Small Sample Effects on Information-Theoretic Estimates

A thesis submitted in partial fulfillment of the
requirement for the degree of Bachelor of
Science with Honors in Physics from the College
of William and Mary in Virginia,

by

Andrew Davis

Accepted for _____
(Honors, High Honors or Highest Honors)

_____
Advisor: Dr. Eugene Tracy

_____
Dr. Keith Griffioen

_____
Dr. Zia-ur Rahman

Williamsburg, Virginia
May 2002

Abstract

An analysis of mutual information and transfer entropy as measures of correlation between coarse-grained symbolic time series is presented. The mutual information and transfer entropy may be used to measure the correlation between time series. Because of budget or response rate limits, or due to the nature of the system, the data to be analyzed is often restricted to short sequences of coarse-grained time-series data. Due to the limitations on the data, the measured correlation values are not expected to be the actual correlation value for the system. A method of studying the deviations from the actual values is demonstrated, as well as calculations of the mutual information for several different systems.

Acknowledgments

I would like to give special thanks to Professors Eugene Tracy and Dennis Weaver for guidance with this project. I would also like to thank George Andrews and Christopher Kulp for providing much-valued feedback.

Further, I would like to thank Professor William Cooke and Wei Yang for providing access to data from their laser experiments, as well as offering a different viewpoint on several subject matters.

Contents

List of Figures

List of Tables

This storm will be magnificent. All the electrical secrets of Heaven. And this time we're ready, eh Fritz?

– Dr. Victor Frankenstein

1 Introduction

The dramatic improvement of micron and sub-micron digital computer devices in the past twenty years has led to a renewed interest in coarse-grained symbolic time-series. As far back as 1965, Gordon Moore [1] observed that the complexity of the minimum-cost state-of-the-art integrated circuit seems to double roughly every year to eighteen months. Moore predicted that this trend would continue through 1975, but for at least one manufacturer [2], this trend has continued to the present day, resulting in extremely powerful, low-cost microprocessors. Because the current crop of microprocessors offers such a high degree of computing power in a small, lightweight, inexpensive product, they are finding their way into many different products.

Making an object "smart" by implanting it with a microscopic digital logic and control system allows designers great creative freedom. They can increase the number of features offered in a device, while at the same time reducing weight by eliminating bulky mechanical control systems. They can also improve the safety and reliability of a device by allowing it to diagnose and even correct its own failures. To that end, many intelligent devices incorporate various sensor systems that return data to the microprocessor so that the device may analyze its operation. By virtue of the fact that the logic of most processors is driven by some sort of clocking mechanism, at some point all of the sensor data must be converted into some kind of symbolic time-series.

However, the need for smart devices to make rapid decisions places limits on the amount of data the can be collected, and the time that can be afforded to process it. In many instances, a split-second decision about the state of a system is needed in order to insure its safe operation. It may not be possible, with any sensors, to collect many data points from which to make a prediction about the system's future state. For example, plasma instabilities in a fusion reactor could cause a catastrophic failure of the system unless diagnosed and corrected quickly. Since the time scale for plasma instabilities is measured in milliseconds, the diagnostic routine may not have time to collect a large number of data points for analysis prior to having to make a decision on whether or not an instability is present. This problem is exacerbated if the system's statistical character changes over time; that is, if the system is non-stationary. Should a system be non-stationary, there may be constraints on the amount of previously collected data that remains valid as the statistical character of the system evolves.

Along with time constraints, budgetary constraints also place limits on the types of sensing and processing equipment that can be used in a particular application. Because the signals used by a computer must be discrete and finite, they must be converted from their real-world, analog values. While a top of the line sensor system might be able to report a wide range of values, with many small steps in between approximating a continuous set of values, it is often not feasible to put the best sensors in a device without pricing it outside of the range of comparable models. However, a less costly sensor might only be able to report a very limited set of states. As a result the data supplied by these less costly sensors is very coarse-grained. Using detectors that report only a limited number of states may also be a design decision. Sensors that report only a few possible states are more likely to be robust

against noise effects, which may be an important consideration if the sensors are to be used in a hostile environment. System memory is also expensive. By reducing the amount of memory available to save money, the amount of memory to store the past history of data is also reduced, limiting the ability of the predictive systems.

