CCBIAS: AN EVENT DETECTION OPTIMIZATION FRAMEWORK

by

THEO FARIDANI

A THESIS

Presented to the Department of Physics
and the Robert D. Clark Honors College
in partial fulfillment of the requirements for the degree of
Bachelor of Science

June 2019

An Abstract of the Thesis of

Theo Faridani for the degree of Bachelor of Arts
in the Department of Physics to be taken June 2019

Title: CCBias: An Event Detection Optimization Framework

Approved: _____

We present a software, CCBias, to assist researchers in observing events of all kinds. Given characteristic information about a population of objects and observational methods, CCBias can generate synthetic observational data. CCBias can also recommend search strategies if told what observational outcomes are desirable. Lastly, CCBias can estimate the bias in real data by transforming the problem of identifying bias into a problem of estimating model parameters. We demonstrate the strengths and weaknesses of CCBias in a case study focused on planetary defense. CCBias is written in the Python programming language.

Acknowledgements

I want to thank Professor Greg Bothun for prompting me on this journey and giving me the guidance and affirmation to see it through to the end.

Many thanks also go out to Professor Eric Corwin for welcoming me into his lab and helping me find a fun, tractable project to work and fail on.

Professor Carol Paty is owed a massive debt of gratitude for her excellent questions during my defense, and her clear recommendations how to make the thesis the best it can be.

Thanks to Jonah Rose and Peter Lovett for indefatigably supporting me despite my vague nonsense

Of course, without the support of my parents, brother, Nana, Tata, or Oma, I would never have been able to make it here. To all of you: Thank you

Table of Contents

List of Accompanying Materials

1. Source Code: https://github.com/Mountebank6/CCBias

List of Figures

Introduction

Observational research can be abstracted down to observers searching for events that live in some space. These events have positions in space and properties that evolve with time. The observer interacts with the events through the observation process. In the observation process, a subspace of the space the events live in is observed and a subset of the events in this subspace are detected. After observation concludes, the observer can attempt to draw conclusions about the properties of all the events by analyzing the properties of the events they detected. When the properties of the set of observed events are not representative of the set of all events, their data is biased: either they have obtained an unrepresentative sample by poor luck, by an observing strategy that systematically led them astray, or by a combination of the two.

We present a software, CCBias, to aid observers of all types with the observing process. CCBias aims to help observers in three ways. First, it simulates the observing process with the hope that simulation can help observers understand what factors are important in their observations. The output of the simulation is a set of synthetic data: an attempt to mimic reality, but ultimately fake. Second, it recommends observing strategies. Every kind of event is unique, and every observer wants something different out of their observations. CCBias takes the individual observer's desires into account and recommends a search strategy that is right for them. Third, CCBias estimates the bias present in an observer's already gathered data. Because bias is defined as a discrepancy between the properties of observed events and the properties of all events, CCBias estimates bias by using the properties of the observed events to estimate the properties of all events—any difference between these two sets of properties is the bias.

Overall, CCBias has generality as a critical design goal—it must be written such that if a real-life situation can be thought of as a set of observers observing a set of events, then that situation should be able to be modeled in CCBias. However, CCBias is primarily meant to be used in astronomical contexts.

Literature Review

Broadly, CCBias deals with decision making when presented with a dataset that has had certain data points systematically removed or capped—censored data. This is a particularly salient problem in Astronomy, where objects can be censored because they are too dim or because they emit radiation in the wrong ranges of frequencies and so cannot be detected by certain observers. This has led to the development of many statistical tools (many in the '70s-'90s) designed to compensate for the censorship, estimating the true properties of the class of objects being observed (Huang, Wellner 1997). CCBias does not (yet) make explicit use of these methods because it is designed to be useful in arbitrary event detection scenarios. This means that the data CCBias generates will often incompatible with the assumptions of one or more of the methods described. However, it is important that we discuss the historical approaches to censored data problems to place CCBias within a larger context.

Censored data analyses have been used in many different fields (particularly medicine), consistent with CCBias's design goal of being widely applicable to scenarios outside of astronomy. Heagerty *et al* use a novel censored data method to predict a person's survival probability after breast cancer is detected (Heagerty *et al*, 2005). In the data sample they use, some of the subjects are alive, so the time between their detection of breast cancer and their death is unknown. This means that the longer a

patient survives, the more likely it is that their data is removed from the data. Without a censored data approach, this effect biases the data set towards representing sooner deaths. Working with a similarly biased dataset, Breslow (1974) compares several popular censored data regression techniques in the context of childhood leukemia treatment effectiveness at reducing all-cause mortality. In an industrial application, Nelson and Hahn (1972) apply a similar censored survival data method to estimate time-to-failure of a set of alloys. These observers all develop methods that seek to estimate the same thing: how long until an event occurs. This is one of the motivating questions behind CCBias which seeks to help researchers understand the relationships between time spent observing and populations of events detected. CCBias seeks to help researchers answer this question from a simulation approach rather than a statistical one, giving researchers multiple options in answering these questions.

Censored data approaches have found much success in Astronomy. Akritas and Siebert describe a method they designed to estimate statistical significance in the presence of censored data (Akritas, Siebert, 1996). They use this method to find that a strong correlation between the x-ray luminosity and the total radio luminosity of 88 radio-emitting galaxies is likely an artifact of two selection effects that combine to cut out galaxies that do not fit this correlation (Akritas, Siebert, 1996). Akritas, Murphy, and LaValley also use a similar technique to confirm more confidently a set of conclusions about the relationship between radio emission and star formation rate (Akritas, Murphy, LaValley, 1995). In both of these papers, the “ASURV” package for survival analysis, written by LaValley, Isobe, and Feigelson, is used to handle some of the statistical legwork of their analysis (LaValley, Isobe, and Feigelson 1992). CCBias

aims to fill a similar role to ASURV did in these analyses: providing a supporting role in the research process rather than taking center stage. CCBias takes a very different approach to this than ASURV, however, generating insight by allowing a freer relationship between the researcher and their target system rather than statistical formulae.

