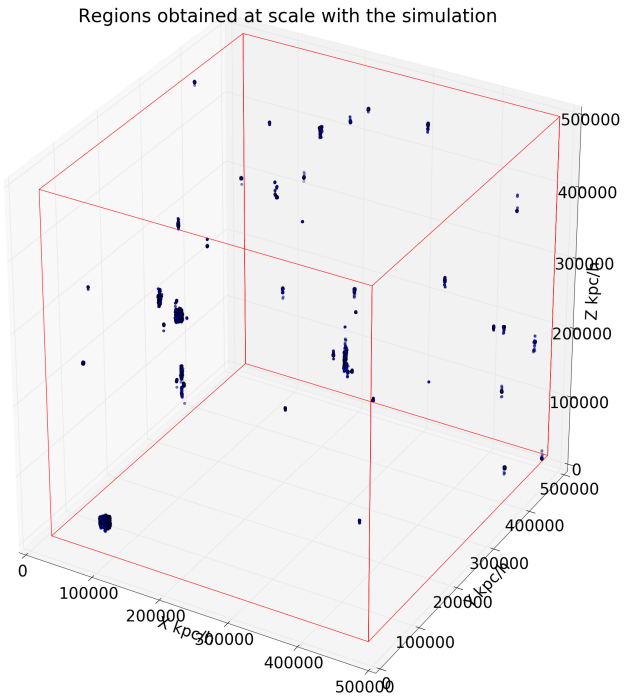

Figure 5: Region identified by the algorithm with the second thresholds

We have different observations:

1. The details of the region seem accurate with what we would expect. It is more dense in the center and less dense in the borders.

2. The regions obtained have a dimension of only a few Mpc/h (Laniakea's dimension is two orders of magnitude higher). This could mean that Laniakea is atypical.

3. There is first only one region identified. This indicates than all the high velocity particles are congregated in one region (the detected region). If we set lower th_{high} we can see we get more regions. The algorithm must be improved.

The script in Appendix A was executed in a student's laptop because of inconveniences with the HPC cluster. We think the main problem in the laptop is the limited memory (8GB). We expect to run it in the HPC to get more accurate results.

6 Future Work

- Understand more in detail how the work done by Hoffman et al[**?**] in the V-web algorithm, was used to determine Laniakea.
- Write our own Gadget-2 snapshop reader in C or Python which implements the region growing algorithm.
- Optimize our Algorithm. The current results are not sufficient.
- Run a bigger simulation (boxsize: $5 \times 10^6 Mpc/h$ and 1024^3 DM particles.
- Run the algorithm on this simulation.
- Compare the properties of the results with the Laniakea supercluster.
- Discuss the sources of systematics in both methods and observations.

