Wavelet graphs for the direct detection of Gravitational Waves

Samantha Goldwasser*
AstroParticule et Cosmologie, Universite Paris Diderot
(Dated: August 1, 2017)

The research in this paper focuses solely on data analysis techniques. It is part of the development of a data analysis tool know as "Wavegraph" which is part of a larger analysis scheme known as the Coherent WaveBurst (cWB). This new time-frequency search method is an alternative to the matched filtering techniques that are usually employed in the detection of gravitational wave signals.

1. INTRODUCTION

Before discussing the specifics of the wavelet graphs there will be a small introduction which includes: general relativity, data analysis techniques used for the detection of gravitational waves (including the matched filtering technique as well as the cWB) and why these types of data analysis are different from other kinds of astronomical data analysis.

Foreword on general relativity and gravitational waves

In 1915 Albert Einstein published his theory of general relativity, which drastically changed our concept of "space" and consequently of "time". This theory relies in the geometry of space-time to explain what we perceive as "gravity". A corollary to this theory is that if a massive object spins in a way that is not spherically symmetrical then gravitational radiation will be emitted in the form of ripples; these ripples distort the space-time they travel through. By using laser interferometry we are able to detect these ripples, which we refer to as "gravitational waves". Ground-based interferometers such as LIGO (the Laser Interferometer Gravitational wave Observatory) and Virgo have been able to detect this kind of gravitational radiation.

Overview of Gravitational Wave Data analysis

In order to talk about gravitational wave data analysis we ought to consider the fact that this type of analysis is quite different from other kinds of conventional astronomical data analysis. Most astronomical observations occur through the lens of electromagnetic radiation, and so because of this most of the data we have from studying the universe can be found somewhere in the electromagnetic spectrum. With gravitational wave astronomy this is absolutely not the case. Because of the kind of instrument we use to detect the waves (Michelson interferometer) there are certain limitations we have to work around, among them the following: gravitational wave antennas are essentially omni-directional (with a response better than 50% of the RMS over 75% of the sky)[1] because of this, our data analysis systems will have to perform all-sky searches for sources when constrained to individual detectors (although it is theoretically possible to do directional search by triangulation when using three detectors). Additionally, interferometers cover three orders of magnitude in frequency (from 10 Hz to a few kHz) and so searches have to be carried out over this range of frequencies as well. Polarization measurements can only be achieved with a multiple detector network, and so algorithms that work with data from several interferometers have to be developed. A couple more limiting factors are the severe demand on both our theoretical understanding of the known waveforms and our data analysis pipeline, and finally the tremendous amount of data we are analyzing (a few kilobytes per second for as long as the instrument

*Electronic address: sgoldwasser93@ufl.edu

is running)[1].

Analyses of LIGO and Virgo data are carried out by collaborating groups who focus on four distinct source types: compact binary coalescences, un-modeled bursts, continuous waves, and stochastic background[1]. However, these categories represent archetypical extremes and many sources fall between these extremes; these sources can be analyzed with complementary methods that arise from the techniques developed for the four main chategories. Figure 1 [1] shows a way in which LIGO and Virgo searches can be broken down. As one moves from left to right waveforms increase in duration, while as one moves from top to bottom our previous definition of what the waveform should be decreases. The research that will be discussed in this paper focuses on long bursts, which is not explicitly in this diagram, however, it can be classified as a category in between bursts (lower left corner) and continuous waves (upper right corner).

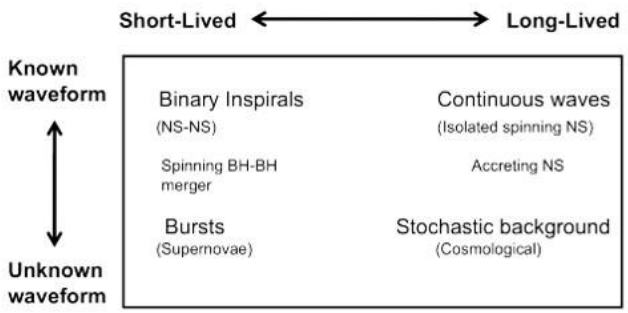

FIG. 1: Gravitational waveform categories which affect search strategies

Long bursts are expected to be emitted by compact astrophysical sources with hydrodynamic instabilities. Some of the examples of this category include: fallback accretion onto a new neutron star, non-axisymmetric accretion disc instabilities, and finally non-axisymmetric deformations in magnetars can also lead to generating this kind of signal [5]. The wavelet graphs have been proven successful in analyzing data coming from coalescing binaries of neutron star and/or blackholes [3](from the two detector LIGO network) and so this is our motivation for presently investigating the upper right corner of the aforementioned diagram.

The Matched Filtering Technique

Matched Filtering is a data analysis technique that efficiently searches for signals of known shape buried in noise. This technique consists in correlating the output of a detector with a waveform known as a filter[4]. Given a specific signal the task is to find the optimal template that would produce on average the best signal to noise ratio possible.

