Chapter 6

Data Analysis Pipeline and Tuning

In this chapter we describe the standard pipeline used to analyze data from GW detectors, such as the LIGO detectors, for GW signals from CBC. In addition, we will also describe the tuning that was done for the Search for Low Mass CBCs in the First Year of LIGO's Fifth Science Run (S5) Data, referred to as "this search."

The name of the executable that generates the standard pipeline is the Hierarchical Inspiral Pipeline Executable (HIPE); and we colloquially refer to this portion of the pipeline as the HIPE pipeline. The goal of the HIPE pipeline is to distill GW data in the form of a time series of strain data into a collection of candidate GW triggers. The HIPE pipeline is generic enough that it is used for many different matched filtering searches including searches for GW signals from low mass CBC, triggered CBC searches (i.e., GRBs), high mass CBC, CBC whose component objects are spinning, and black hole ringdowns.

The HIPE executable, LALAPPS_INSPIRAL_HIPE, generates a directed acyclic graph (DAG) of computational jobs that are run by the Condor High Throughput Computing environment, where a DAG, as described by the name, is a collection of jobs with interdependencies such that there are no loops. HIPE reads an input file containing all of the arguments to invoke at different stages of the pipeline as well as the locations of files containing lists of times associated with individual detectors' operating status. The time files for each detector included for this search were times the detector was in "science mode" (i.e., when the detectors were operating nominally), which we call *analyzed time files*, and four sets of time files indicating increasingly severe excursions from nominal

operation, which we call *category veto files*.

Using these inputs, HIPE first computes all of the times an individual detector was in science mode for 2064 continuous seconds. Because of the way our filtering is done, HIPE throws out the beginning and end 72 seconds in order to calculate the "analyzable times". It then takes intersections of these times across different detectors to obtain different kinds of "coincident times" during which two or more detectors were in operation. Finally, HIPE sets up job dependencies across the different stages of the pipeline and produces the DAG to be run by Condor.

The different stages of the HIPE pipeline employed in this search were:

- template bank generation (detailed in section 6.3);
- first-stage single-detector trigger generation (detailed in section 6.4);
- first-stage coincident trigger generation (detailed in section 6.5);
- template bank regeneration (detailed in section 6.6)
- second-stage single-detector trigger generation (detailed in section 6.7);
- second-stage coincident trigger generation (detailed in section 6.8).

The template bank generation stage generates a bank of templates in some specified portion of the parameter space. Those templates are then used for the first-stage single-detector trigger generation by performing matched filtering of the templates against the data, producing a list of single-detector triggers. The first-stage single-detector triggers from different detectors are then combined and used to search for coincident triggers in the first-stage coincident trigger generation by identifying triggers that are coincident across detectors in parameter space, including time. The first-stage coincident triggers are converted back into single-detector template banks for a second round of matched filtering in the template bank regeneration stage. This new template bank is used for the second-stage single-detector trigger generation by matched filtering the data against the templates again, this time also calculating and vetoing triggers based on signal-based vetoes. Finally, the second-stage single-detector triggers are used for second-stage coincident trigger generation, where

triggers occurring during times in the category veto files are thrown out, coincident triggers are found, and more signal-based vetoes are calculated and used. The output of this stage of the pipeline is the final output of the HIPE pipeline. The pipeline flow is represented by figure 6.1.

Multiple stages of matched filtering and coincidence occur because the calculation of the signal-based vetoes in the second-stage single-detector trigger generation is very compute intensive. In order to speed the search up, we only compute these signal-based vetoes on triggers that have already passed coincidence.

The follow sections will go through each of these stages in detail, but first let us describe the blinding and tuning procedure we use and how we test our pipelines using software injections.

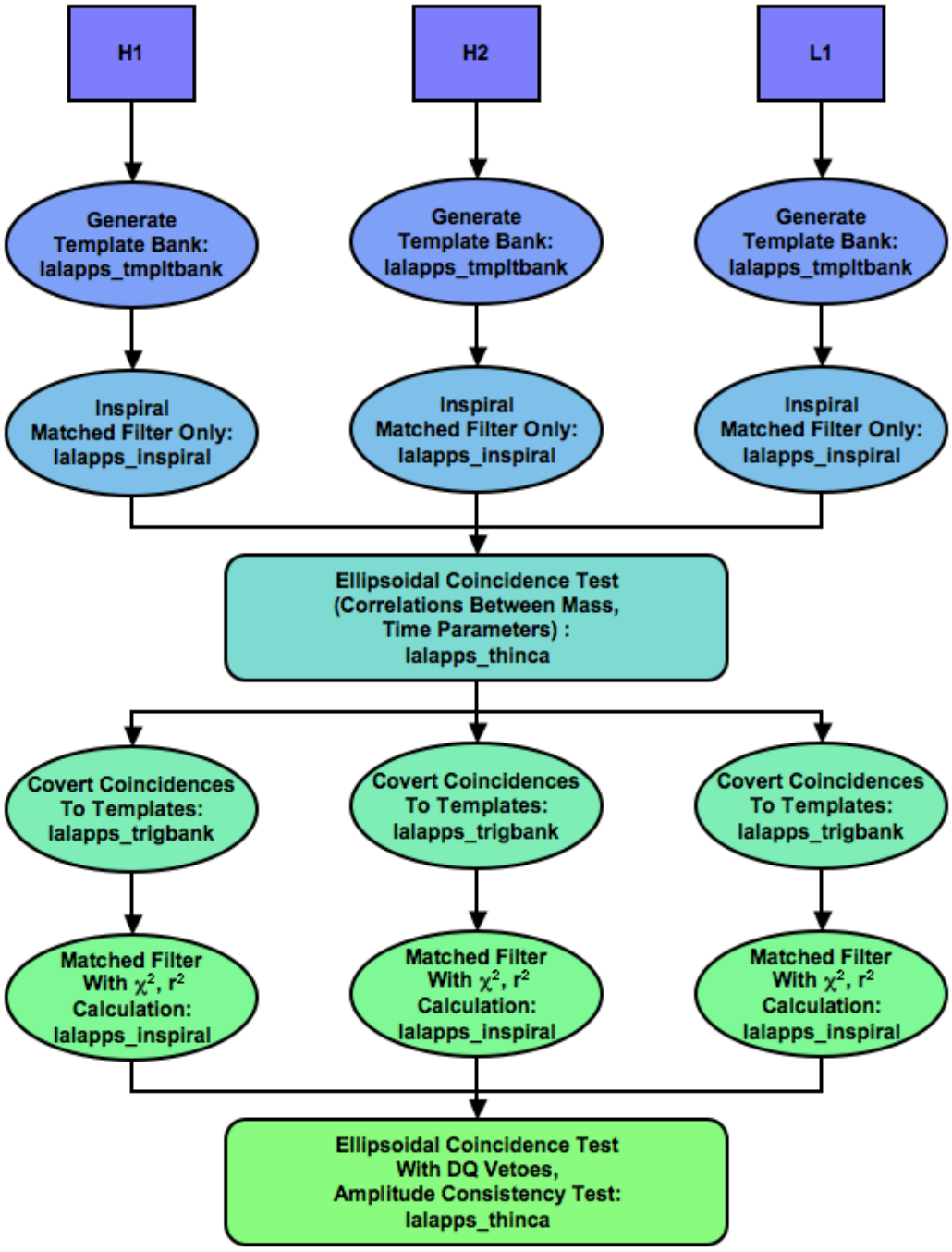

Figure 6.1: The HIPE Pipeline
A flowchart showing how GW detector data are analyzed in the HIPE pipeline.