Once data has been collected by the system, some type of analysis needs to be performed. Many common analytical techniques measure information theoretic quantities such as information flow within a system. The objective of this project was to explore how these limitations affect the performance of various information theoretic calculations when they are supplied with short-length, coarse-grained, time-series data.

1.1 Time Series

Time series data was used extensively in this project, so a quick review of the properties of time series data will be useful to help with interpreting the results. The most general description of time series is a sequence of time ordered observations of some system. Consider making a recording of some music and storing it on a compact disc. Music is a continuous, analog, real-world processes, while information stored on a CD is digital and finite. In order store the music on the CD, the music is converted into a digital time series. At fixed intervals, (near) instantaneous measurements of the music's intensity are made. This may be accomplished by using a microphone to convert the music into an electrical signal, and measuring of the voltage of the signal. The intervals between measurements are related to the accuracy of the time series as a representation of the measured process. The shorter the interval between measurements, the better the representation, but the larger

the time series. These measurements are then discretized: they are converted to one of a finite number of digital values or symbols. The granularity of a time series observation is related to how many possible states each element (measurement) in the series may be selected from. The granularity is also related to accuracy. Increasing the number of states from which a measurement may be selected improves the accuracy of the time series representation, but it also increases the amount of information required to store the time series. For a compact disc, a single sample may be one of $2^{16} = 65536$ values, which is a relatively fine-grained observation. This project dealt with representations of time series that were extremely limited in the number of elements (measurements) and extremely coarse-grained, with each element having only one of two possible values.

1.2 Probability

In order to understand some of the computations that were performed for this project, it is useful to review some of the basic symbology and terminology used in the discussion of probability [3]. The probability per observation event that some value x occurs is written as $p(x)$. The quantity x may denote anything of interest. A common example is the value that comes up on a thrown die. The probability of rolling a six on one throw of a standard six-sided die, $p(6)$, is $1/6$. Another example is the chance of winning a prize in a lottery. Many lottery tickets have a statement on the back listing the probabilities of winning the various prizes. For example, at the time of writing, one "Pick 4" lottery [4] states that the probability of winning the jackpot, $p(\mathit{jackpot})$, is $1/10000$.

Probabilities may be added together. This is equivalent to performing a logical "OR"

operation: What is the probability of event 1 happening OR event 2 happening? For example, consider purchasing two "Pick 4" lottery tickets. Then the probability of winning with the first ticket would be $1/10000$, and the probability of winning with the second ticket would also be $1/10000$. So the total chance of winning with either the first OR the second ticket would be

$$\frac{1}{10000} + \frac{1}{10000} = \frac{2}{10000} = \frac{1}{5000} \tag{1}$$

Or, with the example of rolling dice, suppose a six-sided and a twenty-sided die are cast. The probability of rolling a six on either the first OR the second die would be

$$\frac{1}{20} + \frac{1}{6} = \frac{13}{60} \tag{2}$$

Probabilities may be multiplied together. This is equivalent to performing a logical "AND" operations: What is the probability of event 1 AND event 2 happening? This is one way to compute the chances of winning a "Pick 4" game. One way to think of it is the chance of selecting one four-digit number from all possible four-digit numbers (0000-9999). Since there are ten thousand four-digit numbers, the chance of picking one must be $1/10000$. However, this probability may be looked upon as the the probability of picking a particular combination of one-digit numbers (0-9). There are ten one-digit numbers, so the chances of picking the correct one are $1/10$. Picking the first and second and third and fourth numbers correctly may be written as

$$p(x_1) \cdot p(x_2) \cdot p(x_3) \cdot p(x_4) = \left(\frac{1}{10}\right)\left(\frac{1}{10}\right)\left(\frac{1}{10}\right)\left(\frac{1}{10}\right) = \frac{1}{10000} \tag{3}$$

Or, with the dice example, what is the probability of rolling a six on both a twenty- AND six-sided die? It would be

$$\left(\frac{1}{20}\right)\left(\frac{1}{6}\right) = \frac{1}{120} \tag{4}$$

Notationally, the probability of several events occurring is called a joint probability, and is written as $p(x,y)$. This is read as "The probability of events x and y occurring is $p(x,y)$".