Most gravitational wave searches include the matched filtering technique in one way or another. The matching filtering technique is optimal [4], however, this approach requires the construction of large template banks using precise theoretical models (For example in the case of compact binary mergers the size of a template bank can be in the order of 10^5 templates). Even though progress has been made in constructing these template libraries and there are good approximations of the complete merger waveforms these are not from a single model. These waveforms are built from gluing several models together, and so we do not have complete waveforms for the full parameter space, we know there are cases where the waveform model is not accurate (i.e high spins and high mass ratio, or binaries in eccentric orbits), this is the motivation for using an alternative technique.

2. COHERENT WAVEBURST (CWB)

In addition to template-based searches there is another class of search that has been carried out for gravitational waves from un-modeled sources, this type of method is

independent of the large template banks [3]. This type of detection is based on excess signal power and correlation of signals found in different gravitational wave detectors. This is useful in the regions of the parameter space which have no descriptive waveforms to use as templates.
This kind of un-modeled search is expected to be less effective than the optimal matched filters but it encompasses a wider class of sources that may be missed by the template searches[2]. The Coherent Waveburst is one of the leading data analysis pipelines used for the detection of un-modeled transient gravitational wave events. Coherent WaveBurts was used to detect gravitational waves for the first time in history (GW150914). The Wavegraph project exists in order to improve the accuracy of the cWB pipeline [3]; because of this I will elaborate on how the cWB identifies event candidates and on the selection criteria for these events.

Coherent WaveBurst searches for short duration bursts of gravitational waves using information from all detectors in a network simultaneously. It operates in the frequency range of 24 Hz to 2048 Hz [5]. This method is performed using time-frequency decomposition. This decomposition transforms a sampled time series into a time-frequency map, the values of pixels in the map represent the energy amplitude at that time-frequency location [2]. Since each detector has its own time-frequency map, time-frequency maps from all detectors are combined into a single map coherently by applying a time delay with respect to the travel time of the gravitational wave.

Coherent WaveBurst identifies event candidates as regions on the time-frequency plane which have excess power [2]. A clustering procedure is employed in order to identify the same time-frequency event across multiple decomposition resolutions. Once all of the clusters are identified, then detection statistics are calculated for each event candidate.
In order to identify possible event candidates Coherent WaveBurst uses a likelihood ratio (equation 1)[2]. This is the ratio of the joint probability of a gravitational wave signal being present in the data to the joint probability of no signal.

$$L(h_+, h_\times) = \prod_j^N \prod_k^K exp(\frac{x_{kj}^2}{\sigma_{kj}^2} - \frac{(x_{kj} - \epsilon_{kj}(h_+, h_\times))^2}{\sigma_{kj}^2}) \tag{1}$$

Here x_{jk} is the sample output strain (or total energy in the detector), σ_{jk} is the noise variance and ϵ_{jk} is the detector response.

The likelihood functional is summed over each of the K detectors in the network and time-frequency regions composed of N samples. The likelihood functional is analytically maximized to determine estimators for $h_\times$ and h_+ amplitudes [2] (the two amplitudes of the polarizations of the gravitational wave). The process involves maximizing a significant number of parameters simultaneously and it may allow for unphysical solutions. In order to lessen this problem constrains are applied to the likelihood functional [2].

Selection criteria for events

Because data collected by gravitational wave detectors is infested with noise there are four sets of statistics which are used for selection of true events: First, a network correlation coefficient *cc* which tests the overall consistency of the event candidate based on the comparison of the reconstructed correlated signal energy (E_c) and the residual noise energy (E_n) [2]

$$cc = \frac{E_c}{E_n + E_c} \tag{2}$$

This network correlation coefficient varies between 0 and 1, where true gravitational wave events should have a *cc* near 1.

The second criterion we use is a penalty factor, which is a measure of consistency of reconstructed detector responses, and it is described by

$$P_f = k_{min}\sqrt{\frac{< x_k^2 >}{< \epsilon_k^2 >}} \tag{3}$$

The quantities $< x_k^2 >$ and $< \epsilon_k^2 >$ are the summed data sample amplitudes and reconstructed detector responses for a selected time-frequency region in an individual detector k. True gravitational wave events should have a P_f near 1 [2].

The next criterion is the energy disbalance, which is similar to P_f. It is characterized by the mismatch between the reconstructed energy of the event and the energy of the data. It is described by the following.

$$\Lambda = \frac{\sum_{k=0}^{K} | < x_k \epsilon_k > - < \epsilon_k^2 > |}{E_c} \tag{4}$$

True gravitational wave events should have Λ values near 0 [2].

Lastly, the coherent network amplitude is the main detection statistic used to rank events, the following is described by the equation below. This statistic (along with cc) is the main tool used to reject various types of glitches [2].

$$\eta = \sqrt{\frac{E_c cc}{K}} \tag{5}$$

Here K is the total number of detectors.

The following is a table with the values of selection criteria for each of the aforementioned statistics [2].

TABLE I: Values for selection criteria

Statistic	*Threshold*
Network Correlation cc	>0.6
Penalty factor P_f	>0.6
Energy disbalance Λ	<0.35
Coherent Network Amplitude η	>3.0

It is important to point out that the energy disbalance and the penalty factor p_f are not as important anymore, the main detection statistics used in the preset are the network correlation coefficient cc and the coherent network amplitude η

3. WAVELET GRAPHS

Compact Binary Coalescences

In few words the cWB extracts clusters of significant coefficients from time-frequency decompositions which result from the coherent combination of data from multiple gravitational wave detectors[3]. I will now elaborate on how these time-frequency representations are computed.