6.1 Blinding the Analysis Pipeline

One of the goals of searching for GWs is to give a statistical probability that the triggers seen are real signals or due to background. This requires that the search pipeline's tuning not be informed by the *in-time* triggers that are present in the data. This can be achieved by *blinding* the data during the tuning process.

To do this, we only look at coincident triggers associated with 1) *time-shifted data* in which the detector data streams streams have been slid against each other such that no true GW signal could produce coincident triggers in that time shift (this data estimates the background of the pipeline), 2) *software injected data* in which GWs have been artificially introduced into the data in software (we use these to measure the efficiency of our pipeline at detecting real signals), and 3) *playground in-time data* that we have defined to be 600 of every 6370 seconds of data, which we use to check our pipeline to make sure nothing has gone awry in our analysis (we search for GW signals in this data but exclude it from statistical statements resulting from the search). Using these triggers, we tune the parameters of the pipeline and construct a detection statistic to rank the importance of the coincident triggers such that the time-shift coincident triggers are suppressed while the coincident triggers associated with software injections are emphasized.

Only at this point do we look at the in-time coincident triggers ("open the box"), unblinding the analysis to the part of the data in which putative GW detections can be found. In this way we avoid any bias in establishing the detection statistic that can come from tuning the search parameters after candidates are examined, which could lead to subjectively elevating or "killing" such candidates. However, we choose to apply an unavoidably subjective follow-up procedure (described in chapter 10) to develop sufficient confidence in a potential detection candidate to warrant the announcement of the first discovery of GWs.

6.2 Simulated Waveform Injections

In order to measure the efficiency of our pipeline to recovering GW signals from CBC, we inject several different PN families of waveforms into the data and check to see the fraction of signals that are recovered. The different waveform families used for injections in this search include GeneratePPN computed to Newtonian order in amplitude and 2PN order in phase using formulae from reference [3], EOB computed to Newtonian order in amplitude and 3PN order in phase using formulae from references [4, 5, 6, 7], PadéT1 computed to Newtonian order in amplitude and 3.5PN order in phase using formulae from references [8, 9], and SpinTaylor computed to Newtonian order in amplitude and 3.5PN order in phase using formulae from references [10] and based upon references [11, 12, 3, 13, 14, 15, 8, 16]; using code from reference [126]. Each of these families except for SpinTaylor ignores the effects of spin on the orbital evolution.

The parameters of the signals injected for each of the families are chosen from either an astrophysical distribution or a nonastrophysical distribution. The parameters that are astrophysically distributed include:

- the sky location (right ascension, declination) is distributed uniformly on the surface of a sphere, since we expect no preferred direction in the sky for extragalactic GW sources;
- the inclination angle (ι) where $\cos\iota$ is distributed uniformly between 0 and 1, since there is no preferred orientation of sources with respect to the observer's line of sight;
- the polarization azimuthal angle (ψ) distributed uniformly between 0 and 2π, since there is no preferred orientation of sources with respect to the observer's orientation;
- (for the SpinTaylor waveform family) the component objects' spin orientations relative to the initial orbital angular momentum are individually distributed uniformly on the surface of a sphere. No correlation is expected amongst the three orientations of the component objects' spin orientations and the orientation of the orbital angular momentum for captured binaries. However, there may be correlations for primordial binaries.

The parameters that are nonastrophysically distributed include:

- the total mass is distributed uniformly between 2 and 35 $M_\odot$, since there is no reliable theory for predicting the distribution in total mass, and we wish to measure the CBC rate for a range of masses;

- the mass ratio $q = m_1/m_2$ is distributed uniformly between 1 and the maximum value such that $m_1, m_2 > 1M_\odot$, since there is no reliable theory for predicting the distribution in mass ratio;

- the physical distance D is distributed uniformly in $\log_{10} D$, since we want to test our pipeline on a large range of signal amplitudes;

- (for the SpinTaylor waveform family) the component objects' unitless spin magnitudes $\hat{a} \equiv (cS) \,/\, \left(Gm^2\right)$ are individually distributed uniformly from 0 to 1, since there is no consensus on the physical distribution of the component objects' spin magnitudes nor the correlations between the component objects' spin magnitudes.

6.3 Template Bank Generation

The search we are performing for GW signals from CBC is a matched filter search where we compare the data against a model signal waveform to see how much the data looks like the model. In this case the model, which we call a "template," is a gravitational waveform calculated from PN theory for a CBC of a certain mass (as discussed in chapter 2). In section 5.2.2 we discussed theoretically the generation of a template bank with the use of a metric on the parameter space. In this section we will go over the specific choices that were made in the construction of template banks used in this search.

Based on the goals of the search, we know the masses of the templates should cover the range of total mass from 2 to 35 $M_\odot$with a minimum component mass of 1 $M_\odot$. A factor that governs the total number of template is that we are willing to lose at most 10% of the signals. A mismatch due to the discreteness of the template bank reduces the SNR we can recover, and thus the distance to which we can see, by a factor of the mismatch. Assuming an signal distribution uniform in volume,

this translates the 10% tolerance in signal recovery to a match of $(90\%)^{1/3} = 97\%$ or a 3% mismatch.

The next choice we make is which waveform approximation is used to generate the templates. In interests of speed and simplicity of code, we choose to use 2.0 restricted PN templates, which are second PN order in the phase evolution and Newtonian order in the amplitude, generated in the frequency domain using the SPA. These choices are validated by studies that show these templates can recover time domain signals of different orders and different approximant families [127, 124, 128].

The final items we must choose in order to generate a template bank are the frequency range of our calculations and the calculation of the PSD of the data, S_n. The LIGO noise varies over many orders of magnitude for the frequency range we are interested in. The lowest mass signals we are searching for extend from arbitrarily low frequencies up to a few kHz. Because of the steep slope of the seismic noise in the detectors starting at 40Hz and below, we choose to suppress the data below 30Hz using two rounds of eighth-order Butterworth high-pass filters, and then analyze data above 40 Hz. This is shown in the spectra of the data after the different stages of filtering in figure 6.2. The gap between where our high-pass filters end and where we start analyzing data is to ensure we are not suppressing the signals in the band we analyze.

To choose our high frequency cutoff, we look at the frequencies our signals end on, a few kHz and below, and we choose 2048 Hz as our high frequency cutoff. In order to reduce the amount of data we work with, we down-sample the data from 16384 samples per second to 4096 samples per second, applying low pass filters above the Nyquist frequency of 2048 Hz in order to prevent aliasing of power from frequencies above this to low frequencies.

We calculate S_n using a 2048-second-long segment of data. This is broken up into 15 overlapping chunks of 256 seconds, which we Fourier transform to calculate 15 PSDs. S_n is then constructed by taking the median of these in each frequency bin. This PSD is used in the metric calculation of the chosen parameter space and, later on, in our matched filtering.