The joint probability described above assumes that events 1 and 2 are unrelated with each other. That is, events 1 and 2 are said to be "independent". As an example, consider patrons in a movie theater buying items from the concession stand. On a given day, 80% of the patrons purchase just popcorn, while only 20% purchase just a soda. If popcorn and soda purchases were independant of each other, than the expected joint probability of a patron purchasing both popcorn and a soda would be

$$p(popcorn,\ soda) = p(popcorn) \cdot p(soda) = (0.8)(0.2) = 0.16 \tag{5}$$

However, suppose the theater owners observe that there is actually a 40% chance of a patron purchasing both popcorn and a soda. This value is different than the expected joint probability, so popcorn and soda purchasing must be related to each other. The two events are said to be "dependent" on each other. Using the known probability $p(x)$ of event x occurring and the joint probability $p(x,y)$ of the same event x and another event y ocurring, the relationship between y on x may be computed as

$$\frac{p(x,y)}{p(x)} = p(y|x) \tag{6}$$

where $p(y|x)$ is known as the "conditional probability". $p(y|x)$ is interpreted as the probability that event y will occur *given that event x has already occurred*. Using the theater example, the conditional probability that a patron will purchase soda given that they have purchased popcorn is

$$p(soda|popcorn) = \frac{p(popcorn,\ soda)}{p(popcorn)} = \frac{0.40}{0.80} = 0.50 \tag{7}$$

So there is 50% chance that a patron who purchases popcorn will then buy a soda.

Mathematically, event y is said to be independent of event x if $p(y|x) = p(y)$. Applying this to Equation 6 leads to the identity

$$p(x,y) = p(y|x) \cdot p(x) = p(y) \cdot p(x) \tag{8}$$

if two events are known to be independent. This relation does not hold if one of the events depends on the completion of the other in some way.

For many systems, the full range of outcomes is unknown. There may be several outcomes that are not known to exist, or their probability of occurrence may not be characterized. This would be like having a die where not all of the sides were visible. This project investigated systems where the probabilities of all outcomes were known. That is, there was a probability $p(x)$ associated with every outcome or possible value of x for the system. Because these systems were required to do something, and there were no unknown outcomes or probabilities, the sum over all probabilities was equal to one, $\sum_x p(x) = 1$.

1.3 Information Theory

Although information theory has early roots in late 19^{th} Century statistical theory, it was the publication of C. E. Shannon's paper [5] in 1948 that firmly established it as a scientific discipline. Information theory is based upon the idea that observations of a system should give the observer some information about the system. If many observations of a system were made and each observation returned the same result, then the observations after the first one would not yield any new information. But if different values were returned for the subsequent observations, that would be new information about the system and would be

useful.

Consider some dynamical system for which all potential outcomes for any observation are known. Suppose many observations were made on the system, with records kept concerning how many outcomes of each type were observed. Perhaps the possible outcomes for the system were Up, Yellow, and Christmas tree, and four Ups, two Yellows, and one Christmas tree were observed for a total of seven observations. The probability of seeing each event may be computed. For this example, that would be a $4/7 = 57\%$ chance of seeing an Up, a $2/7 = 29\%$ chance of seeing a Yellow, and only a $1/7 = 14\%$ chance of seeing a Christmas tree (and all the probabilities must add up to 100% since there must have been some result for each observation).

Based on intuitive arguments, one would suppose that outcomes which happen less frequently would yield more information about a system. Suppose many observations were made of some system. If the results of the subsequent observations were identical to the first observation, then the additional observations would not seem to be providing any new information about the system. However, if a few of those observations returned different values, then those rare values would seem to be providing new information (perhaps the rare values represented changes in the state of the system). Using several arguments, Shannon demonstrated that the additional information conveyed by an observation is proportional to the negative logarithm of the probability of seeing the particular outcome. Mathematically, the information of a particular event x is written as

$$I(x) = -\log_a(p(x)) \tag{9}$$

where x is an index over all possible outcomes, and $p(x)$ is the probability of seeing each