A wavelet graph is a graph that combines the paths from a family of chirp signals that cover a region of the parameter space. In order to obtain said graph one first determines the chirp path (or time-frequency-scale curve) that collects the large wavelet coefficients associated to a given chirp signal [3]. Figure 2 is an example of a wavegraph with 1009 nodes for a system of coalescing binaries in the mass range of 2.5-10 solar masses.

Generating the graph

The wavelet graphs are generated using a software called "Wavegraph". This software determines which wavelet in the Wilson basis has the maximum coupling with the chirp we are interested in [3]. This algorithm works in the continuous time, frequency and scale limit. The wavelet at time t_0, frequency f_0 and scale a_0 is expressed as:

$$\omega_0(f) = g(f - f_0; a_0) exp(-2\pi i f t_0) \tag{6}$$

Where the function $g()$ represents the envelope. The scale parameter is approximated by $a_0 = f_s \sigma_0$ where f_s is the sampling frequency. The time frequency map that we are interested in is defined by the following:

$$\rho_0^2 = \rho(t_0, f_0, a_0)^2 = | \int \frac{\omega_0(f) s(f)}{N(f)} \, \mathrm{d}f |^2 \tag{7}$$

Where $N(f)$ is the noise power spectrum. What we are looking for is the time t_0 and the scale a_0 which will maximize ρ_0 for a given f_0

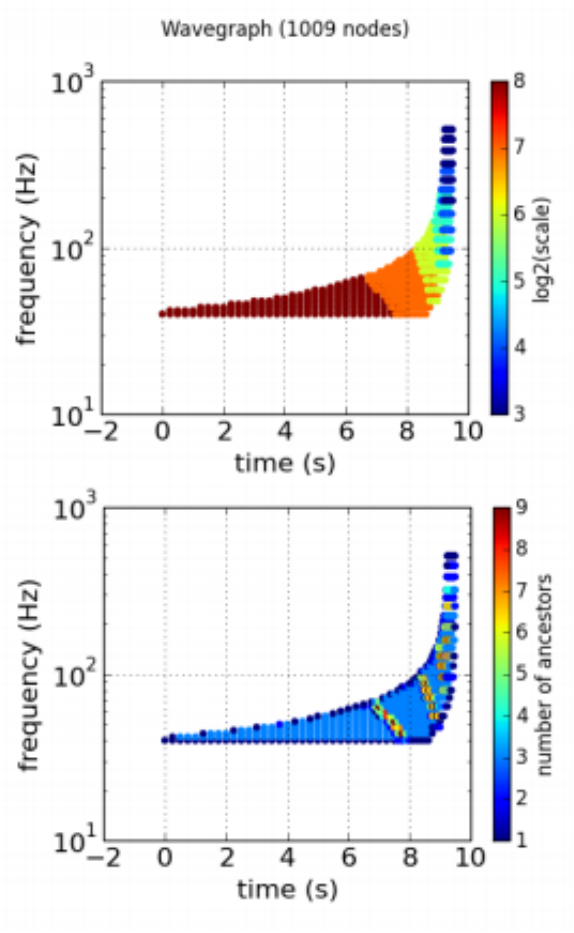

FIG. 2: Example of wavegraph, The top image shows the distribution of selected pixel nodes in the time, frequency, scale space. The bottom image shows the number of nodes per ancestor

Since chirps can always be expressed in the form of a complex exponential $s(f) = A(f)exp(i\psi(f))$ in the Fourier domain, we can write equation seven as an oscillatory integral and then evaluate it using the stationary phase approximation, and so we obtain:

$$\rho_0^2 = \rho(t_0, f_0, a_0)^2 = \frac{f_s|A(f_0)|^2}{\sqrt{\pi}N(f_0)^2 a_0} \tag{8}$$

This is what the time-frequency-scale map ρ looks like. Chirp paths are computed this way and then combined into the wavelet graph that collects the selected pixels and their connection with the previous pixel in the path (the ancestors).

After the wavegraph is generated the output is a list of nodes, each node has eight characteristics which include: the node ID, a time, frequency and scale indices, an average value, a stardard deviation value, a list of ancestors associated with that node. Each node also has a position in the time-frequency-scale plane. We did not have any tools for the visualization of the connections between the nodes of the wavegraph. Having said tool is necessary in order to understand and diagnose the graph, and so my project focuses on generating a visual representation of these wavelet graphs.

Data visualization algorithm for wavelet graphs

The algorithm for the visualization of wavegraphs that I have written (Figures 7,8,9,10) takes each node and it assigns them a position depending on their time and frequency location, afterwards it assigns the node a color depending on the scale used (this is a representation of the third dimension) and a size depending on the number of connections associated with said node. Then each node is connected to its ancestors with a black line. If one is interested in a specific scale the code can be easily modified (Figure 10 lines 179-184) to show, in red, the nodes associated with that scale, their respective ancestors and each of their connections. This feature exists in order to help visualization since it is certainly easier to focus on a specific scale (scale) at a time in order to understand the behavior of the graph. Testing my code on previously tested data from simulations is a good way to get a feel for if the code is working properly or not. This is why the first example of my wavegraph visualization uses data from coalescing binaries (upper left corner of diagram in FIgure 1) even though this paper focuses on long bursts.