With the above procedures, we end up with ∼5000 templates covering the parameter space we are interested in. An example template bank can be seen in figure 6.3. The essentially flat metric in $(\tau_0,\ \tau_3)$ (discussed in chapter 5) produces an essentially uniform hexagonal spacing of templates

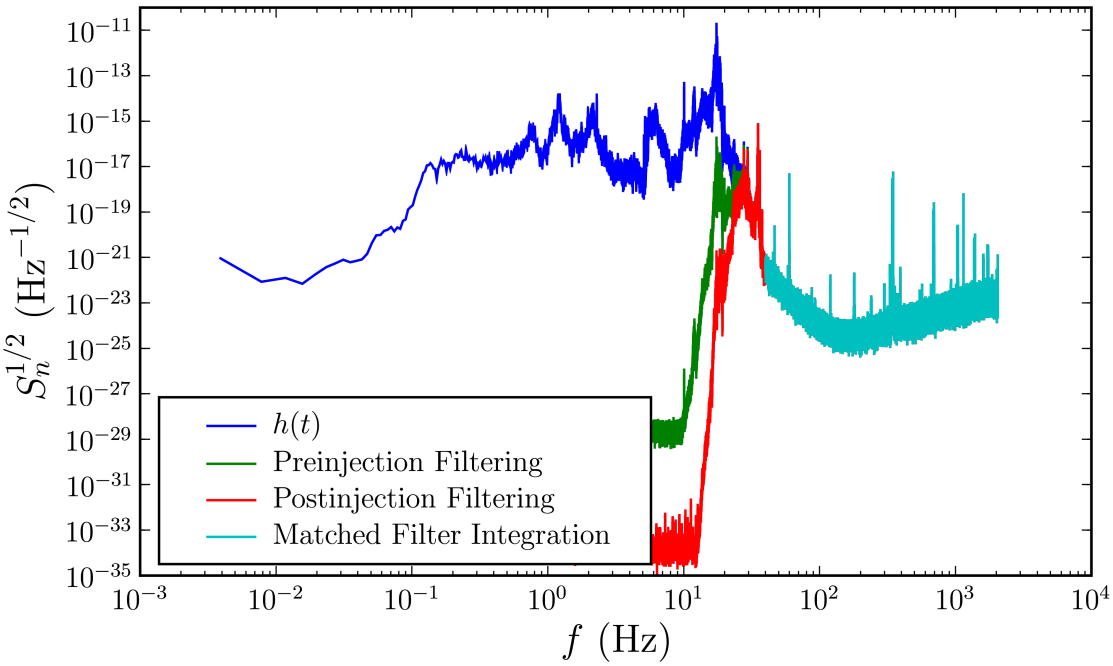

Figure 6.2: Data Preprocessing
Data spectra after the different levels of filtering we perform prior to matched filtering.

in that space, and a nonuniform spacing in $(m_1,\ m_2)$ space. Conversely, the straight bounds of the bank in $(m_1,\ m_2)$ space results in current boundaries in $(\tau_0,\ \tau_3)$ space.

6.4 First-Stage Single-Detector Trigger Generation

The first-stage single-detector trigger generation is the first round of matched filtering we perform on the data. In this stage we perform the same data conditioning as described in the template bank section with an addition step of injecting signals to the data when performing software injections. In that case, after the first round of high-pass filtering, the injection waveforms are added to the data, as noted in figure 6.2. The data conditioning continues after this as before with the second round of high-pass filtering and then resampling.

Once we get to this stage, we perform the matched filtering as described in chapter 5 in order to obtain an SNR time series for each template. Since the templates we are filtering with have a specific duration, there will be wrap-around effects near the boundaries of each chunk where the

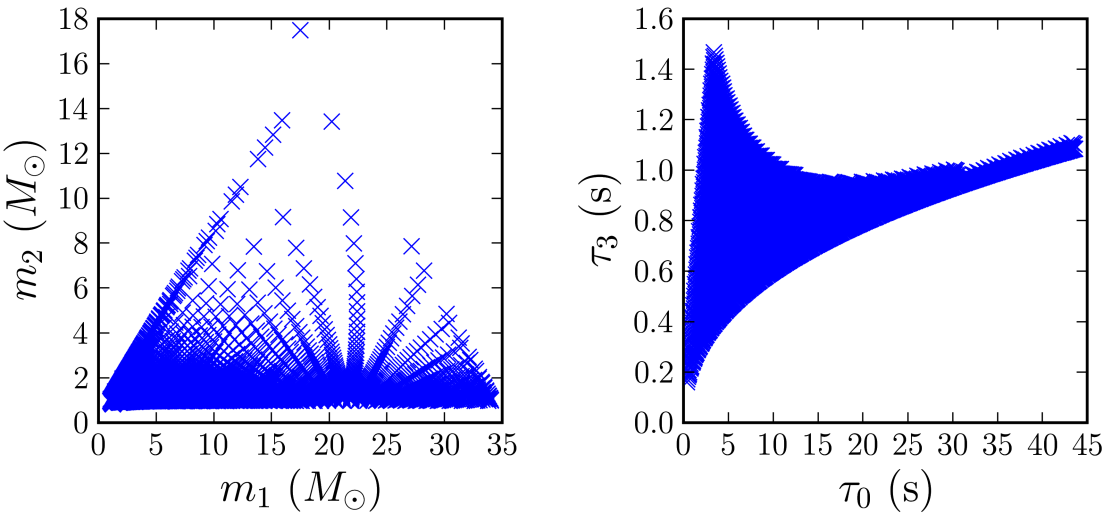

Figure 6.3: Sample Template Bank
A sample template bank is shown in both (m_1, m_2) space and (τ_0, τ_3) space.

SNR time series will be corrupted. We set the 256-second chunks to overlap by 128 seconds on each end, enabling us to throw away triggers within 64 seconds of each boundary containing the corrupted portion of the time series.

The SNR time series is then used to generate single-detector triggers by finding times when the SNR exceeds a threshold value of 5.5 for this search. This may happen for many adjacent points in time, so to reduce the amount of triggers we generate, we cluster the time series over the length of the template t_{tmplt} using a sliding window. This is implemented in the following way:

1. start at the beginning of the time series;
2. find the next time the SNR crosses the threshold;
3. generate a trigger, which records that time and SNR ρ^*;
4. search the time series over the next t_{tmplt} seconds for the first time when $\rho > \rho^*$;
5. if there exists such a time, update the trigger to contain the new time and SNR, then repeat step 4;
6. if there does not exist such a time, store the trigger in a trigger list and, if there is additional

time in the time series, repeat step 2.

After this procedure, we have a list of single-detector triggers from a single template. This process is repeated for each template in the bank. Once we are done filtering the data through all of the templates, we perform an additional round of clustering before we write these triggers to file and end this stage. In this final round of clustering, we cluster triggers coming from different templates when they satisfy some requirements. As we have noted before, adjacent templates have a match close to 1 in order for a signal falling between templates to have a mismatch with the nearest template of less than 3%. This results in templates near a particular template that produces triggers having a good chance of also crossing SNR threshold and producing triggers.

An algorithm called "trigscan" was devised that clusters these triggers across the template bank using the metric calculated during the template bank generation [129]. In generating the template bank, the portions of the metric that were used are those that ignored the time dimension of parameter space, as mentioned in chapter 5. In contrast, trigscan employs the full 3D metric to do the clustering. Trigscan works using the following algorithm:

1. sort the trigger list to be time ordered and start with the first trigger and use it to seed a cluster of one trigger;

2. create an error ellipse of size $\epsilon_{\text{trigscan}}$ around the trigger using the metric at that point in $(\tau_0,\ \tau_3,\ t)$ space;

3. for all triggers with a time $t \in (t_i - T, t_i + T)$, where t_i is the time of the current trigger and T is some time window, create error ellipses of size $\epsilon_{\text{trigscan}}$ at those points using their values for the metric;

4. for any triggers whose error ellipse overlaps the current trigger's, add them to the current cluster;

5. repeat steps 2–4 for any trigger added to the cluster;

6. when no additional triggers can be added to the cluster, save the trigger with the largest SNR

in the clustered trigger list;

7. if there is a next trigger in time, use it to create the next cluster of one trigger and repeating steps 2–5

We decided to use a trigscan metric scaling value of $\epsilon_{\text{trigscan}} = 0.06$. Ignoring the time dimension for a moment, let us assume we are in flat space with hexagonally spaced templates where the furthest distance any point can be from a template is 0.03. In this case, the templates have a separation of $2 \times 0.03 \cos(\pi/6) \approx 0.052$. If we are to create circles around each template and expand them to radius 0.06, templates slightly more than twice the templates separation distance will have circles that intersect. This can be seen pictorially in figure 6.4. Since the $(\tau_0,\ \tau_3)$ space in which we have chosen to place are templates is almost flat, our chosen value of $\epsilon_{\text{trigscan}} = 0.06$ allows triggers occurring at the same time (i.e., ignoring the time dimension of the 3D space) to be clustered when they are from next-to-nearest-neighbor templates in the template bank.

