A Program for ϵ-Machine Reconstruction in (1+1) and (2+1) Dimensions

Jacob Usinowicz
jacob@santafe.edu

August 29, 2002

1 Overview

This program includes improved code for the ϵ-Machine reconstruction process in (1+1) dimensional systems and also extends the algorithm to (2+1) dimensional systems. The code for (1+1) dimensional processes is taken from pre-existing code, which in turn was based on the thesis work of Cosma Shalizi. The code for (2+1) dimensions was written from scratch over the course of approximately 2.5 months while in residence at the Santa Fe Institute and [1] sponsored by the NSF REU program.

This code is written in c++, an object oriented language that allows for the placement of code into seperate files (classes) that get called by the main program. Thus there are a number of classes that make up this program. Each of these will be noted below, and its primary functions discussed in varying levels of detail.

2 General Description and Usage

`reconstruct` *[-f]datafile [-w]width [-t]timesteps [-m]samples*

With the following options:

```
[-t]<timesteps> for time steps. Default=0
[-w]<width> for the width of the lattice. Default=10
[-h]<height> for height. Default = width.
[-p]<position> for the line of the file to begin reading from. Default=0
[-r]<radius> spatial radius. Default=3
[-c]<past depth> past depth of light cone. Default=2
[-u]<future depth> future depth of light cone. Default=2
[-d]<delta> delta value. Default=0.05
```

[1] Santa Fe Institute, 1399 Hyde Park Rd., Santa Fe, NM 87501. www.santafe.edu

```
[-n]<samples> samples per CA field. Default=2500
[-m]<fields> number of CA fields sampled. Default=5000
[-k]<dimensions> number of spatial dimensions. Default=2
[-s]<seed> random seed. Default=8
[-b] sets binary matrix mode. Default = off
[-v] sets Von Neumann neighborhhod. Default=off
[-a]<smallest> sets comparison number for chi-sqaures. Default=0.05.
```

2.1 The General Description

This program reads in data from a datafile which must be arranged in one of several acceptable formats. The data is sampled and used to construct causal states. The transition probabilities between states are labeled and determinized. Several information theoretic quantities are also calculated. The end output is: 1) A datafile listing effective states and their grouping after splitting 2) A datafile listing the causal states 3) A datafile that contains different information theoretic quantities in a form plottable by GNUplot 4) A datafile that lists the transitions and their probabilities in a form that is readable to the state graph generating program GraphViz.

2.2 Top Level

To begin with, the top level of the program is main.cc. It contains the routines for command line parsing, and otherwise is primarily responsible for sorting out the form of data that the user is attempting to pass and calling the correct subroutines.

The next level of calling is done by *main_call.cc*. This is where the data sampling and data manipulation routines are directly called. This class also determines the limits on how near to the edge routines may sample without exceeding boundaries of the arrays.

2.3 What is Being Sampled: Causality and Light Cones

The technique of ϵ-Machine reconstruction depends on a strict mathematical notion of causality. The assumption is that a process is composed of internal states that are linked together by directions of causation. What exactly these states are and how they lead to successors depends on the system itself. In the case of a simple time series, an internal state is simply a series of symbols (a left, or past L-block) that precedes another series (a right, or future L- block). The link between a past and a future symbol-block is the symbol that terminates the past block. In otherwords, the switch from the past state to the future state is determined by the symbol observed immediately after seeing the past state.

The algorithms here are based on the extension of causality and causal states to abstract (1+1) and (2+1) systems: cellular automata. If you are not familiar with cellular automata, I recommend reviewing some of the basics. It would at least help to know what one (typically) looks like. Due to the increase in dimensionality, the exact representation of a causal unit must be adapted to take this into consideration. Unlike a time series, a single symbol is not influenced solely by those to the left of it. The range of previous sites that determines the value of a chosen site spreads out in a cone-like manner moving backwards in time. The same thing is true for the sites that any single site is capable of influencing in the future; the range of influence spreads in a cone shape. These cones of influence are the causal states.

```
Light Cones for (1+1)

0 1 0 1 0 1 1                         X
  1 0 0 1 1                         1 0 1
    0 0 1                         1 0 0 1 1
      X                         0 1 1 0 0 1 0

Past Light Cone                Future Light Cone
For Site X                     For Site X
```

The extension to the (2+1) case produces a similar causal object, with difference that it exists in one additional spatial dimension. Conceptually speaking, it more closely resembles a pyramidal object.

2.4 Datafiles and Sampling

The program reads data in from a datafile in one of two forms: 1) coordinate and value form, 2)binary matrix form. The coordinate form follows the syntax of (x,time, value) for (1+1) data sets and (x,y,time,value) for (2+1) data sets, with no paranthesis and spaces instead of commas. The values must be listed in a single vertical column. The second form is essentially a picture of the lattice, arranged spatially on the page. For the (1+1) case, each picture contains the temporal and spatial data. For the (2+1) case, each picture contains only the spatial data, and so essentially a snapshot. The temporal data is present over a series of these snapshots. Each timestep snapshot must be separated from the next with **Note: Code is under work to be more flexible in terms of the data formats it will accept. Also, values can only be drawn from a binary alphabet. This is also being changed.**

The program converts the data from the data files to either a 2d array (for (1+1)) or a 3d array (for (2+1)). Vertices of lightcones are chosen randomly. The number of samples defaults to 2500, but can be set at the prompt with **[-n]sample size**. These sites represent vertices of light cones, which in turn represent the causal states (see discussion below). For all cases, past, future, next-past, and light cones for spatial transitions are sampled. The only thing that differs is the precise nature of the spatial transitions. In the (1+1) case, left and right past light cones are sampled, since spatial transitions only happen left to right or right to left. For the (2+1) case, North, South, East, and West neighbors are sampled for von Neumann neighborhoods. The other type of neighborhood, the eight-neighbor setup, also takes into consideration the four diagonals.

$$0\,1\,1\,1 \tag{1}$$

$$0\,1\,0\,1 \tag{2}$$

$$1\,0\,1\,0 \tag{3}$$

$$1\,0\,1\,0 \tag{4}$$

$$\tag{5}$$

An example of Binary Matrix format. Width = 4 and Time = 4 in (1+1) case. Width =4, Height =4, and Time =1 for (2+1) case.