Figures 3 and 4 show a Wavegraph of 533 nodes computed using 2950 Binaries; in the first figure we see every scale and every connection with all 533 nodes, and in the second figure we select a specific scale to only show

the nodes associated with that scale and their ancestors. One modification that could be done to the code in the future would be for ancestors and nodes to have different colors in order to be able to identify them easily.

Long bursts from compact astrophysical sources with hydrodynamic instabilities are expected to decrease in frequency approximately linearly in time, these unmodeled transient events are expected to have durations between 10 and 500 seconds [5]. I wrote a short code (Fig 6) that generates an arbitrary chirp signal whose frequency increases linearly in time by entering four user inputs: the initial and final frequencies, the sampling frequency and the total time of the chirp. Using this code I generated a waveform that increases in frequency going from 100 Hz to 200 Hz.

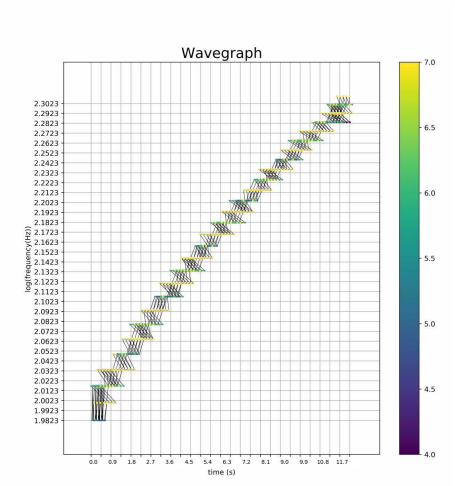

FIG. 5: Wavegraph created from one single chirp of frequency which increases linearly in time

Afterwards I ran this chirp through wavegraph to generate the time-frequency-scale graph and finally I plotted all the nodes by using the wavegraph visualization code I wrote. The result is shown in figures 5 and 6, where figure 5 is the graph for all the scales (4 through 7) and figure 6 is the same but limiting the code to a window of a=7 . The graph grows linearly with time, as we we would expect intuitively.

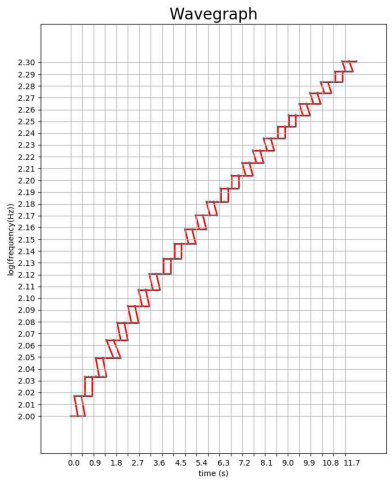

FIG. 6: Wavegraph created from one single chirp of frequency which increases linearly in time

Once I generated figure 5 and was sure my wavelet graph visualization scheme was working properly I ran some time-strain data (which is chirping downward), provided by my advisor, through wavegraph. After the graph was generated I ran the file through my visualization code.

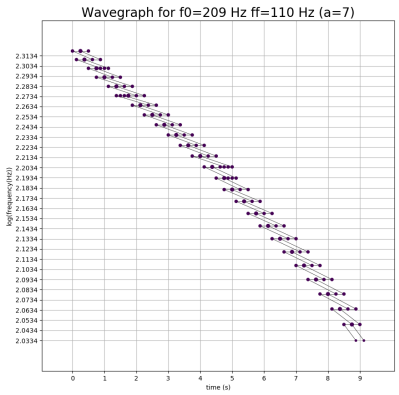

FIG. 7: Wavegraph from time-strain data, chirping downward from 209 to 110 Hz

Surprisingly the resulting graph did not follow the correct chirp path, and so I generated a chirp down signal by using my generic waveform generator (figure 7) in order to emulate an analogous signal to the time-strain chirp.

I used the same parameters (initial,final and sampling frequency and total chirp path) $f_0 = 110$ Hz, $f_{end} = 209$ Hz, $f_s = 1024$ Hz and $t_f = 9$ seconds as the time-strain data has.

The time-strain data plotted using the visualization code resulted in Figure 7, and the emulated signal using the same parameters resulted in Figure 8. Comparing figures 5 and 6 with figures 7 and 8 it is apparent that the graph is not as smooth, and if one looks at the node numbers and the chirp path taken between these nodes one can see that the graph actually goes back in time (and sometimes forward and then back in time) which is borderline nonesensical.