The trigscan-clustered trigger list is then saved to an xml file where each trigger contains information such as the GPS time it occurred, the SNR (ρ), the effective distance (D_{eff}), the phase at coalescence (ϕ_0), the mass parameters ($m_1,\ m_2,\ M,\ \eta,\ \mathcal{M}$), the chirp time parameters ($\tau_0,\ \tau_2,\ \tau_3,\ \tau_4,\ \tau_5$), the length of the template (t_{tmplt}), and the six metric components of the 3D metric at that point. Additional information associated with the time analyzed that is also calculated and saved to file is the horizon distance (equation (5.38)) for the detector using S_n for that 2048-second segment.

6.5 First-Stage Coincident Trigger Generation

A true GW signal from a CBC should affect all detectors around the world, and indeed, to verify a GW detection, it will have to be found by multiple means (i.e., associated electromagnetic, neutrino, or GW observation). For this search, we are only considering data from the LIGO detectors, so we rely on finding coincident GW triggers from different detectors.

Additional information useful in evaluating the performance of the pipeline is the background rate of triggers. The background is assumed to be due to accidental coincidences, which can be

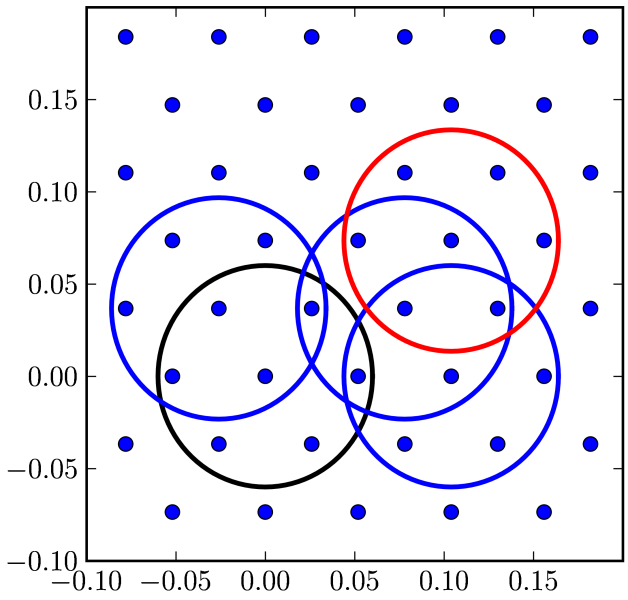

Figure 6.4: Trigscan Example
Templates are plotted as blue dots, which are spaced such that the furthest distance any point can be from a template is 0.03. We draw a black circle of radius 0.06 around a chosen template. Blue circles are drawn around three example templates that can be trigscan clustered with the chosen template (i.e., the blue circles intersect the black circle). A red circle is drawn around a single template that is slightly too far to be clustered with the chosen template.

estimated by time shifts (described in section 6.1). This estimation fails for collocated detectors, such as H1 and H2, due to time correlated background arising from the shared environment and crosstalk between the detectors (e.g., stray optical signals or electric control signals). Because of this, we can not reliably estimate the background in this way, thus H1H2 coincident triggers are excluded from this search. As these effects should only occur between collocated detectors, there is no in-time correlation between the Hanford and Livingston detectors.

Below, section 6.5.1 describes how the coincidence algorithm is executed, while section 6.5.2 describes how the time shifts are performed.

6.5.1 Coincidence Algorithm

The coincidence algorithm that has been developed for finding coincident triggers between multiple detectors is called The Hierarchical Inspiral Coicidence Algorithm (THINCA) and we commonly

refer to this stage of the pipeline as the first THINCA stage. Here we take as input triggers from up to 3600 seconds of time when a unique set of detectors was generating triggers. Potential coincident triggers could lie across the boundary of two segments and thus not be recorded, however we find a negligible loss in efficiency due to this effect. The coincidence test used for this search in THINCA is similar to the trigscan clustering algorithm described above, called the "ellipsoidal" THINCA or "e-thinca" coincidence test. The main difference between the two is that trigscan builds up *clusters* of triggers from a *single detector*, while the e-thinca test checks whether triggers from *two* detectors are coincident *without agglomerating* coincident triggers into clusters.

First we will describe the e-thinca coincidence test before moving on to the full coincidence algorithm. The e-thinca test for two triggers from different detectors is as follows

1. create an error ellipse of size $\epsilon_{\mathrm{e-thinca}}$ around the triggers using their respective metrics;
2. if their error ellipses overlap, return yes, otherwise return no.

With the coincidence test defined, we now describe the full coincidence algorithm. It is as follows:

1. create a time sorted trigger list from all the triggers of the multiple detectors, and start with the first trigger;
2. for all triggers with a time $t \in (t_i, t_i + T)$, where t_i is the current trigger and T is some time window. If the trigger is from a different detector, perform the coincidence test between the two triggers;
3. if coincidence test return yes, create a coincident trigger in a coincident triggers list and link both single-detectors triggers to this coincident trigger;
4. repeat 2-3 for the next trigger in the list until end of list;
5. loop over coincident triggers, if all the single-detector triggers in the current coincidence are coincident with an additional trigger from a different detector, add that trigger to this coincidence;

6. loop over coincident triggers and remove coincident triggers that are subsets of higher-order coincident triggers.

After this procedure, the list of coincident triggers is saved to an xml file where the single-detector triggers of a coincidence all have the same event_id as well as all the information saved during the single-detector trigger generation.

6.5.2 Background Estimation: Time Shifts

As mentioned previously, time shifts are performed between different detectors in order to estimate the amount of background triggers we expect from the pipeline. These time shifts are chosen such that any coincidences coming from the time shifted data could not have come from a real GW signal. To ensure this, we have chosen to create a vector that we multiply by a shift number in order determine the amount of time to shift each detector. The vector contains a unit time shift for each detector, which must be unique.

For this search we have chosen the vector

$$v = \{0, 10, 5\} \ , \tag{6.1}$$

which we associate with the detectors H1, H2, and L1 respectively. This is used for 100 different time shifts ranging in shift number from -50 to 50, skipping 0. The maximum number of time shifts we could do with these shift vectors is 360, however we have found 100 to be adequate to estimate the false alarm rate for the loudest in-time triggers in the past. Additional time shifts could be performed with smaller shift vectors, however there is a time correlation for triggers coming from real signals that must be avoided.

As we said before, THINCA takes in as input up to 3600 seconds worth of triggers from a unique combination of multiple detectors. Using the above information, we find that when we shift triggers from different detectors against each other, we can end up with up to 2000 seconds where there are no H1 triggers to search for coincidence with H2 triggers. In order to avoid this but keep with the

3600-second limit, we have chosen to wrap the triggers on a 3600-second ring defined by the original start and end times of the coincident segment. In this approach, when a trigger's shift time becomes greater than the end time of the ring, we subtract the duration of the ring from the trigger's time, which moves it to the beginning of the ring. Conversely, when a trigger's shifted time becomes less than the start time of the ring, we add the duration of the ring to the trigger's time, which moves it to the end of the ring.

Once we have shifted the triggers in time, we can apply the above coincidence algorithm to produce coincident triggers for the individual time shifts. At the end of the algorithm, before we save the coincident triggers to file, we unshift the triggers' times so that they return to their original times.