Observation	Number	Probability	Information (Decimal Digits)
Up	4	0.57	$-\log_{10}(0.57) = 0.24$
Yellow	2	0.29	$-\log_{10}(0.29) = 0.54$
Christmas tree	1	0.14	$-\log_{10}(0.14) = 0.85$

Table 1: Up/Yellow/Christmas tree Information

outcome *x*. The positive integer *a* is the base of the logarithm, but it also has a more subtle interpretation. It is possible to show the information for a particular event is the number of digits that would be required to represent the event in a numerical base, where the base of the logarithm is the base of the number system for the digits. Table 1 shows the results of the information calculations for the Up/Yellow/Christmas tree example. Based on these values, it would take at least 0.85 decimal digits to represent a Christmas tree value in a sequence of observations, but only 0.24 decimal digits to represent an Up value.

This example may be converted to binary. Since binary is a base-2 numbering system, the $\log_{10}$ values for information must be converted into $\log_2$ values. This may be accomplished by dividing the the decimal information values by $\log_{10}(2) = 0.30103$. Table 2 shows the binary information values. From the table, it would take 2.82 binary digits (bits) to represent a Christmas tree versus 0.79 bits for an Up observation.

Given a long sequence of Up/Yellow/Christmas tree observations, it is possible to determine the average, or expected, number of digits required to represent each observation in the sequence. This value, called the *Shannon Entropy* and written as H, may be defined

Observation	Decimal Digits	Binary Digits
Up	0.24	$0.24/0.30103 = 0.79$
Yellow	0.54	$0.54/0.30103 = 1.79$
Christmas tree	0.85	$0.85/0.30103 = 2.82$

Table 2: Binary Information Values

using Equation 9 as

$$H = \langle I(x) \rangle = \sum_x p(x) I(x) = -\sum_x p(x) \log_a(p(x)) \tag{10}$$

It is sum of the number of bits of information associated with each possible observation ($I(x)$) multiplied by the probability $p(x)$ of seeing that symbol in the string. Along with being the expected number of digits required to represent an observation, Shannon was also able to show that H is the *minimum* number of digits required to represent an observation. Using the Up/Yellow/Christmas tree example, the Shannon Entropy (computed with binary entropy values) is

$$H = -\sum_x p(x) \log_a(p(x)) = 0.79 \left(\frac{4}{7}\right) + 1.79 \left(\frac{2}{7}\right) + 2.82 \left(\frac{1}{7}\right) = 1.37 \tag{11}$$

For example, suppose the need existed to transmit a sequence of observations to another researcher. The transmission might be accomplished by converting the sequence of observations to a sequences of bits, and transmitting the bits. One way to perform the conversion would be to represent each symbol as a two bit number. This should be acceptable, since a two bit number may have four possible values, and there are only three symbols. Define $E(x)$ such that E returns the number of bits used to represent observation x. Then, with this

scheme

$$\langle E(x)\rangle = \sum_x p(x)E(x) = 2\sum_x p(x) = (2)(1) = 2 \tag{12}$$

since each symbol uses exactly two bits and the total probability of all symbols is 1. However, comparing this with H, the minimum number of bits required for the transmission, H, shows that this encoding scheme is inefficient. Observe that

$$\langle E(x)\rangle - \langle I(x)\rangle = 2 - 1.37 = 0.63 \tag{13}$$

indicated that, on average, 0.63 bits more than necessary are used to transmit each symbol. This is almost a 50% excess.

It is possible to improve the efficiency by selected a different conversion scheme. Since Up occurs with a greater frequency, it might be possible to reduce the number of wasted bits by representing Up with one bit instead of two. Let Up be represented by bit '0', Yellow by bits '10', and Christmas tree be represented by bits '11'. The average number of bits for this encoding scheme is

$$\langle E(x)\rangle = \sum_x p(x)E(x) = \left(\frac{4}{7}\right)1 + \left(\frac{2}{7}\right)2 + \left(\frac{1}{7}\right)2 = 1.43 \tag{14}$$

By letting Up be represented by only one bit, the excess number of bits used in the transmission is reduced to $1.43 - 1.37 = 0.06$ bits, a ten-fold improvement in efficiency. It is important to note that because H represents a lower bound, no matter what coding scheme is used

$$\langle E(x)\rangle \geq H \tag{15}$$

Further, $\langle E(x)\rangle = H$ only if $p(x) = a^{-j}$ for all x, where a is the base of the logarithm used to compute H and j is a positive integer.