Terminology

This thesis makes extensive use of the language associated with Object Oriented Programming (OOP), particularly “classes”, “objects”, “functions” or “methods”, and “attributes”. A class is a general description of a type of thing, and an object is a particular example of that type of thing. For example, Pickup trucks and sedans are both in the class “car”, so they share many fundamental properties, but they are still very different objects. In OOP when we use the information about a class to create an object, we say that the object is instantiated. A function converts zero or more inputs into some output. A method is a function that every member of a particular class has. For example, all cars have the `drive` method, but all phone books do not. An attribute is a piece of information held by an object. For example, the `numberOfCupHolders` attribute of a pickup truck might be the number 4, and it might be the number 24 for a sedan.

In this document, all names of `classes, objects, functions, methods`, and attributes will be represented in a different font than the main body of the text. Additionally, unless they appear at the start of a sentence or in a chapter title, classes and objects will have the first letter of every word in their name capitalized. For example, the `TransientEvent` class. All methods, functions, and attributes will have

the first letter of their name lower case and the first letter of every other word in their name capitalized. For example, the `extraObstruction` function.

Finally, CCBias models the observing process as a long series of discrete operations, each associated with a small increase in time. A “time-step” refers to a complete simulation of that small increase in time: all events evolve their properties, all observers observe a little bit, and maybe some new events appear. Just like how a film represents fluid motion by displaying a series of static images quickly, CCBias simulations the observing process by moving forward in time one step by step.

Part I: A Generalized Survey

Our goal is to generalize the event-observing process. We say that when an observer systematically searches for events, performing observations of any kind, they are conducting a survey. A survey describes the entire process of searching for a particular set of events. It is a class that contains all the information necessary to simulate the entire observation process and output synthetic data of what synthetic events were detected. Hereafter, "survey" refers to a simulation, or to a class that performs a simulation, unless explicitly stated otherwise. Therefore, our survey object will require a complete description of: what kinds or classes of events will be simulated, how those events are generated in space and time, how observations are being performed, and confounding variables that impede observation. To construct a survey class that holds all this information and simulates the observing process, we will define several new classes that will help the survey perform its task. In the following sections, we will use the phrases like "X class must be given information that…" to represent supplying that class with a user-defined function or variable upon instantiation.

The `TransientEvent` Class

The `TransientEvent` class is the workhorse of the simulation. The motivation behind defining this class is that a single `TransientEvent` object represents a single event, and it will contain methods that evolve its properties as the simulation continues. These events must have distinct positions in space, be able to evolve their properties with time, and perhaps eventually die. To this end, when instantiating a

`TransientEvent`, all the information necessary to calculate the event's properties at an arbitrary time must be given by the user (or by a `TransientGenerator`).

The `TransientGenerator` Class

To simulate the survey process, it is necessary to construct class that controls how events are generated and what properties they have. To this end, we define the `TransientGenerator` class, which must be given by the user all the information about the topology of the search space the events live in, how new events are generated at what times, and the intrinsic properties of those events. This allows the `TransientGenerator` class to populate our simulation with new events at every time-step.

The `ObservingProfile` class

Once the `TransientEvent`s are created, we need a way to simulate the observation process. The observation process stands between the individual researcher and an accurate description of the events being studied, controlling precisely what events are detected. Therefore, it is reasonable to assume that most of the bias researchers experience in their data is mediated by the observation process.

Functionally, the `ObservingProfile` class contains all the functions and parameters necessary to answer the questions: "If there was an event at an arbitrary location in space, would we have even a chance of detecting it?", and "If we do have a chance of detecting an event, what information do we need to confirm that detection?". The first question essentially is asking "Where is the observer looking?": events that occur outside the observer's field of view, or events that are obstructed, can never be

detected. Therefore, the user must give the `ObservingProfile` class information that characterizes the observer's field of view and what might obstruct it.

Even if the observer is looking at an event, however, that is no guarantee that the event will be detected. An event might be too dim, drowned out by the noise, or

The `TransientSurvey` class

The classes just described all play critical roles in the simulation of the survey process--generating synthetic events and simulating observational selection. To complete this, we create the `TransientSurvey` class, which ties together the `ObservingProfile` and `TransientGenerator`, handles storage of the generated `TransientEvent` objects, and has the helper methods necessary for the next two parts: optimizing survey strategies, and estimating bias in real data. When instantiated, a `TransientSurvey` object takes both an `ObservingProfile` and a `TransientGenerator` as arguments, because these two objects together uniquely describe a survey. It then handles operations that are survey-level. The most important attribute of the `TransientSurvey` class is the `events` attribute, which is a list that contains all the events that have been generated. The most important method of the `TransientSurvey` class is the `advance` method. This method updates the current time in the simulation, iterates all the events forward one time-step, uses the information from `ObservingProfile` to see which of the events are candidate detections, then uses `TransientGenerator` to create new events.