There a several nice features of this approach. The first is that it keeps the amount of analyzed time in each time shift the same as the in time. In addition, this approach limits the maximum time offset between time-shifted coincident triggers to 3600 seconds. Since we expect the detectors' behaviors to be roughly stationary over this time period, this more accurately estimates the background for the in-time coincident triggers.

As discussed in section 6.1, there are noise correlations between H1 and H2 that bias the background estimation for H1H2 coincidences, which we find to also have an effect H1H2L1 coincidences. We have tested using a background estimation for H1H2L1 coincidences in which we shift H1 and H2 triggers by the same amount and found this to correct this bias. As this was done late in the analysis, it was not folded into the final results, however it has become a standard procedure for the other searches we are performing in the S5 data.

6.5.3 Coincidence Tuning

The threshold in the algorithm that controls the maximum e-thinca value ϵ we accept for coincidence we refer to as the e-thinca threshold ϵ_*. Larger values of ϵ_* correspond to accepting larger error ellipses for determining coincidence. We tuned the e-thinca threshold by looking at time-shifted coincident triggers and triggers associated with software injections. Figure 6.5 shows the fraction

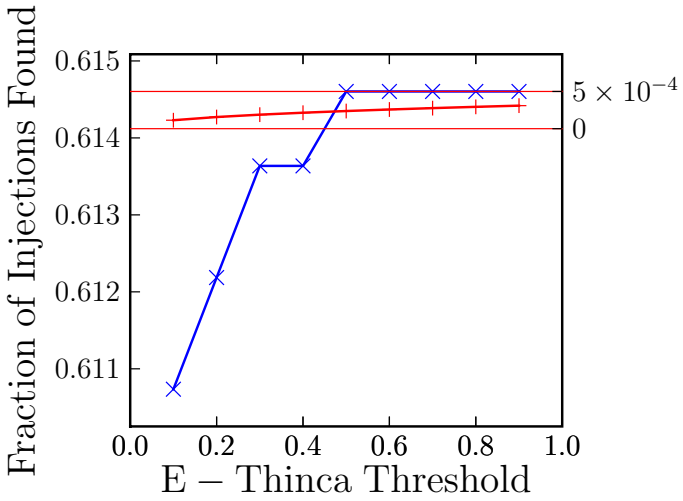

Figure 6.5: E-Thinca Tuning
The percentage of injections recovered (blue and left y-axis) and the expected percentage of injections recovered due to background coincidences (red and right y-axis) as a function of the e-thinca threshold. We have offset the blue and red curves for ease of comparison. The horizontal red lines show the efficiency gain per additional found injection. We see that over the whole range of e-thinca thresholds, there is less than one expected injection due to the background.

of injections recovered as a function of the e-thinca threshold. There we plot both the fraction of injections recovered and the expected fraction of injections whose time windows overlap time-shifted coincidences F_{false}, which is given as

$$F_{\mathrm{false}} = \frac{N_{\mathrm{time-shifts}} T_{\mathrm{window}}}{N_{\mathrm{inj}} T_{\mathrm{time-shifts}}} \, , \tag{6.2}$$

where N_{inj} is the number of injections made, $T_{\mathrm{inj}} = N_{\mathrm{inj}} t_{\mathrm{window}}$ is the total amount of time we were searching for injections, t_{window} is the window used to determine whether or not an injection is recovered, $N_{\mathrm{time-shifts}}$ is the number of time-shift coincidences for a given e-thinca threshold, and $T_{\mathrm{time-shifts}}$ is the total amount of time analyzed for time-shifted coincidences.

Figure 6.5 shows that for injections into H1H2L1 time, we find that the last additional injection is found at an $\epsilon_* = 0.5$ and that there is less than one expected injection due to the background at that value. Because of this, we decided to use $\epsilon_* = 0.5$ as the threshold for this search.

6.6 Trigger to Template

In this stage of the pipeline, we create a new template bank from triggers found in coincidence between detectors. This is done in the following manner:

1. use the coincident segments that a particular 2048-second single-detector segment participated in;

2. keep triggers from only that detector;

3. search through triggers keeping only one trigger for each combination of mass parameters;

Once this is done, we save the remaining triggers to an xml file for use as the template bank in the second-stage single-detector trigger generation.

One thing to note about this stage is that triggers from different single-detector segments can wind up as templates for nearby single-detector segments. This is due to the fact that coincidence segments can take in more than one single-detector segment from a given detector. But are we not trying to reduce the number of templates for the second, computationally expensive stage of matched filtering whereas this effect is causing an *increase* in the template bank? Even though this is the case, the increase in the template bank due to this effect is much smaller than the reduction we are trying to accomplish.

6.7 Second-Stage Single-Detector Trigger Generation

Here we describe the second-stage single-detector trigger generation. As noted above, for this stage we use the triggers found in coincidence with other detectors as the template bank for the second round of matched filtering. This requirement significantly reduces the number of templates with which we perform this stage of matched filtering. This reduction is necessary because in addition to performing the standard matched filtering we have described above, we also calculate two signal-based vetoes that test the consistency of the data to what we expect from a signal, one of which is computationally intensive. This veto requires an additional p inverse Fourier transforms, which

causes this stage to be the time-limiting stage of the pipeline.

However, there are also drawbacks to this two-stage approach. Adding in the second stage of the pipeline adds significant complexity to the pipeline. In addition, coincident triggers found after the first stage of coincidence may not be the same as after the second stage of coincidence. This adds ambiguity when trying to track specific triggers through the pipeline.

Before the veto calculations begin, and after they are finished, the same algorithms are used as in the first-stage single-detector trigger generation described in section 6.4. Below, in section 6.7.1, we describe how we tuned the $\chi^2$and r^2signal-based vetoes (described in sections 5.2.3.1 and 5.2.3.2 respectively).

6.7.1 Signal-Based Veto Tuning

As in the e-thinca threshold, the veto parameters for both the χ^2veto and r^2veto are tuned by observing how they respond to software injection triggers and how they respond to time-shifted coincident triggers. The goal of the vetoes is to find an area of parameter space where there is a separation of time-shifted coincident triggers from software injection triggers that can be removed, thus reducing the chance of a background trigger occurring while not affecting the chance of recovering a true signal.

The χ^2statistic, as described in section 5.2.3.1, has an expected behavior for software injection triggers that grows proportional to ρ^2, where ρ is the SNR. This can be seen in figure 6.6. We are trying to tune this search such that triggers associated with signals whose waveforms differ slightly from the templates are still recovered. The most extreme divergence from the template waveforms we hope to recover come from CBC waveforms whose component objects have significant spin angular momentum, as discussed in section 6.2. By injecting these waveforms in as software injections, we find that by loosening the χ^2veto parameters as compared to other injection waveforms, we are still able recover these spinning signals. This is shown in figure 6.7. The parameters for the χ^2veto that were chosen for this search were a mismatch parameter of $\delta = 0.2$ and a threshold of $\xi^2 = 10.0$.

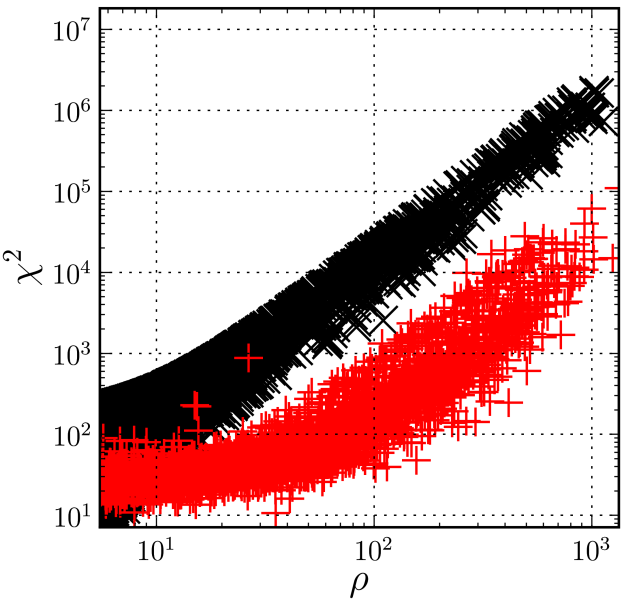

Figure 6.6: χ^2Tuning
χ^2values versus SNR for triggers associated with nonspinning software injections (red) and time-shifted coincident triggers (black). We can clearly see the ρ^2 dependence of the χ^2value for both categories of triggers.