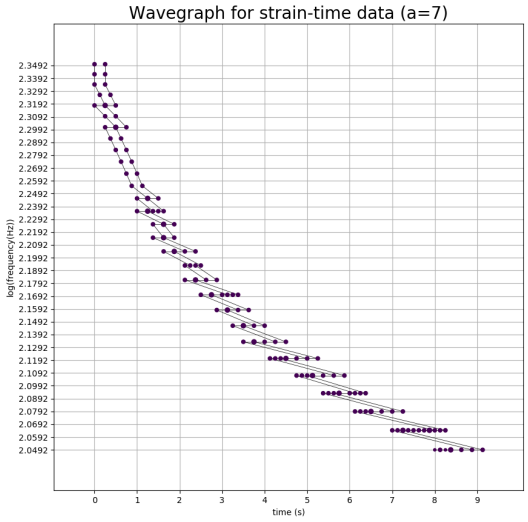

FIG. 8: Wavegraph from generic waveform generator, chirping downward from 209 to 110 Hz

My code shows evidence that the sorting procedure for finding connections between nodes does not work for long bursts of decreasing frequency, it exclusively works with coalescing binary systems (since they chirp upward) and with systems that increase linearly in frequency (such as the ones I generated using the generic graph code). The issue became more obvious when I generated a graph from a generic waveform chirping downward from 173 to 119 Hz (Figure 7), the chirp path taken goes back and forward in time and it is rough in contrast to the smooth chirp path of the graphs increasing linearly in frequency. The need for a new sorting scheme in order to connect the nodes properly is evident, and so the next step for the visualization of wavegraphs should focus on diagnosing the problem of why the scheme seems to fail with data that decreases in frequency with time.

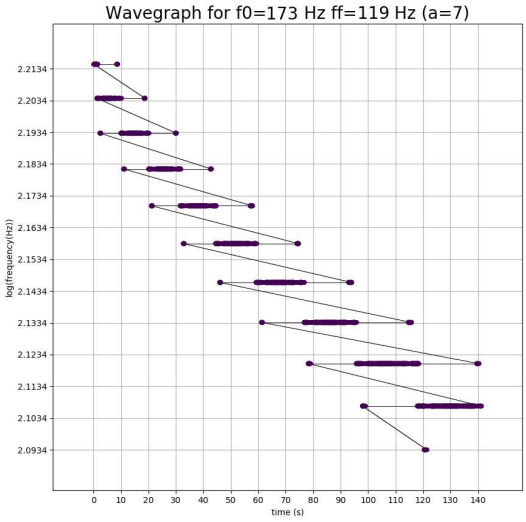

FIG. 9: Wavegraph generated from generic waveform chirping downward from 173 to 119 Hz

4. BIBLIOGRAPHY

[1] K.Riles, *Gravitational Waves: Sources, Detectors and Searches*, Physics Department, University of Michigan, 2013.

[2] Classical and Quantum Gravity, Volume 26, Number 20, *A burst search for gravitational waves from binary black holes*, C Pankow, S Klimenko, G Mitselmakher, I Yakushin, G Vedovato, M Drago, R.A Mercer, P Ajith 2009 IOP Publishing Ltd.

[3] Eric Chassande-Motin, Eric Lebigot, Hugo Magaldi, Even Chase, Archana Pai, Gayathri V, Gabriele Vedovato *Wavelet graphs for the direct detection of gravitational waves*, APC, Univ Paris Diderot, CNRS/IN2P3, CEA/irfu, Obs. de Paris, Sorbonne Paris Cite, France. Tsinghua University, Beijing, China. IISERTVM, Computer Science Building, CET Campus Trivandrum Karala, India. INFN, Seziono di Padova, Padova, Italia ecm@apm.univ-paris7.fr

[4] Living reviews in relativity *Physics, Astrophysics and Cosmology with Gravitational Waves*, B.S. Sathyaprakash, Bernard F. Schutz, 2009.

5. ACKNOWLEDGEMENTS

I would like thank the University of Florida and the National Science foundation for giving me the opportunity to partake in this program and expand, so heavily, my knowledge of Python and data analysis techniques; since these are priceless abilities. I'd like to thank everyone involved in the organization of the IREU program, including Bernard Whiting, Guido Muller, Michaela Pickenpack, Kristin Nichola and Andrew Miller. I would like to give special recognition to the people who helped me at the APC, among them Eric Chassande-Mottin, Philippe Bacon and Gayathri Vivekanandhan, not only for teaching me about Python and about the wavegraph project but also for making my stay in Paris truly unforgettable.