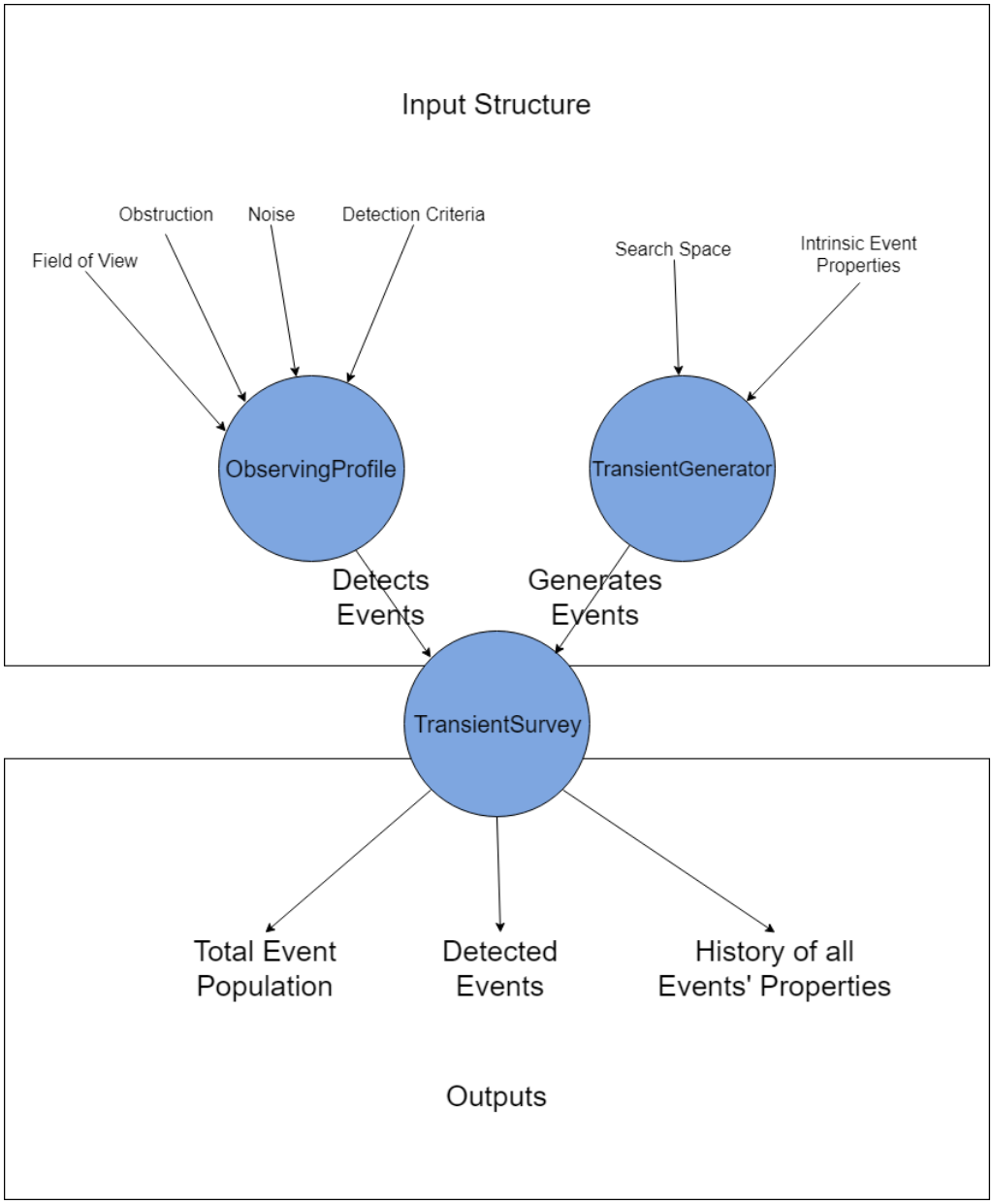

Figure 1: Outline of the Simulation Object Hierarchy

This figure outlines the components of a survey. The TransientGenerator tells the survey how to create events and what properties those events have. The ObservingProfile tells the survey how effective the observer is at detecting those events. Then the TransientSurvey uses all this information to run the simulation, generate events, and output which of them were detected by the observer

Part II: Optimizing Survey Strategies

When observers are searching for events of any type, they typically have many degrees of freedom in the details of how that search is performed. An astronomer must choose how long to collect data from a single region of sky before pointing their telescope somewhere else. They must also choose how to filter the incoming light, the time between multiple samplings of the same location, and the field of view of the telescope used. These parameters can influence the statistical properties of the data ultimately collected. Since performing observations typically costs both time and money, these parameters ought to be chosen to maximize the expected quality of the data produced. CCBias has a built-in feature to help the user choose the best set of observing parameters. Specifically, the observing parameters are the extra arguments to the user-given functions, `viewingField`, `extraObstruction`, `surveyNoiseFunction`, and `holisticDetection`. For example, a user might provide a `viewingField` that, in addition to what `ObservingProfile` requires, also must be given a `driftSpeed` parameter that controls how quickly the simulated telescope pans across the sky. Then `driftSpeed` would be an observing parameter.

Evaluating Performance

To find the best set of observing parameters, CCBias needs a way to determine which of two parameter sets is better. What 'better' means is up to the user, but typically includes motivations like maximizing number of detections and minimizing cost. To give the user the full freedom to optimize given their particular situation, CCBias requires that the user upload a `scoringFunction`. The `scoringFunction`

takes in only one argument, a `TransientSurvey` object. The `scoringFunction` looks at what events in that `TransientSurvey` object were detected, which events were undetected, and returns a real number corresponding to a score for the survey—higher being better. This construction means that two different sets of observing parameters can be directly compared. To compare two sets, create an `ObservingProfile` object for each set of observing parameters, put each `ObservingProfile` into its own `TransientSurvey` (with the same `TransientGenerator` for both), advance both `TransientSurvey` objects for the same number of time steps, then apply `scoringFunction` to both and compare the scores that `scoringFunction` returns. Whichever set of observing parameters has the higher score is, by definition, better.