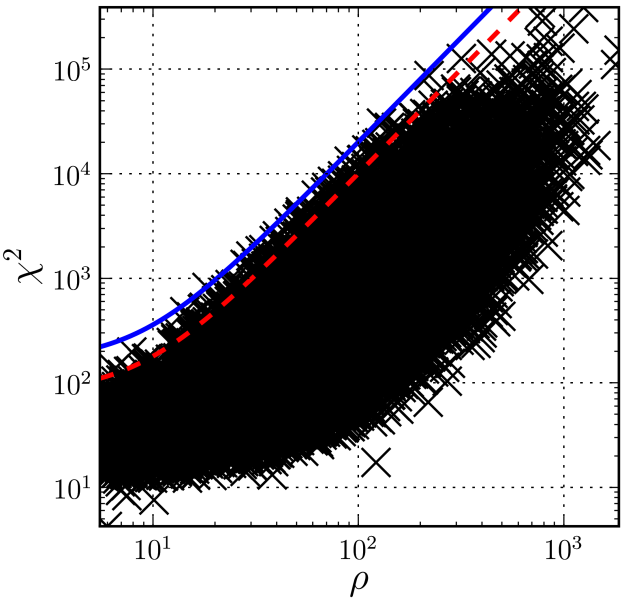

Figure 6.7: χ^2Threshold
χ^2versus SNR for software injections that include spin (black). The old χ^2threshold is plotted in red (tuned with nonspinning injections) along with the chosen χ^2threshold in blue (tuned with spinning injections).

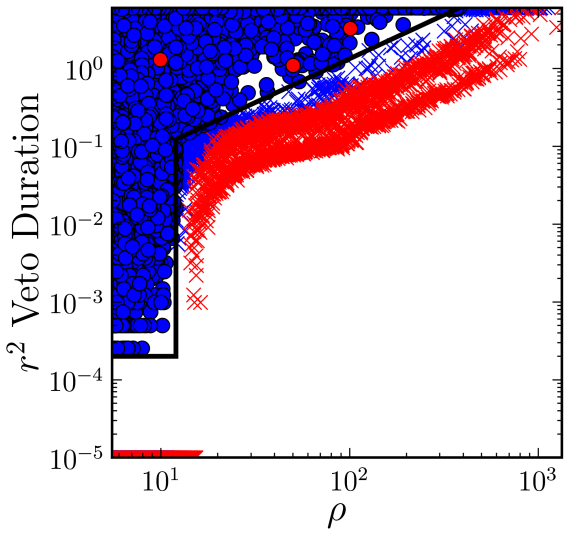

Figure 6.8: r^2Tuning
r^2veto duration versus SNR for software injection triggers (red) and time-shifted coincident triggers (blue). The black line denotes the r^2veto that was chosen for this search.

The r^2veto (described in section 5.2.3.2) also is expected to have a dependence on ρ because it derived from the χ^2statistic. Since there is a threshold below which the χ^2can fluctuate without contributing to the $r^2_{\rm duration}$ calculation, we expect the $r^2_{\rm duration}$ to be zero below a certain value of $\rho = \rho_*$. Above that value of ρ, since we have not modeled the $r^2_{\rm duration}$ dependence on ρ, we choose to veto triggers with an $r^2_{\rm duration} > A \times \rho^B$, where A and B are tuned in this search. For this search we have chosen $T = 6$ sec. and $r^2_* = 15.0$. With these values fixed, we have tuned ρ_*, A, and B using figures such as figure 6.8. Using this information, we have chosen $\rho_* = 12$, below which we veto triggers with $r^2_{\rm duration} > 0$, and above which we veto triggers with $r^2_{\rm duration} > A \times \rho^B$. We have also chosen $A = 7.5 \times 10^{-3}$ and $B = 1.12764$. All of these choices are loose enough such that the leave some space for signals to move due to noise fluctuations or waveform uncertainties without being vetoed.

In figure 6.8 there are several interesting features that require explanation. The main feature that jumps out is the bimodal distribution of software injection triggers. We have found a clean

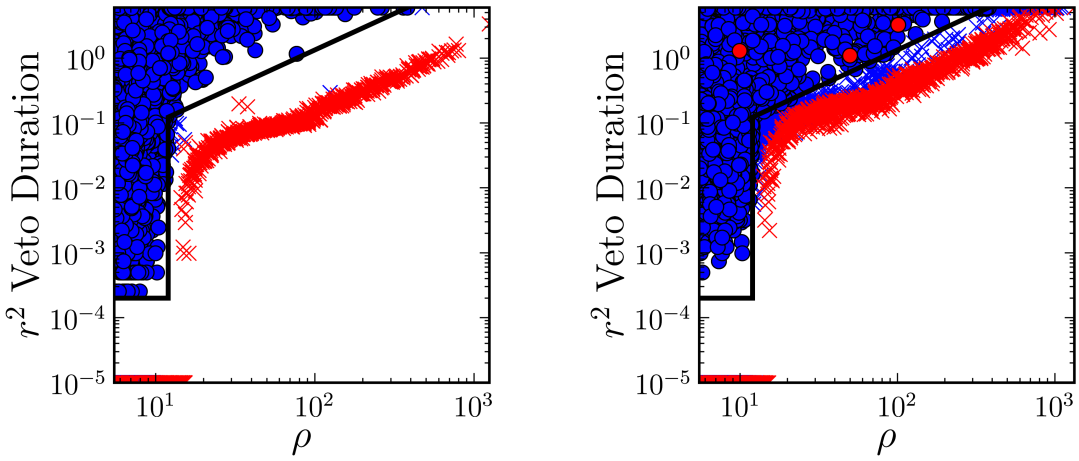

Figure 6.9: Different r^2Veto Populations
r^2veto duration versus SNR for software injections with total mass less than 20 $M_\odot$ (left) and greater than 20 $M_\odot$ (right).

break between to two distributions where the distribution with lower r^2_{duration} is from software injections with $M < 20M_\odot$ while the distribution with higher r^2_{duration} is from software injections with $M > 20M_\odot$, where M is the total mass of the system, as can be seen in figure 6.9. The total mass is used to define the frequency at which to stop the waveform calculation since the PN approximation breaks down at the frequency of the ISCO. From equation (2.38), the ending frequency f_{ISCO} corresponding to $M = 20M_\odot$ is $\sim 220Hz$, which is roughly when the f_{ISCO} is in the "sweet spot" of the detection band (i.e., where the noise is lowest).

Another feature in figure 6.8 is a handful of injections above the line denoting the veto. Closer inspection of these triggers showed there was a glitch several seconds before the injections. An example is seen in figure 6.10 showing the r^2time series for the same data stretch with and without the injection present. This figure also shows how there is nonzero r^2_{duration} expected for injections above a certain ρ.