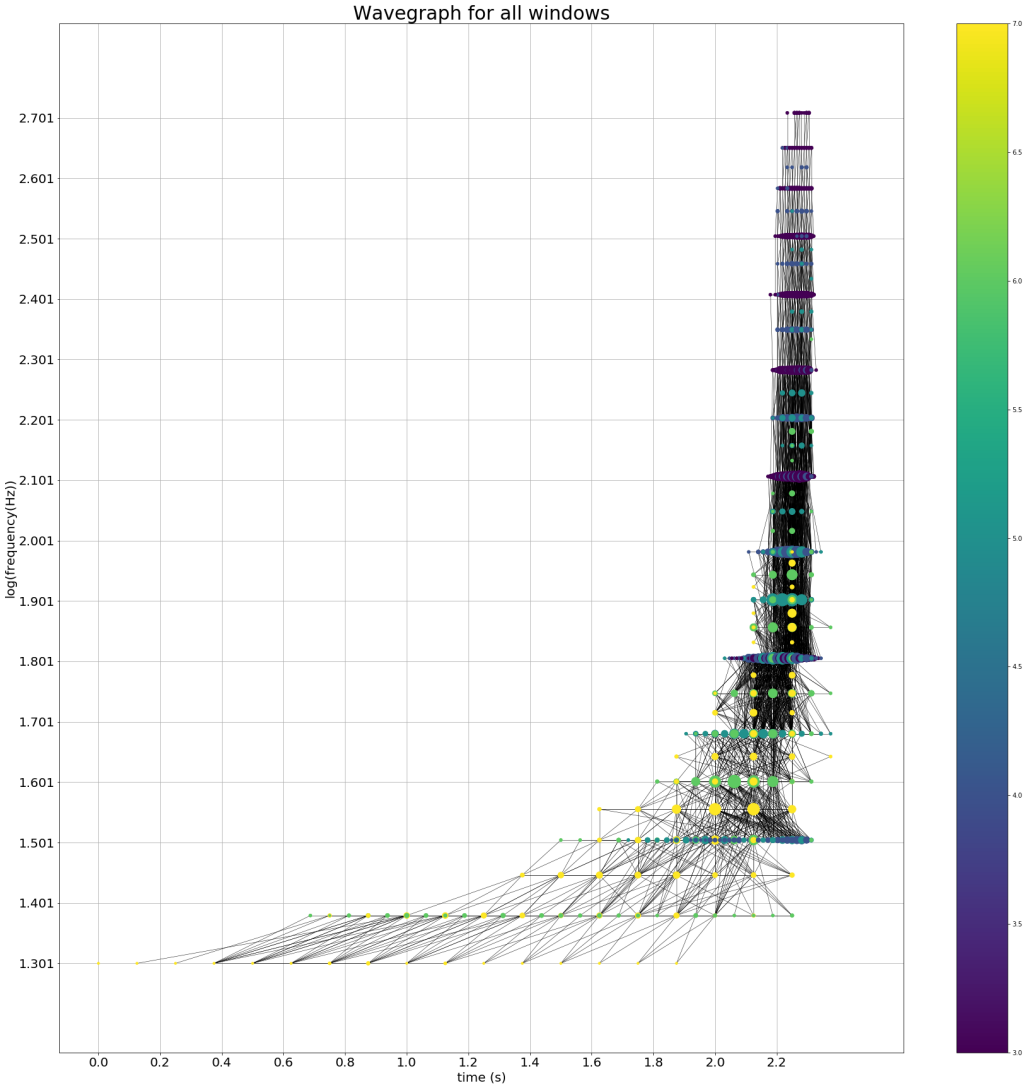

FIG. 3: Example of wavegraph for coalescing binaries. Computed using 2950 binaries, the graph has 533 nodes, in this graph we see nodes of every scale with different colors, the bigger the scale the lighter the color