1.4 Mutual Information

Using the Shannon entropy, it is possible to define a quantity M called the mutual information. Suppose there are two systems, each of which generates events. Let the probability of two events, one from each system, be described by $p(x,y)$. Then the Shannon entropy for the two systems would, in general, be

$$H_1 = -\sum_{x,y} p(x,y)\log(p(x,y)) \tag{16}$$

However, if the systems are independent of each other as defined in Section 1.2, then $p(x,y) = p(x)\cdot p(y)$, and the Shannon entropy would be

$$H_2 = -\sum_{x,y} p(x,y)\log(p(x)p(y)) \tag{17}$$

Here, the probability $p(x)\cdot p(y)$ has simply been substituted inside the logarithm expression. This is exactly the same thing that was done in the Up/Yellow/Christmas tree example, where different probability/information values were substituted into the Shannon entropy calculations.

Now, suppose H_1 is subtracted from H_2. This is exactly the same as the computations performed to find the number of wasted bits incurred from not using the precise Shannon entropy values.

$$\begin{aligned} H_2 - H_1 &= -\sum_{x,y} p(x,y)\log(p(x)p(y)) + \sum_{x,y} p(x,y)\log(p(x,y)) && (18)\\ &= \sum_{x,y} p(x,y)\left[\log(p(x,y)) - \log(p(x)p(y))\right] && (19)\\ &= \sum_{x,y} p(x,y)\log\left(\frac{p(x,y)}{p(x)p(y)}\right) && (20) \end{aligned}$$

The reduced equation represents the number of extra bits that are required to represent the output of the two systems as though they were independent as opposed to their actual

relationship. If the systems actually are independent of each other, then the value of $H_2 - H_1$ will equal zero. Otherwise, if the systems are somehow correlated or dependent on each other, then $H_2 - H_1$ will be a non-zero positive real number. The value $H_2 - H_1$ is known as the *mutual information*.

As an example, suppose there are two systems, 1 and 2, each of which generates a sequence of ones and zeros as shown below. Nothing is known about how the systems generated the sequences.

```
System 1:  11101101001100111101101
System 2:  10101010110110110110001
```

The mutual information calculation may be applied to the output of these systems to see if they are correlated. First, count the number of zeros and ones in the output from each system.

$$p(0_1) = \quad 8/22 \quad = 0.364 \tag{21}$$

$$p(1_1) = \quad 14/22 \quad = 0.636 \tag{22}$$

$$p(0_2) = \quad 9/22 \quad = 0.410 \tag{23}$$

$$p(1_2) = \quad 13/22 \quad = 0.590 \tag{24}$$

The subscripts indicate which measurement is associated with which system. Now take the pairs of numbers, one from each system, and count the different types of pairs. The data is written below with some spaces to try and highlight the pairs of numbers.

```
System 1:  1 1 1 0 1 1 0 1 0 0 1 1 0 0 1 1 1 0 1 1 0 1
System 2:  1 0 1 0 1 0 1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1
```

$$p(0_1,0_2) = \quad 3/22 \quad = 0.136 \tag{25}$$

$$p(0_1,1_2) = \quad 5/22 \quad = 0.227 \tag{26}$$

$$p(1_1,0_2) = \quad 6/22 \quad = 0.273 \tag{27}$$

$$p(1_1,1_2) = \quad 8/22 \quad = 0.364 \tag{28}$$

Using these probability values, it is easy to compute the mutual information associated with the two systems for the above data (Base-ten logarithms are used to make it easy to double-check).