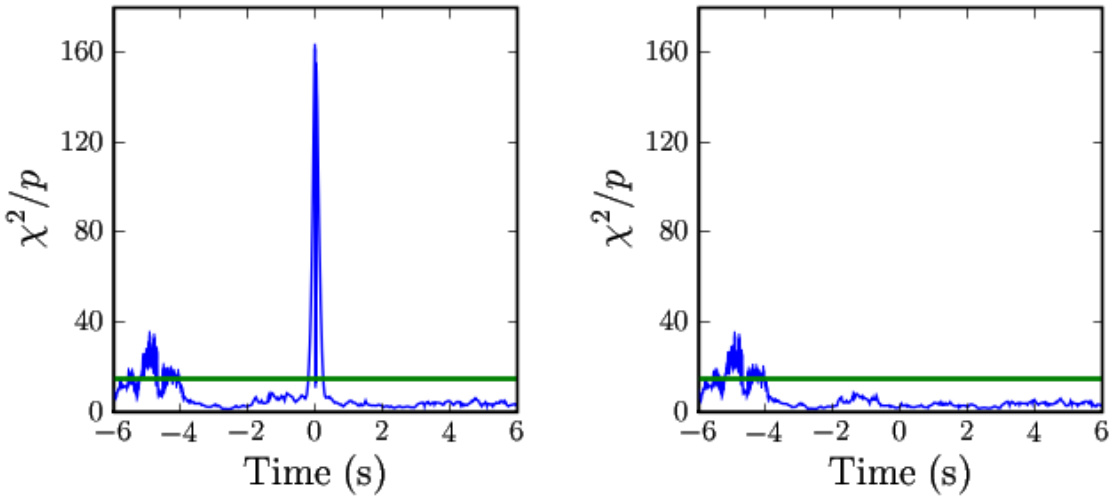

Figure 6.10: r^2Sample Time Series
The r^2time series for the same example stretch of data with (left) and without (right) a software injection present. The green line denotes the threshold value above which we count time samples in calculating the r^2veto duration. Comparison of the two plots reveal features that are due to the injection and present without the injection.

6.8 Second-Stage Coincident Trigger Generation

This is the final stage of the HIPE pipeline. We perform the second-stage coincident trigger generation using the same algorithm as in the first stage (described in section 6.5) with the addition of several vetoes. These different vetoes include vetoing times the instruments were not behaving nominally, what we call Data Quality (DQ) vetoes, as well as some signal-based vetoes using information from coincident triggers.

In the following sections, we discuss the use of DQ vetoes, and the tuning of two signal-based vetoes based on the collocation of the Hanford detectors, the H1H2 effective distance cut and the H1H2 consistency check, described in section 5.2.3.3.

6.8.1 Application of Data Quality Vetoes

While operating the LIGO detectors, we monitor many auxiliary channels that measure the state of the detectors or their local environment, in addition to the GW channel. We use these channels to determine whether or not we want to analyze data from the detectors at those times. This is based on how severe the excursions are from nominal observation mode, what we call "science mode," due to known artifacts introduced into the GW channel data by instrumental or environmental effects coupling to the GW channel. DQ flags are then defined as time segments during which these effects are present in the instrumental or environmental channels. Based on several considerations, we define several different categories of DQ flags for use as vetoes in this analysis.

Category 1 DQ flags are times when we know of a severe problem with the detector, bringing in to question whether the detector was actually in science mode. Some example cases are when calibration for the data is not present, or when loud vibrations were caused in the detector environment in order to test the response of the seismic isolation systems. We choose not to analyze these times because the noise from those times is too high for the data to be useful and we do not want it to affect our median PSD for adjacent times.

We then analyze the data, producing single-detector triggers for use in the rest of the categorization of the DQ flags. In the categorization we take into account several considerations. The first is

the "efficiency ratio." For this, we calculate the fraction of triggers from an individual detector that occur during a particular proposed DQ flag and divide this by the "dead time," which is the fraction of the time flagged by the DQ flag. When the efficiency ratio is much greater than one, this DQ flag is highly correlated with false trigger production and is thus a good candidate for eliminating excess triggers. Another criteria we use in the categorization procedure is the "use percentage" of a DQ flag. The use percentage is defined as the fraction time segments from a particular DQ flag that contain single-detector triggers. This value is a measure of how strongly a particular channel couples to the GW channel. Ideally, the use percentage for a good DQ flag veto would be one, meaning that background triggers always occurred whenever that particular DQ flag was on. Finally, we measure the "safety" of using these DQ flags as vetoes by comparing them with the times of hardware injections, which measure the response of the detector to a simulated GW signal. This helps to confirm that the DQ flags will not be correlated with real signals. We require all of the DQ flags used as vetoes to pass this safety check.

Category 2 DQ flags are defined as times when there is a reasonably well established coupling between the GW channel and an auxiliary channel, the flag has a high efficiency ratio, particularly with high SNR triggers, and when there is a use percentage of 50% or greater. An example is when any of the data channels in the length-sensing and control servo reach their digital limit. These are times when we do not trust the data from the detector, however the data were not bad enough to mess up the PSD calculation. We allow these times to be included in our segment generation, but remove triggers that come from these times while calculating bounds on the rate of CBCs.

Category 3 DQ flags are defined as times when the coupling between the auxiliary channel and the GW channel is less well established or when the use percentage is low, but we still find high efficiency ratio for the particular DQ flag. An example is when the winds near the detectors are over 30 Mph. We allow these times to be included in our segment generation, but, as with the Category 2 vetoes, remove triggers that come from these times while calculating bounds on the rate of CBCs.

Category 4 DQ flags are defined as times when the coupling between the auxiliary channel and the GW channel is not well established, when the use percentage is low, when the overall dead time

is several percent or greater, or the efficiency ratio is not much greater than one. An example is when nearby aircraft pass overhead. We allow these times to be included in our segment generation and also retain triggers that come from these times. This category is used in following up the loudest events from the search.

Summarizing the uses stated above, we do not analyze data vetoed by category 1 DQ flags, we veto but still follow up triggers in times defined by category 2 and 3 DQ flags, and we use the category 4 DQ flags as information when following up the loudest events from the search. These veto categories significantly reduce the SNR of outlying triggers, as can be seen in figure 6.11.

One could argue that the procedure used to define these vetoes biases the search since it uses in-time triggers. However, the triggers that are used are single-detector triggers *before* coincidence. There are many more triggers at this stage than after coincidence so that triggers associated with real GWs make up a small fraction and do not affect the tuning procedure to a significant degree. This procedure is in the process of being automated (as much as possible) to bias and to minimize the chance that GWs will be vetoed.