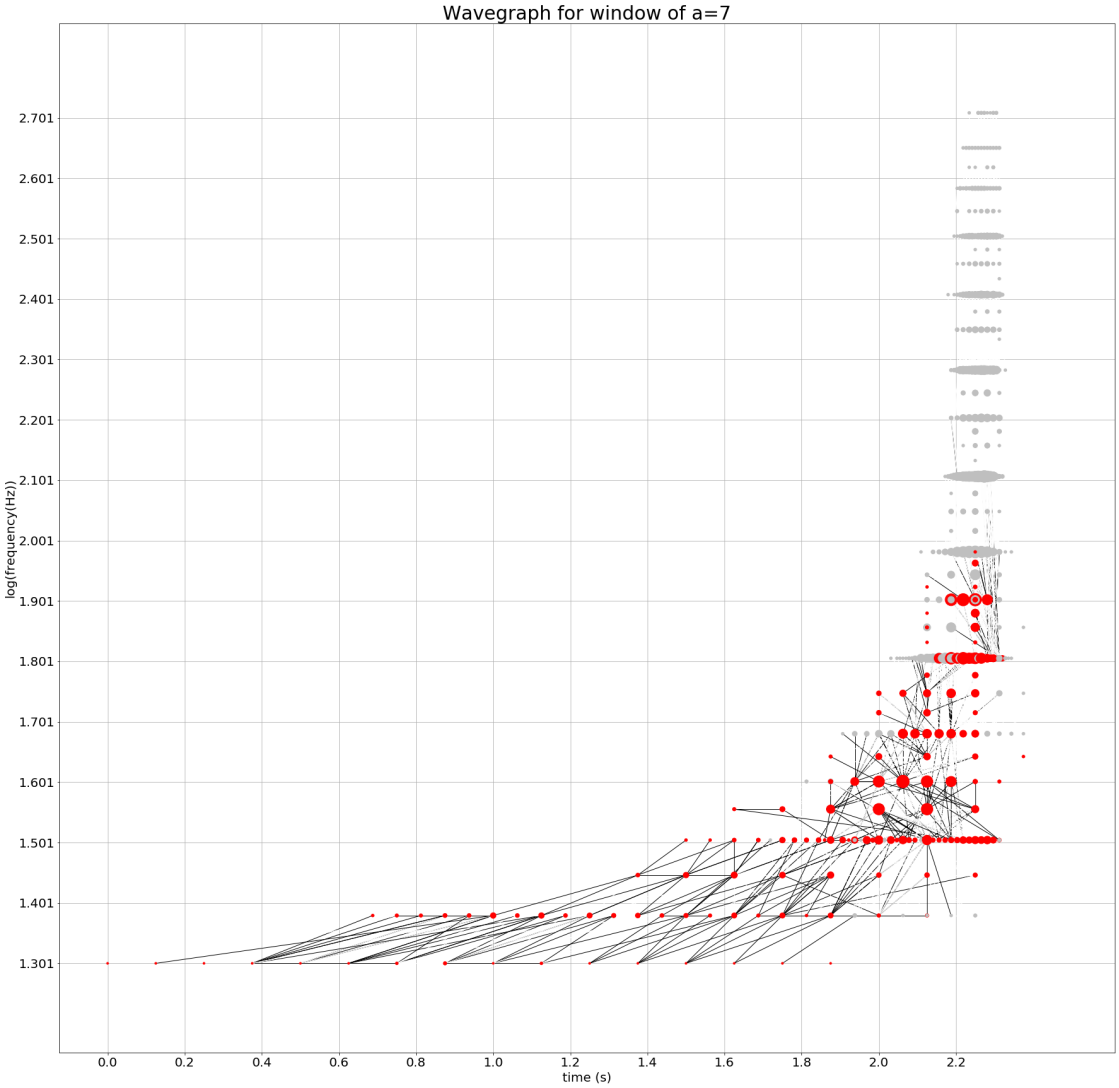

FIG. 4: Example of wavegraph for coalescing binaries focusing on a specific scale of size 7. Computed using 2950 binaries, the graph has 533 nodes total, in red we see the nodes of scale 7 and their ancestors

```python
#Waveform generator with time dependent frequency and f_final parameter

import numpy as np
import pylab
import matplotlib.pyplot as plt

#Define omega as a function of frequency
def freqt(t,f_0):
    freqt=a*t+f_0
    return freqt
def omegat(t,f_0):
    omegat = 2*np.pi*freqt(t,f_0)
    return omegat

#Parameters (user inputs)
fs=1024 # Samples/Second
f_0= 100 # Hz
f_final=200 #Hz
tfinal=10 # Seconds

#Total number of samples
N=fs*tfinal

#Rate of change
a=(f_final-f_0)/tfinal

#Define time as a discrete variable
t=np.linspace(0,tfinal,N)
freqtime=freqt(t,f_0)

#Define frequency and time. Make sure fs>2f is satisfied at every point
f_new=[]
t_new=[]
for i in range (0,len(freqtime)):
    if fs >= 2*freqtime[i]:
        t_new.append(t[i])
        f_new.append(freqtime[i])
    else:
        print "Cannot sample signal with fs < 2f"

y=np.cos(np.pi*a*np.array(t_new)*np.array(t_new)+2*np.pi*f_0*np.array(t_new))
#To plot and show
plt.plot(t_new,y,'b')
plt.axis([0, 10, -1.1,1.1])
plt.xlabel('time')
plt.ylabel('amplitude')
#plt.stem(t,y, 'm', )
plt.show()

```

FIG. 10: Generic Waveform Generator

```
#Node Graph
% matplotlib inline
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import math
import matplotlib.colorbar as cbar

#Import text file
text_file = open("wavegraphevent.txt", "r")
lines = text_file.readlines()

#Eliminate the lines that start out with % or ##
lines = [i for i in lines if i[1] != '#' and i[0] != '%']

#Divide the list into a list of node characteristics and freq-time-window
oldcharacteristics = lines[::2]
oldfreqtime = lines[1::2]

#Node characteristics
characteristics=[]
for x in oldcharacteristics:
    a=map(float, x.split(' '))
    characteristics.append(a)

#NodeIDs
nodeIDs=[]
for i in characteristics:
        nodeID =i[0]
        nodeIDs.append(nodeID)
nodeIDs=[int(i) for i in nodeIDs]

#Ancestors
nodeIDs_anc=[]
for i in characteristics:
    if len(i) > 7:
        nodeIDanc=i[0]
        nodeIDs_anc.append(nodeIDanc)

#For the Freq-time-window lines
freqtime=[]
for i in oldfreqtime:
    newelements= i.split(' ')
    freqtime.append(newelements)

#Frequencies
frequencies=[]
for i in freqtime:
    eachfrequency=float(i[2])
    frequencies.append(eachfrequency)
```