The Genetic Algorithm

CCBias implements a genetic algorithm for the optimization of observing parameters. Qualitatively, this means that CCBias generates a population of random observing parameters and treats them as competing organisms, with resources assigned to each set of parameters according to the score that the given `scoringFunction` assigns to that set of parameters. The sets of parameters with higher scores have higher chances to produce offspring. Each pair of "mating" parameter sets produces offspring that are a mixture of both parents, and have an additional chance to mutate, moving the value of one or more parameters to one present in neither parent.

The genetic algorithm was chosen over other possible algorithms because it is simple to implement, and the structure of `TransientSurvey` is amenable to high

populations. It is typically computationally inexpensive to load a `TransientSurvey` with one set of observing parameters, score it, then swap out its `ObservingProfile` with another and score that one, so it is possible to use very high populations with little cost. An upside of the genetic algorithm is that, unlike many other optimization algorithms, it does not require that the `scoringFunction` be differentiable or even continuous. This allows the user far more freedom in selecting a `scoringFunction` that suits their needs.

The implementation of the algorithm is handled by the `TransientGenetic` class. To be instantiated, a `TransientGenetic` object must be given: a complete `TransientSurvey` (containing the `ObservingProfile` whose parameters will be optimized and a `TransientGenerator` to create the events), a `scoringFunction` to rank the parameter sets that will be optimized, the number of time steps to run the `TransientSurvey` objects before they are scored, the number of parameter sets to generate (`popSize`), a mutation rate probability controlling likelihood of children mutation, a crossover rate probability controlling if children are highly similar to one parent or a fine mix of both, and the number of iterations to run the genetic algorithm.

Gene Structure

To function, the genetic algorithm requires the observing parameters to be represented like DNA—finite lists of values, and each individual value must be associated with a lower and upper bound on what values are acceptable. This prevents the simulation from wasting time probing non-physical parameter values. To formalize this, when `viewingField`, `extraObstruction`, `surveyNoiseFunction`, and

`holisticDetection` are given to an `ObservingProfile` object, for each of their observing parameters a characteristic gene must also be given to that `ObservingProfile`. A characteristic gene is a list of the form `[low, high]` where low and high are the lower and upper bounds, respectively, on that observing parameter. These characteristic genes are combined to form the characteristic genome, which lists the bounds on every observing parameter, uniquely characterizing the bounds on the parameter space that the genetic algorithm will explore. It is assumed that any observing parameter is orthogonal to the bounds on any other—changing the value of observing parameter A does not change the characteristic gene for observing parameter B. Hereafter, a particular set of observing parameters will be called a genome.

Initialization and Reproduction

When `TransientGenetic` is instantiated, it extracts the full characteristic genome from the `ObservingProfile` object contained within the `TransientSurvey` it is given. `TransientGenetic` uses the characteristic genome to generate a number of genomes equal to `popSize`. Then, it plugs each genome into the supplied `TransientSurvey` object and uses the `scoringFunction` to score the genome.

After each genome in the initial population is scored, the scores are summed, and this sum is stored as scoreSum. In the reproduction process, each genome has a probability of score/scoreSum to be selected to be a parent. By this method, high scoring genomes contribute more heavily to the gene pool, but low scoring genomes are not categorically excluded.

Each pair of parents (hereafter, mother and father) selected to reproduce produces two children. First, the children are initialized such that one is equal to the mother and the other is equal to the father. Then a random number between 0 and 1 is generated for each gene in the characteristic genome. If the number associated with a particular gene is less than the crossover rate, then that gene is marked as a crossover point. At each crossover point, the children switch which parent they draw genes from. For example, if the genomes were 10 genes long, and gene #3 was the only crossover point, one child genome would be the first 3 genes from the mother and the last 7 genes from the father. The other child would be the reverse—the first 3 genes from the father and the last 7 genes from the mother. After crossover is performed, both children undergo mutation. Each gene in each child has a probability equal to the mutation rate to mutate. When a gene mutates, its value is set according to the uniform distribution with bounds determined by the associated characteristic gene.

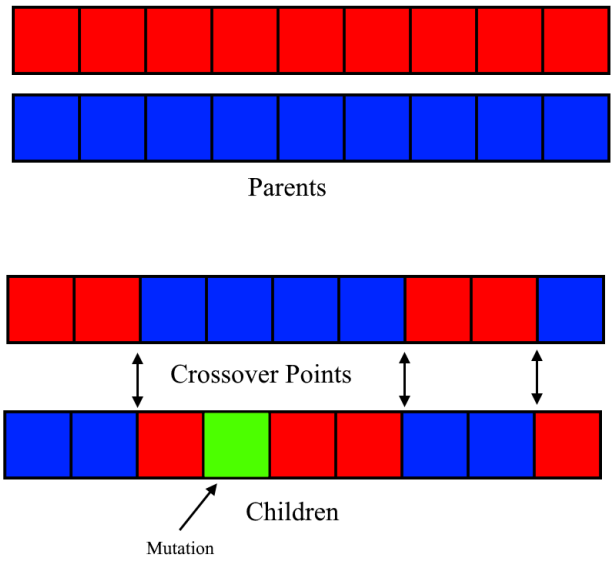

Figure 2: The Reproductive System

This process repeats until a number of children equal to half of `popSize` are generated. An individual genome can be selected to be a parent multiple times. After all the children are generated, the bottom scoring half of the population is deleted and replaced by the children. The top-scoring half lives on for the next generation. This algorithm of scoring, reproduction, and culling repeats a number of times equal to `totalGenerations`.