From each vertex, past, future, and all appropriate spatial neighbor cones are recorded. Cones are stored in an integer format. Starting with the vertex, the members of the cone are read backwards in time, from left to right, and stored as a binary integer, which is then converted to a base-ten integer. Each of these base-ten integers represents a cone label. Each of these data samples is stored in its own tree data-structure. Counts of past-future combinations are also kept. The counts are used to calculate probabilities over individual, joint, and conditional distributions.

2.5 Merging and Splitting

The data is stored in such a way that for each of the neighbors sampled (past, future, left, right, etc.) the association between observed pasts and their subsequent futures is maintained. This is done by creating a list of pasts (actually a tree), where each member has its own sublist (or tree) of observed futures. These lists also keep track of the number of times each past and future were observed. A basic chi-squared test is used to compare each set of past-future combinations to every other list member. If the chi-squared test determines that it is unlikely that the two compared data sets (lists of pasts with sublists of futures) are drawn from different distributions, then they are merged into the same past. Essentially, this is a way of accounting for undersampling of the more rare light cones.

The next step is to group past-future combinations into causal states.In terms of the algorithm, this is a two step process that first creates equivalency classes, then breaks them into causal states. Essentially, states are effective states to begin with. A statistical test is systematically applied to compare similar past-future combinations and decide which states are similar enough to be kept in the same causal state. Depending on the outcome of the test, states may be kept in their current effective state, placed into another one, or split to form a new one.

After establishing which pasts belong in which causal states, the next step is to label the transition probabilities. This is done easily enough. First, pasts of the various spatial transition trees are labeled by the proper causal states.

Then the members of the future lists for each past state are labeled. Transition probabilities are determined, and the determinized structure is printed into a datafile.

2.6 Information Theory

This program returns the following quantities:
Shannon Entropy of past light cones. $H[X]$
Conditional and joint entropies. $H[X|Y], H[X,Y]$
Statistical complexity. $H[model]$
Conditional and joint entropies. $H[X|model], H[X, model]$
Mutual information.

2.7 Recommendations on General Usage

The code works just fine for any binary alphabet on a regular lattice. The dimensions of the lattice are easily determined at the command prompt by the user. You should note that for the (1+1) case, height does not need to be set, since the height is really the number of timesteps. On a (2+1) lattice, dimensionality is straightforward enough.

It is important to make sure that the data you are trying to read is in the same directory as the program. Otherwise it will look for something that is not there and return a segmentation fault or a file not found error. Also, be sure to set the dimensionality and the form of the data input correctly. Otherwise the program will generate strange errors, and likely return a segmentation fault.

The files containing the generated data will be the name of your datafile, prefixed with a series of letters that will indicate its status as either: (equiv)alency class and causal state data, (graph) data for GraphViz, (trans)ition data, or (info)rmation theory measurements. Turning the graph file into a postscript file (with $dot - Tps < filename > outputname$) will show you the reconstructed machine. But it will not be entirely evident from this printout what each state actually contains. The (equiv) and (trans) datafiles should be consulted for more detailed information. In the (equiv) file, along with the members of the different classes or states, there should also be pictures of what each state looks like.

Deciding on how many samples to take is always tricky. If you have the data for it, the program can easily take in excess of 200,000 samples. Deciding on whether or not more samples need to be taken, and also on how finely tuned the comparison percentages for the splitting test ([-d]delta) and the chi squares test ([-a]smallest) is something of a guess and check game at this point. Make sure the information theoretic quantities make sense, and try to get the program not to generate spurious or under-sampled states (usually states that show up as ovals with no arrows pointing outwards).

2.8 Quick Breakdown of Classes

There are classes in this program, each of which is comprised of a c++ (cc or cpp) file and a header file. The only exception is the main part of the program, which has no complementary header file.

main This is where command line parsing happens, and where key things such as input type and dimensionality are determined.

main_call Makes direct calls to sampling and data manipulation routines. Also establishes boundary conditions for sampling.

sample Routines for converting infile data to a useable array form, then sampling the data with functions appropriate to the dimensionality and neighborhood type called.

pftree This is where data storage, organization, and all the calculating happen.

Test Code for chi-squared test.

ran2 A random number generator.

3 Conclusions: What is it Good For?

Thus far, there are a number of things that stand as room for development. At the simplest level, the code needs to be extended to include languages other than binary, to work with the 4-site (Von Neumann) neighborhoods in (2+1) CAs, and to take data directly from stdin. More complicated tasks might include automating the process of deciding if enough states have been sampled, if [-d]delta and [-a]smallest are set well, and if deep enough light cones have been considered.

Despite these shortcomings, the code seems to be quite good (or very near to being quite good) at what it does. It is capable of returning reasonable states and accurate information theoretic measurements. It is versatile enough to do this for (1+1) and (2+1) systems. Theoretically, it is also quite capable of analyzing data from other places (other than CAs), as long as the data is in the correct form.

If nothing else, this code represents a stage of learning in the process of practical implementation and extension to causal state theory.