A Region Growing Algorithm

Listing 1: Region Growing Algorithm Code

```
from pygadgetreader import *
import matplotlib
matplotlib.use('Agg')
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import matplotlib.pyplot as plt
import numpy as np

snap_file_sim='/lustre/home/ciencias/fisica/pregrado/sd.hernandez204/Outputs/28_02_2016/snapshot_005'

requested='dmcount'
n_dm=readheader(snap_file_sim,requested)
print "Number of Dark Matter particles"
print n_dm

array_pos=readsnap(snap_file_sim,'pos','dm')
array_vel=readsnap(snap_file_sim,'vel','dm')

array_magnitud_vel=np.zeros(n_dm)
for i in range(0,n_dm):
    array_magnitud_vel[i]=np.linalg.norm(array_vel[i,:])
print "The speed array has been calculated"

longitud = (int) (array_pos.shape[0])
print "The lenght of the position array should be the same number of particles"
print str(longitud)

array_copy=array_pos.copy()
array_results=np.zeros((longitud,4))
array_results[:,:-1] = array_copy
print "The results array has been created."

vel_max = np.max(array_magnitud_vel)
vel_min = np.min(array_magnitud_vel)
vel_std = np.std(array_magnitud_vel)
vel_mean = np.mean(array_magnitud_vel)

print "Vel_max, Vel_min, std_vel, vel_mean"
print vel_max, vel_min, vel_std, vel_mean
thresh_v_hig = vel_mean + 8.0 * vel_std
thresh_v_low = vel_mean + 2.0 *  vel_std
print "Thresh_high, Thresh_low"
print thresh_v_hig, thresh_v_low

#Gets the indexes of the seeds
```

```
indexes = np.where(array_magnitud_vel>(thresh_v_hig))[0]
print 'There are: '+ str(indexes.shape[0]) +' seeds'

def region_accretion(l):
    #Resize the window if necesary
    size_window_up = n_window
    if(((l+size_window_up) > longitud)):
        size_window_up = longitud - l
    size_window_down = n_window
    if(((l-size_window_down) < 0)):
        size_window_down = 0 + l
    distancias_l_up = np.zeros(size_window_up)
    distancias_l_down = np.zeros(size_window_down)
    distancias_l_up[:] = 1000000
    distancias_l_down[:] = 1000000
    if(size_window_up == size_window_down):
        for k in range(0,size_window_up):
            if(((l+k) < longitud)):
                if((array_results[l+k,3] == 0)):
                    distancias_l_up[k]=np.sqrt(np.sum((array_results[l,0:2]-array_results[l+k,0:2])**(2.0)))
            if(((l-k) > 0)):
                if((array_results[l-k,3] == 0)):
                    distancias_l_down[k]=np.sqrt(np.sum((array_results[l,0:2]-array_results[l-k,0:2])**(2.0)))
    else:
        for k in range(0,size_window_up):
            if(((l+k) < longitud)):
                if((array_results[l+k,3] == 0)):
                    distancias_l_up[k]=np.sqrt(np.sum((array_results[l,0:2]-array_results[l+k,0:2])**(2.0)))
        for k in range(0,size_window_up):
            if(((l-k) > 0)):
                if((array_results[l-k,3] == 0)):
                    distancias_l_down[k]=np.sqrt(np.sum((array_results[l,0:2]-array_results[l-k,0:2])**(2.0)))
    distancias_l_up[0]=1000000
    distancias_l_down[0]=1000000
    closest_l_up = np.min(distancias_l_up)
    index_closest_up = np.argmin(distancias_l_up)
    closest_l_down = np.min(distancias_l_down)
    index_closest_down = np.argmin(distancias_l_down)
    #Now si if these validate the condition
    #It has to see if it got any new candidates (not marked particles)
    if ((closest_l_up != 1000000)):
        vel_l = array_magnitud_vel[l]
        vel_closest_up = array_magnitud_vel[l + index_closest_up]
        if( (vel_l>= vel_closest_up>=thresh_v_low)&(array_results[l+index_closest_up,3]==0)):
            #it is marked
            array_results[l + index_closest_up,3] = 1
            #Recursion
            l_up = l + index_closest_up
            region_accretion(l_up)
    if ((closest_l_down != 1000000)):
        vel_l = array_magnitud_vel[l]
        vel_closest_down = array_magnitud_vel[l + index_closest_down]
        if( (vel_l >= vel_closest_down>=thresh_v_low)&(array_results[l-index_closest_down,3]==0)):
            #it is marked
            array_results[l - index_closest_down,3] = 1
            #Recursion
            l_down = l - index_closest_down
            region_accretion(l_down)

#Define the default size of the search window
n_window=10000
#Now mark the seeds
for i in indexes:
    array_results[i,3] = 1
#Now do the recursion
for i in indexes:
    #Recursion
    region_accretion(i)

print 'The regions have: '+ str(len(array_results[array_results[:,3] == 1,:])) +' galaxies.'
print 'Starting with points with V> '+ str(thresh_v_hig)
print 'Ending at points with V< ' + str(thresh_v_low)

#Now it gets the points of the regions
array_pos_region=array_results[(array_results[:,3] == 1),:]
```