A significant downside to the algorithm is that, because of how crossover is performed, the order of the genes in the genome matters. For example, for low crossover rates, any pair of adjacent genes is likely to come from the same parent, but the first and last genes have a different probability of coming from the same parent. Therefore, if the observing parameters that those genes represent interact strongly with each other (as measured by the scoring function), then the genetic algorithm would converge faster at low crossover rates if those parameters are closer in the genome than if they are farther away.

Part III: Bias Estimation

When observational researchers collect data, they reasonably expect that their data is biased—the statistical properties of the events they observed are not precisely representative of that class of events as a whole. This bias can come from many sources: insufficient technology and poor observing strategies are a pair of simple examples. Since observers are aware of this, they try to estimate what events their scheme is biased against detecting. However, when the observing strategy is complex, this can be difficult to estimate. To help with this, CCBias can estimate the bias in real data.

Assumptions and Structure

CCBias defines 'bias' as a discrepancy between the statistical properties of the set of all events that have been observed ($\boldsymbol{\gamma}$), and the properties of the set of all events that exist in reality ($\boldsymbol{\Gamma}$). If $\boldsymbol{\Gamma}$ is known, and $\boldsymbol{\gamma}$ is a sufficiently large subset of $\boldsymbol{\Gamma}$, then any statistical discrepancy between $\boldsymbol{\Gamma}$ and $\boldsymbol{\gamma}$ can only be explained by a systematic failure to detect a representative sample of the events being investigated.

To estimate bias, CCBias takes as input data from a set of observed events (i.e. a $\boldsymbol{\gamma}$) and a fixed `ObservingProfile`, then uses that to estimate the model parameters of a given `generatorFunction`. Once these model parameters are estimated, they can be used to generate a $\boldsymbol{\Gamma}$. Then, by definition, any discrepancy between the statistical properties of $\boldsymbol{\gamma}$ and $\boldsymbol{\Gamma}$ is the bias present in the system. This means that the problem of estimating bias has been converted into a problem of guessing model parameters. Critical to this process is that there is a causal connection between the model parameters given to the `generatorFunction` and the data produced after the generated events are

passed through the `ObservingProfile`. If an observer wants to guess the abundance of an event that is impossible to detect, CCBias can only provide useful information if the abundance of that event is coupled to properties of events that are detectable. Therefore, when using CCBias to investigate the presence of events that are extremely difficult to detect, the `generatorFunction` used must describe very precisely the coupling between events that are difficult and easy to observe.

Simultaneous Perturbation Stochastic Approximation

To estimate bias, CCBias requires that the user define a `lossFunction`. This is a function that takes as input the observed set of fixed data the user wants to estimate the bias of, a fixed `ObservingProfile` object that describes how that data was collected, and a set of model parameters to test. The `lossFunction` uses those model parameters to generate a family of events, applies the given `ObservingProfile` to them to calculate a set of measurements, and compares those measurements to the given observed data. The `lossFunction` returns a (strictly positive) real number that represents how different the generated and observed data are. Therefore, if a set of model parameters makes the `lossFunction` return a value very close to zero, then those model parameters are consistent with the data observed.

CCBias estimates the `generatorFunction`'s model parameters by implementing the Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm (Bhatnager, 2013). This algorithm is a minimization algorithm that is designed to be effective when the function being minimized has many unknown

parameters. Since CCBias has generality as a design goal, the ability to handle many-parameter models is attractive.

The first step of the SPSA process is to define the search space of `generatorFunction` parameters that `lossFunction` will be minimized over. When the `TransientGenerator` object containing `generatorFunction` is instantiated, the user is required to give a characteristic genome for all of the extra parameters that the `generatorFunction` requires. Although the SPSA algorithm is very different from the genetic algorithm that optimizes survey strategies, the concept of the characteristic genome defining the search space is the same, so the language has been carried over.

The SPSA algorithm is fundamentally a gradient-descent algorithm: at each step of the algorithm, it estimates the gradient of the `lossFunction` around its current best guess of the minimum and updates its guess along the direction that lowers the `lossFunction` value (Bhatnager, 2013). Let $\vec{\mathrm{r}}_n$ denote the algorithm's current best guess. Then at the next iteration of the algorithm $\vec{\mathrm{r}}_{n+1} = \vec{\mathrm{r}}_n - a_n \hat{g}_n(\vec{\mathrm{r}}_n)$, where $\hat{g}_n(\vec{\mathrm{r}}_n)$ is an estimate of the gradient of the `lossFunction` at $\vec{\mathrm{r}}_n$, and a_n is a sequence of positive real numbers that converges to zero and satisfies $\sum a_n = \infty$. CCBias modifies this algorithm by forcing the best-guess vectors $\vec{r}_n$ to lie inside of the bounds defined by the characteristic genome. The i[th] component of the gradient estimator, $\left(\hat{g}_n(\vec{\mathrm{r}}_n)\right)_i$ is defined as:

$$\left(\hat{g}_n(\vec{\mathrm{r}}_n)\right)_i = \frac{L\left(\vec{\mathrm{r}}_n + c_n\overrightarrow{\Delta_n}\right) - L\left(\vec{\mathrm{r}}_n - c_n\overrightarrow{\Delta_n}\right)}{2c_n\left(\overrightarrow{\Delta_n}\right)_i}$$

where $L(\vec{r})$ represents the value of `lossFunction` evaluated at parameter set $\vec{r}$, c_n is $\frac{1}{n^{\gamma}}$ with $\gamma \in [\frac{1}{6}, \frac{1}{2}]$ and additionally satisfying, $\sum \left(\frac{a_n}{c_n}\right)^2 < \infty$, and $\overrightarrow{\Delta_n}$ is a random vector whose components are $\pm\frac{1}{2}$, with equal probability for each. Given these constraints, if $L(\vec{r})$ is thrice continuously differentiable, with bounded third derivatives, then the sequence of best guesses, $(\vec{r}_n)$, is guaranteed to converge to one of `lossFunction`'s global minima (Bhatnager, 2013). The global minima are interpreted as the sets of model parameters that are most consistent with the given input data and the given `ObservingProfile`. If estimates on precision are desired, `lossFunction` can be probed along each model parameter to estimate the distribution of `lossFunction` values in along that parameter and return a standard deviation of that distribution.