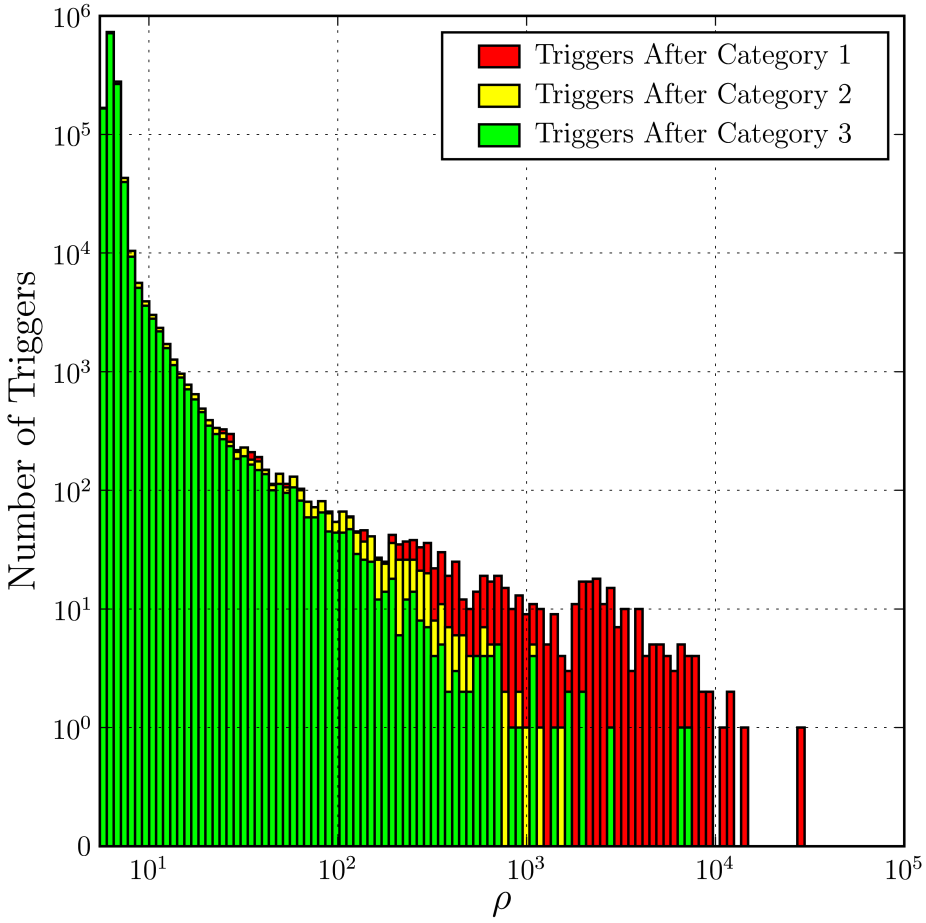

Figure 6.11: Effects of DQ Vetoes

Histogram of triggers for the H2 detector at the single-detector level before any coincidence requirement, clustered by the trigger with maximum SNR within 10 seconds, plotted after veto categories 1, 2, and 3. That is, category 2 vetoes remove the triggers in red, category 3 vetoes remove the triggers in yellow, and triggers in the green histogram remain.

6.8.2 Multidetector Signal-Based Veto Tuning

As described in section 5.2.3.3, there are additional signal-based vetoes we can perform for the collocated Hanford detectors. These vetoes, the effective distance cut and the amplitude consistency check, determine whether coincident triggers from two coaligned detectors satisfy certain amplitude conditions given the detectors aligned antenna patterns.

The veto parameter for both the effective distance cut and the amplitude consistency check is the same κ_* (cf., equation (5.37)), which is tuned by observing the fractional effective distance difference κ distribution for software injection triggers and time-shifted coincident triggers.

As described in section 5.2.3.3, κ should be near zero for triggers associated with signals (e.g., from software injections), while this is not necessarily the case for coincident noise triggers from time-shifted data. This can be seen in figure 6.12. From this figure, we see that the distribution of the time-shifted coincident H1H2 triggers peaks around $\kappa \sim 0.7$. We choose our threshold value to be $\kappa_* = 0.6$. At this value we veto the majority of the time-shifted coincident triggers, while vetoing a very small fraction of the software injections.

In figure 6.12 there is an interesting feature in κ plotted versus H1 SNR for the time-shifted coincident triggers. Here we see strip of time-shifted coincident triggers that extends up to high SNRs. We have found that this strip is consistent with what you would expect if you had found an H1 trigger coincident with an H2 trigger with an SNR of 5.5, while H2 had a horizon distance of 16 Mpc and H1 had a horizon distance of 32 Mpc. This situation is shown with the thin black line.

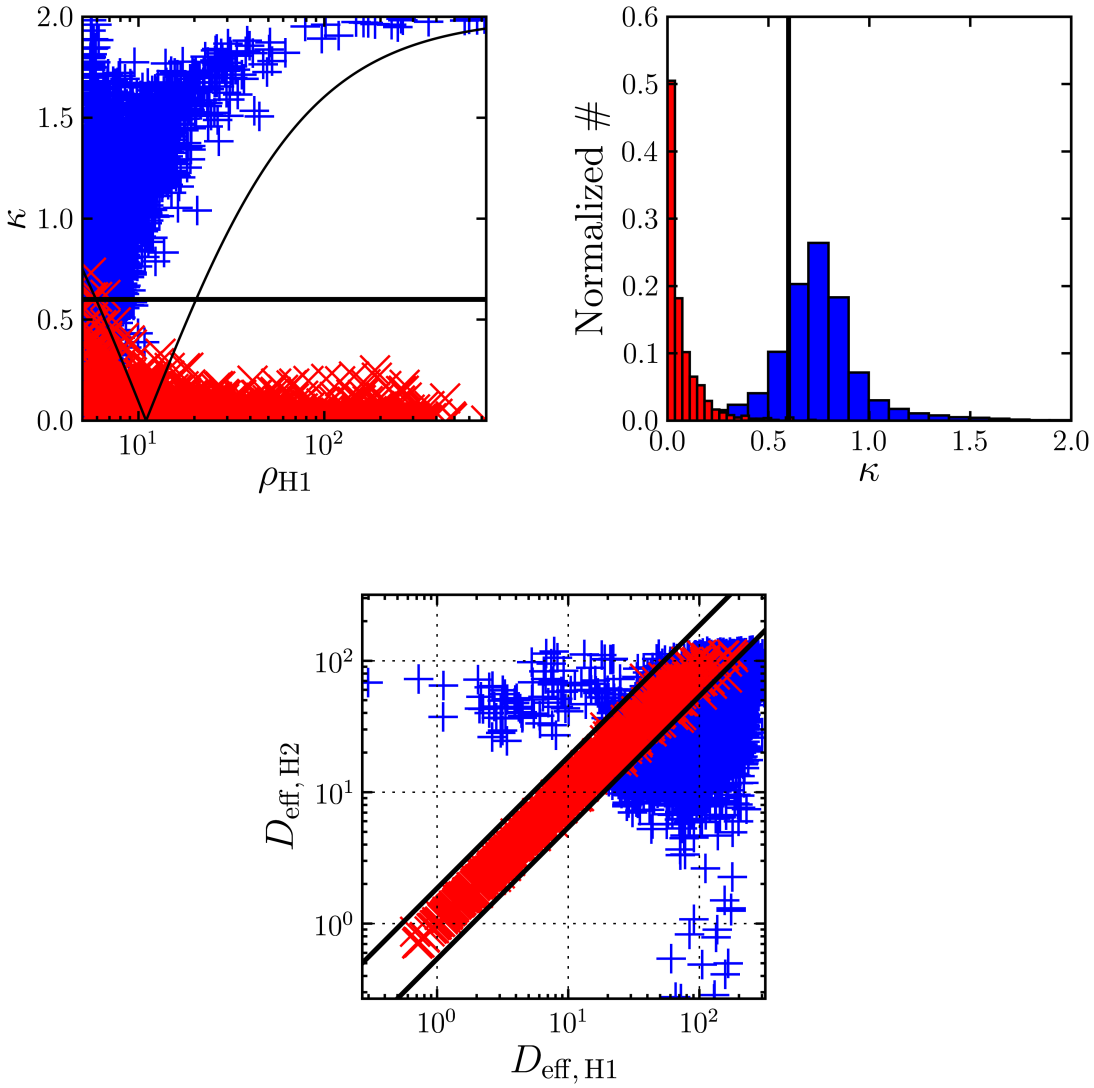

Figure 6.12: H1H2 Effective Distance Cut
The fractional effective distance difference κ versus the H1 SNR (top left), the histogram of κ (top right), and the H2 effective distance versus the H1 effective distance (bottom) for time-shifted coincident triggers (blue) and software injection triggers (red) in H1H2L1 time. The thick black lines show the veto value of $\kappa_* = 0.6$ and the thin black line shows a fit discussed in the text.