$$M = 0.136\log\left(\frac{0.136}{0.364\cdot 0.410}\right) + 0.227\log\left(\frac{0.227}{0.364\cdot 0.590}\right) \tag{29}$$

$$+0.273\log\left(\frac{0.273}{0.636\cdot 0.410}\right) + 0.364\log\left(\frac{0.364}{0.636\cdot 0.590}\right) \tag{30}$$

$$= 6.08\times 10^{-4} \tag{31}$$

Since the mutual information is nonzero, these systems are in some way dependent on each other. One might suppose that because the mutual information is such a small value, that these systems are only very weakly coupled. This might be the case, but because the lengths of the sequences of ones and zeros are so small, there is no way to know for sure. This value of the mutual information might be close to the actual mutual information value for the system. It could represent an extremely high value for two systems that are coupled in an extremely weak fashion. Or it could represent an unusually low value for two systems that are actually very strongly coupled. One of the goals of this project was to determine how accurately the mutual information of short lengths of symbols represents a population mutual information value.

1.5 Transfer Entropy

There are a few problems with mutual information as a useful measure of correlation. One is that in the given form, the mutual information is not very effective at predicting future events from current data. One remedy is to time-shift one of the variables. In the given form, the mutual information is evaluated with simultaneous observations of Systems 1 and 2. When one of the variables is time shifted, it might be that the mutual information is evaluated with the current observed value of System 1, but with the value of System 2 that was observed five time-steps ago.

Another difficulty with the mutual information is that it does not indicate which way the information is flowing, or which system is driving which. Again, this may be remedied by time shifting one of the variables. Then it is possible to reason that because the earlier event occurred prior to the later one, the earlier event must be driving the later one. As Schreiber [6] notes , this is a "somewhat *ad hoc* way" of introducing a sense of directionality.

In his paper [6], Schreiber describes a new measure of statistical correlation he calls the *transfer entropy*. It is based on principles similar to the mutual information, but instead of being derived directly from the Shannon entropy, it is based on rates of entropy change. Therefore it can capture some of the dynamics of a system.

Suppose, as before, there are two systems, each of which generates events. In general, we may define an entropy rate (which is the amount of additional information required to represent the value of the next observation of one of the systems) in terms of both of the systems as follows,

$$h_1 = -\sum_{x_{n+1}} p(x_{n+1}, x_n, y_n) \log_a p(x_{n+1}|x_n, y_n) \tag{32}$$

where x and y represent the two systems, x_n is the value of system 1 at time n, y_n is the value of system 2 at time n, and x_{n+1} is the value of system 1 at time $n+1$. Suppose, however, that the value of observation x_{n+1} was not at all dependent on the current observation y_n, then

$$p(x_{n+1}|x_n,y_n) = p(x_{n+1}|x_n) \tag{33}$$

This may be substituted into h_1, resulting in

$$h_2 = -\sum_{x_{n+1}} p(x_{n+1},x_n,y_n)\log_a p(x_{n+1}|x_n) \tag{34}$$

In the mutual information calculations, the Shannon entropy for two systems was evaluated as though the systems were independent, and from this was subtracted the correct Shannon entropy for the two systems to determine the mutual information value. A similar operation may be performed here. The quantity h_1 represents the entropy rate for the two systems, and h_2 represents the entropy rate assuming that x_{n+1} is independent of y_n. Subtracting them yields

$$\begin{aligned} h_2 - h_1 &= -\sum_{x_{n+1},x_n,y_n} p(x_{n+1},x_n,y_n)\log_a p(x_{n+1}|x_n) \qquad (35)\\ &\quad +\sum_{x_{n+1},x_n,y_n} p(x_{n+1},x_n,y_n)\log_a p(x_{n+1}|x_n,y_n) \\ &= \sum_{x_{n+1},x_n,y_n} p(x_{n+1},x_n,y_n)\log_a\left(\frac{p(x_{n+1}|x_n,y_n)}{p(x_{n+1}|x_n)}\right) \qquad (36) \end{aligned}$$

which is what Schreiber called his *transfer entropy*. It is written as $T_{J\to I}$. Note that there are actually two equations for the transfer entropy because the transfer entropy has an inherent asymmetry to it.