A fundamental limitation to this `lossFunction` minimization approach is computation time, independent of exactly what minimization algorithm is chosen. On consumer hardware, evaluating `lossFunction` once may take over 10 minutes, and hundreds of algorithm iterations may be needed before a minimum is settled upon. This provides a severe limitation for casual use. This limitation is not shared by the survey strategy optimization's genetic algorithm because in order to test a set of model parameters with `lossFunction`, an entirely new family of events must be generated and observed with the given `ObservingProfile`. However, scoring a genome of observing parameters only requires a re-run of the `ObservingProfile` process over a pre-generated family of events, which allows many more sets of observing parameters to be tested than model parameters.

Case Study: Planetary Defense

CCBias's versatility, benefits, and shortcomings can be demonstrated with a simple case study. A clear example of observing objects with clear stakes is planetary defense—searching for Near Earth Objects (NEOs) whose orbits interest or closely approach Earth's. To constrain the space of possible searches, this case study will mimic the Pan-STARRS survey (Tonry, *et al,* 2012). Pan-STARRS is a sky survey that has been collecting data since 2010. Pan-STARRS employs the largest camera in use at 1.4 gigapixels and scans the sky every night, covering the whole sky once every four days, given ideal conditions (Tonry, *et al,* 2012). In 2005, the US Congress issued a mandate that 90% of all NEOs larger than 140 meters be catalogued by 2020—a follow-up to a 1990s mandate that 90% of all NEOs greater than 1000 meters be catalogued.

Model Design

To improve simulation speed, this case study makes some simplifying assumptions. This model assumes Pan-STARRS is located at the center of the solar system, rather than on Earth. The primary effect of this assumption is that the model can detect NEOs located between the Earth and the Sun, so to account for this, those detections are removed by the `holisticDetection` function used by the model. Additionally, the model assumes Pan-STARRS is located in the ecliptic plane. This assumption gives the model symmetry in its ability to detect NEOs above and below the ecliptic. Finally, the model assumes constant, ideal conditions: no noise in the detector (although there is a minimum brightness it can detect), ideal observing conditions year-round, and no time lost to reorienting the telescope between exposures.

Generating the Events

In this model, the NEOs of interest are assumed to follow perfect Keplerian orbits, never being perturbed by the Moon or other planets in the solar system. Keplerian orbits require 6 parameters to be uniquely identified. The most important ones for this model are the semimajor axis, the eccentricity, the inclination, and the true anomaly. The semimajor axis, a, is the average of the object's closest and furthest approach to the Sun. The eccentricity, e, controls the ratio of the closest and furthest approach. The inclination, i, controls the tilt of the orbit relative to the ecliptic as well as whether the orbit runs clockwise or counterclockwise. The true anomaly, ν, is the only time-evolving parameter, describing where the object is in its orbit at $t = 0$ and at all following times. The other two parameters control how the orbit and its tilt are oriented in the ecliptic plane.

Since the focus of this case study is planetary defense, the model only generates NEOs that are flagged as 'potentially dangerous', which is defined as when their closest and furthest approaches to the Sun are on opposite sides of Earth's orbit. Because NEOs can have inclined orbits, being flagged as potentially dangerous does not imply that their orbits intersect Earth's.

Viewing Field

The model's `viewingField` function mimics Pan-STARRS's observing strategy: taking four exposures over an hour, then reorienting and exposing again. The actual Pan-STARRS setup involves four telescopes, in total surveying about 6000 square degrees of sky per night (Tonry, *et al,* 2012). Assuming a 12-hour night, this corresponds to an effective field of view with 7° diameter, so in this model only one

telescope is simulated and it is given this effective field of view.[1] The model assumes that only 2π steradians of sky are able to be observed at any time, simulating the fact that half of the sky faces towards the sun and Pan-STARRS cannot take data during the day. What 2π steradians of sky are visible depends on the time of year. Additionally, to simulate a similar restriction in the actual Pan-STARRS, the model only detects objects that are within 4π/10 radians of the ecliptic. In total, over the course of a year, the model can observe 3.2π steradians of sky, which is slightly better than the 3π steradians that Pan-STARRS achieves.

Over the course of a night, the model has access to 1.6π steradians of sky. It divides this into a grid of 7° × 7° squares with rows parallel to the ecliptic. The model then systematically moves across the rows, taking four exposures over an hour in each square. Since the Earth is constantly moving around the sun, the available 1.6π steradians of sky evolves over the course of a night (the effect of the Earth's daily rotation is subtracted off), whenever the model finishes sampling a full row, it updates what portions of the sky are visible, and starts over in a new row, starting with the first square of sky that will be removed from view by the revolution of the Earth. If any NEO is in the field of view of the telescope during an exposure, it is marked as a candidate detection regardless of brightness. Whether the NEOs are marked as confirmed detections is handled by `holisticDetection`.

[1] This "effective FOV" method is also required to cut down on computation time and memory usage. Fully simulating the four telescopes, each with smaller fields of view and lower exposure times would increase time-to-completion and memory usage by approximately a factor of 10.