FIG. 11: Wavegraph visualization 1/4

```
51
52 logfrequencies=[]
53 for i in frequencies:
54         everylog=math.log(i,10)
55         logfrequencies.append(everylog)
56
57 #Times
58 times=[]
59 for i in freqtime:
60     eachtime=float(i[6])
61     times.append(eachtime)
62
63 #To make an array for each window to have a color (Have all nodes colored)
64 #Windows
65 dimensions=[]
66 for i in freqtime:
67     eachdimension=float(i[10])
68     dimensions.append(eachdimension)
69 dimensionsarray=np.array(dimensions)
70
71 newtimes= tuple(times)
72 newfrequency =tuple(logfrequencies)
73 coordinates = zip(newtimes,newfrequency)
74
75 #A dictionary with nodeIDs and coordinates as positions
76 dic=dict(zip(nodeIDs,coordinates))
77
78 #Getting a list of the node IDs repeated as many times as there are ancestors per node
79 repeated_nodeIDs=[]
80
81 for i in characteristics:
82     if len(i) > 7:
83         a=len(i)-7
84         repeated_nodeIDs=np.concatenate((repeated_nodeIDs,np.ones(a)*i[0]))
85 repeated_nodeIDs = [int(i) for i in repeated_nodeIDs]
86
87
88 #Getting a list of all the ancestors
89 ancestors=[]
90 L=len(characteristics)
91 for i in range (0,L):
92     l=len(characteristics[i])
93     b=characteristics[i][7:l]
94     ancestors.append(b)
95
96 list_of_ancestors = [item for sublist in ancestors for item in sublist]
97 list_of_ancestors = [int(i) for i in list_of_ancestors]
98
99 #For the colors of the nodes of a specific window
100 #Get the positions of all the indices for the window I am interested in
101 colorlist=[]
102 indx1=[]
103
```

FIG. 12: Wavegraph visualization 2/4

```
103
104 #User enters the window she's interested in, indx1 is the node ID which has that value for the window
105 userinp = 7
106
107 for i,j in enumerate(dimensions):
108     if j == userinp:
109         indx1.append(i)
110
111 #Getting the values for the node IDs associated with those positions (This is the same as the indx1 but in int)
112 nodeIDind=[]
113 for i in indx1:
114     each = nodeIDs[i]
115     nodeIDind.append(each)
116
117
118 indx2=[]
119 for i,j in enumerate(repeated_nodeIDs):
120     for u in nodeIDind:
121         if j == u:
122             indx2.append(i)
123
124 for i in indx2:
125     eachone=list_of_ancestors[i]
126     colorlist.append(eachone)
127
128 colorlist = colorlist+indx1
129 colorlist = list(set(colorlist))
130 colorlist = [int(i) for i in colorlist]
131
132 finalcolor=[]
133 finalcolor=(len(nodeIDs)-len(colorlist))*['0.75']
134 for i in colorlist:
135     finalcolor.insert(i,'r')
136
137 #Convert the newAncestos and the corresponding nodeIDs into a tuple so I can use them to connect the nodes
138 newAncestors= tuple(list_of_ancestors)
139 newNodeID_anc =tuple(repeated_nodeIDs)
140 e = zip(newAncestors,newNodeID_anc)
141
142
143 #This is for getting a list of colors to assign to the edges
144 indicesforedges=[]
145 for i,j in enumerate(repeated_nodeIDs):
146     for u in indx1:
147         if j == u:
148             indicesforedges.append(i)
149
150 indicesforedges = list(set(indicesforedges))
151
152 finalcoloredge=[]
153 finalcoloredge=(len(e)-len(indicesforedges))*['1']
154
```

FIG. 13: Wavegraph visualization 3/4

```
154
155 for i in indicesforedges:
156     finalcoloredge.insert(i,'0')
157
158 #Making a list of the size of each node in order to visually compare the number of connections
159 sizes=[]
160 for i in range (0,len(nodeIDs)):
161     eachsize= newNodeID_anc.count(i)
162     sizes.append(eachsize)
163 arraysizes=np.array(sizes)
164 finalsize=10*(arraysizes+1)
165
166 #Now plot everything
167 G=nx.Graph()
168 G.add_nodes_from(nodeIDs)
169 G.add_edges_from(e)
170
171 #Size of figure and pixel resolution, for resolution set dpi=100 inside
172 fig=plt.figure(figsize=(30,30))
173 axes=fig.add_subplot(111)
174 axes.set_axisbelow(True)
175 plt.grid()
176 #print("Nodes of graph: ")
177 #print(G.nodes())
178
179 #Parameters to draw
180 #For "node_color" one can choose an indivial color by writting "finalcolor" or choose for the color to be a function of the
181 # window by writting "dimensionsarray", for the "edge_color" one can choose between seeing all the edges
182 #(by simply setting it equal to a color e.g 'r' or one can choose to only see the connections of the nodes of a specific
183 # window by choosing "finalcoloredge"
184 #One can view the nodeID's by setting "with_labels" equal to "True"
185
186 nx.draw_networkx(G,with_labels=False,node_size=finalsize,pos=dic,font_size=15,node_color=dimensionsarray,width=0.5,
187                  edge_color='0')
188
189 #For the colorbar, this must be muted to be able to look at an individual window
190 nc=nx.draw_networkx_nodes(G,with_labels=False,node_size=finalsize,pos=dic,node_color=dimensionsarray)
191 plt.colorbar(nc)
192
193 #Title
194 plt.title('Wavegraph for all windows',fontsize=30)
195 #Font size
196 plt.xticks(fontsize=20)
197 plt.yticks(fontsize=20)
198 #Frequency of ticks
199 plt.xticks(np.arange(min(times), max(times), 0.2))
200 plt.yticks(np.arange(min(logfrequencies), max(logfrequencies), 0.1))
201 plt.xlabel('time (s)', fontsize=20)
202 plt.ylabel('log(frequency(Hz))', fontsize=20)
203 #Show and save
204 plt.savefig("TheNodeGraph.png")
205 plt.show()
206
```

FIG. 14: Wavegraph visualization 4/4