Holistic Detection

The candidate detections are marked as confirmed detections if they meet four requirements: they are imaged at distances greater than Earth's orbit, they are imaged at least four times in a single night, they move at least one pixel over those four images, and they exceed a minimum brightness requirement. The second requirement is met by virtually all candidate detections since the effective field of view is so large. Similarly, the 1.4 gigapixel camera allows for the movement of distant NEOs to be resolved over a full hour. The limiting requirement is that the NEOs must be brighter than an apparent magnitude of 24 in the infrared band. This corresponds to approximately 10 or more infrared photons per second reaching the detector. Even objects as large as Fenrir, a moon of Saturn, 4000m wide at 10AU away do not meet this requirement. To keep the model tractable, no `extraObstruction` or `surveyNoiseFunction` is incorporated into this case study.

Model Results

Since this model is a significant simplification of the actual Pan-STARRS survey, it cannot definitively conclude Pan-STARRS will systematically fail to detect certain classes of events that it ought to be able to. Additionally, the distributions that the model uses for NEO properties are not intended to be representative of any actual population of dangerous NEOs, and wide ranges of parameter values were accepted. The key questions being considered are: given the simulation, what biases can Pan-STARRS be expected to have in the kinds of potentially dangerous objects it detects? Can Pan-STARRS detect objects like the Chicxulub impactor which wiped out the dinosaurs? Under what conditions?

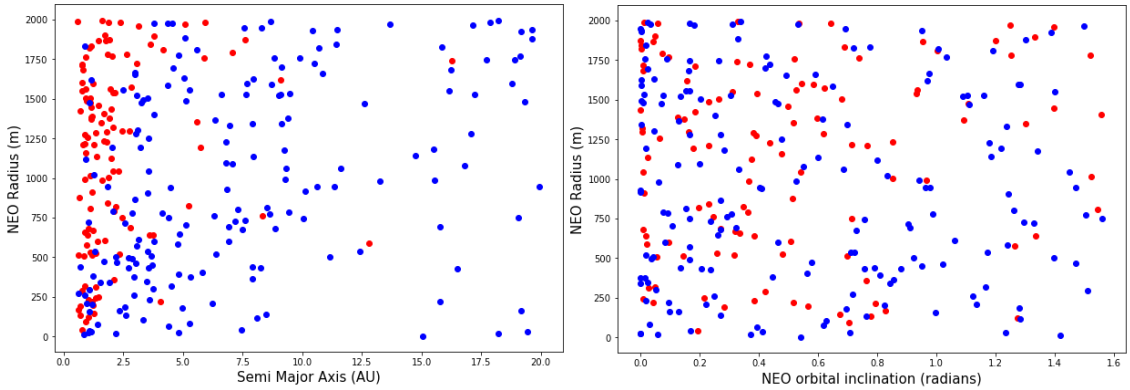

Figure 3: Detection of 300 NEOs with respect to Semi Major Axis and Inclination

In this figure, the properties of detected (red) and undetected (blue) potentially dangerous NEOs are compared (300 events in total). The high mass of low semi major axis and high radius detected NEOs suggests that the claim that over 90% of all >1000m NEOs have been detected is plausible. The number of high mass, high inclination undetected events is worrisome—Pan-STARRS has difficulty detecting objects moving toward Earth orthogonal to the ecliptic, no matter how massive.

Overall, the Pan-STARRS model is competent at detecting midsize (140m-500m) NEOs, detecting around 25% of them. Those that are undetected typically have higher semi major axes than those detected, indicating that the limiting factor is brightness. Indeed, a 250-meter NEO must be less than 2 AU away from the sun before it crosses the brightness threshold in the model. While NEOs with orbits in the ecliptic plane (i.e. low inclination), most likely get cleaned up by interactions with planets, this is not guaranteed for NEOs with moderate inclination. Encouragingly, Pan-STARRS's camera has enough pixels such that none of the events in the 140m-500m range were rejected because they moved less than one pixel on an hour. A potential concern is that surveys like Pan-STARRS will systematically fail to detect objects on collision courses with the Earth. This model suggests that this problem might not be as large as it seems on first examination for objects within 20AU. However, it might simply be an issue of bias.

Objects in the 140m-500m range might be small enough such that the brightness threshold makes them undetectable until they are within distances that require high angular velocities. We can probe this possibility by exploring a very different regime of object, Chicxulub impactors.

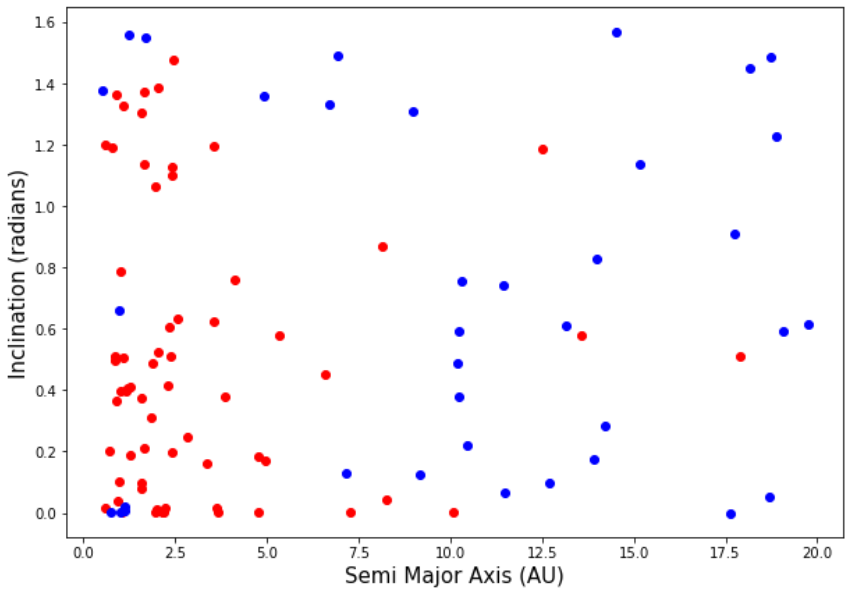

Figure 4: Detection of 100 Chicxulub impactors

100 Chicxulub impactors were inserted into the model and it was ran for one year. The impactors with orbits <3 AU are sometimes undetected because they hide near the sun, and those with higher semimajor axes are often undetected because they are not within 20 AU (the brightness threshold) or not in the space surveyed by the model

Confirming this bias hypothesis, if an asteroid like the Chicxulub impactor (~50km across) is inserted into the model on a random Keplerian orbit it is not always detected even if it is within the 3.2π steradians that the model samples and within 20AU (the brightness threshold for a 50km asteroid). 7/100 of the impactors satisfied these requirements and were undetected because they did not have angular velocities high enough to exceed one pixel per hour. One of the impactors was as close as 10AU, the orbit of Jupiter. Since objects of this scale have the capacity to drastically damage

human civilization, this model suggests that extra care should be taken to confirm whether Pan-STARRS could detect these objects automatically. Of course, because the impactors are bright enough to be visible once they are within 20AU, they will still be present in the data, but the automatic flagging done on the four one-hour exposures does not detect them.

Discussion

NEO Model Limitations

The model assumes that the observing site is at the center of the solar system. Although this is adjusted for in the observed brightness calculation, it does impact the movement threshold for the one-pixel requirement. Since the Earth moves around the Sun, then the observed background of stars would be moving at a different speed than a given NEO, even if the NEO is moving directly towards the Sun or the Earth. Since Pan-STARRS has enough pixel density to resolve the movement of the background of stars over the course of an hour, the model systematically overestimates how many objects will be stationary.

It is worth repeating the limitation that CCBias is inefficient enough with memory and computation that running the observing parameter optimizer and the bias estimator is infeasible for sufficiently complex model. That was the case for this Pan-STARRS model. Even ignoring memory limitations, which are the main bottleneck, it would have taken about 200 hours to run through enough iterations of the genetic algorithm to sufficiently constrain a search strategy. The same is true for bias estimation: the original intent was to run the bias estimation algorithm on the actual data from the Minor Planet Center and approximate the actual distributions of NEOs. This also would have taken over 200 hours even with infinite memory.

General Discussion

Ultimately, the most useful part of CCBias is the content described in Part I, the Generalized Survey. This is because it takes approximately one hundredth to one

thousandth the time to generate survey data than to use observing strategy optimizer or the bias extractor. For any moderately complex survey, where simulating it once takes five minutes, it would take 8 hours to optimize observing strategies. For a software whose explicit purpose is casual, first-look use, this is an unacceptable tradeoff. Additionally, memory limitations reduce the quality of the optimizer's results even when ran. A consumer machine typically has only 16 GB of memory, and in the case study, simulating 300 events for one year (Pan-STARRS's science mission has already lasted five years) used about 10GB of memory. To get higher quality statistics, at least 1,000 NEOs should have been used. Because the long wait times on the observing strategy optimizer and bias extractor are because running one `TransientSurvey` simulation takes many minutes, to resolve these issues and improve efficiency would require a significant redesign of the basic structure of CCBias.

Fortunately, as the case study shows, it is possible to learn a lot about potential bases in a system without using the automatic bias extractor. CCBias's strength is how simple it is to create new surveys with entirely novel observing strategies with the knowledge that the ugly managing of what does and does not count as a confirmed detection is handled by the software. Additionally, it is quite simple, when running CCBias in something like a Jupyter notebook, to extract information about events that is not returned by the `measurementFunction`, which allows for the easy creation of plots and investigation of event properties.

Bibliography

Akritas, Michael G., and J. Siebert. "A test for partial correlation with censored astronomical data." *Monthly Notices of the Royal Astronomical Society* 278.4 (1996): 919-924.

Akritas, Michael G., Susan A. Murphy, and Michael P. Lavalley. "The Theil-Sen estimator with doubly censored data and applications to astronomy." *Journal of the American Statistical Association* 90.429 (1995): 170-177.

Breslow, Norman. "Covariance analysis of censored survival data." *Biometrics* (1974): 89-99.

Bhatnagar, S., Prasad, H. L., and Prashanth, L. A. (2013), Stochastic Recursive Algorithms for Optimization: Simultaneous Perturbation Methods, Springer

Heagerty, Patrick J., and Yingye Zheng. "Survival model predictive accuracy and ROC curves." *Biometrics* 61.1 (2005): 92-105.

Huang, Jian, and Jon A. Wellner. "Interval censored survival data: a review of recent progress." *Proceedings of the First Seattle Symposium in Biostatistics*. Springer, New York, NY, 1997.

LaValley, M., T. Isobe, and Eric Feigelson. "ASURV: astronomy survival analysis package." *Astronomical data analysis software and systems I*. Vol. 25. 1992.

Nelson, Wayne, and Gerald J. Hahn. "Linear Estimation of a Regression Relationship from Censored Data Part I—Simple Methods and Their Application." *Technometrics* 14.2 (1972): 247-269.

Tonry, J. L., et al. "The Pan-STARRS1 photometric system." *The Astrophysical Journal* 750.2 (2012): 99.