$$\begin{aligned} T_{J\to I} &= \sum_{x_{n+1},x_n,y_n} p(x_{n+1},x_n,y_n)\log\left(\frac{p(x_{n+1}|x_n,y_n)}{p(x_{n+1}|x_n)}\right) \qquad (37)\\ T_{I\to J} &= \sum_{y_{n+1},x_n,y_n} p(y_{n+1},x_n,y_n)\log\left(\frac{p(y_{n+1}|x_n,y_n)}{p(y_{n+1}|y_n)}\right) \qquad (38) \end{aligned}$$

Equation 37 is concerned with predicting the $(n+1)^{\text{th}}$ symbol for string i, whereas equation 38 deals with predicting the $(n+1)^{\text{th}}$ symbol for string j. This is different from the mutual information, which only shows the total amount of information moving between the two systems. The transfer entropy is also inherently 'predictive' because it directly incorporates relationships between the n and $n+1$ symbols in a string.

Using the same set of data as was used for the mutual information calculation, a transfer entropy calculation may be performed. However, instead of using the above equations, two slightly different equations will be used. Using the definitions of conditional probability from Section 1.2, the substitutions $p(x_{n+1}|x_n,y_n) = p(x_{n+1},x_n,y_n)/p(x_n,y_n)$ and $p(x_{n+1}|x_n) = p(x_{n+1},x_n)/p(x_n)$ may be performed on $T_{J\to I}$. Similar transformations apply to the $T_{I\to J}$ equation. With these substitutions, these equations become

$$T_{J\to I} = \sum_{x_{n+1},x_n,y_n} p(x_{n+1},x_n,y_n)\log\left(\frac{p(x_{n+1},x_n,y_n)\cdot p(x_n)}{p(x_n,y_n)\cdot p(x_{n+1},x_n)}\right) \quad (39)$$

$$T_{I\to J} = \sum_{y_{n+1},x_n,y_n} p(y_{n+1},x_n,y_n)\log\left(\frac{p(y_{n+1},x_n,y_n)\cdot p(y_n)}{p(x_n,y_n)\cdot p(y_{n+1},y_n)}\right) \quad (40)$$

Here is a copy of the data to be analyzed.

```
System 1:  1 1 1 0 1 1 0 1 0 0 1 1 0 0 1 1 1 0 1 1 0 1
System 2:  1 0 1 0 1 0 1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1
```

This will be the calculation for $T_{J\to I}$. First determine $p(x_{n+1},x_n,y_n)$.

$$p(0_x,0_x,0_y) = 0.0 \quad (41)$$

$$p(0_x,0_x,1_y) = 0.0952381 \quad (42)$$

$$p(0_x,1_x,0_y) = 0.190476 \quad (43)$$

$$p(0_x,1_x,1_y) = 0.0952381 \quad (44)$$

$$p(1_x, 0_x, 0_y) = 0.142857 \quad (45)$$

$$p(1_x, 0_x, 1_y) = 0.142857 \quad (46)$$

$$p(1_x, 1_x, 0_y) = 0.0952381 \quad (47)$$

$$p(1_x, 1_x, 1_y) = 0.238095 \quad (48)$$

Note that the subscripts in the probabilities indicate which system each of the values is taken from. Now calculate $p(x_{n+1}, x_n)$

$$p(0,0) = 0.0952381 \quad (49)$$

$$p(0,1) = 0.285714 \quad (50)$$

$$p(1,0) = 0.285714 \quad (51)$$

$$p(1,1) = 0.333333 \quad (52)$$

Finally, we compute $p(x_n, y_n)$ and $p(x)$.

$$p(0,0) = 0.136364 \quad (53)$$

$$p(0,1) = 0.227273 \quad (54)$$

$$p(1,0) = 0.272727 \quad (55)$$

$$p(1,1) = 0.363636 \quad (56)$$

$$p(0) = 0.363636 \quad (57)$$

$$p(1) = 0.636364 \quad (58)$$

Evaluating $T_{J\to I}$ with these probability values and a log base of 10 yields $T_{J\to I} = 0.0440$. A similar computation may be performed for $T_{I\to J}$ with the result $T_{I\to J} = 0.0113$. This

is interpreted as system J adding 0.0440 digits of predictability to system I, and system I adding 0.0113 digits of predictability to J.

2 Prior Work

A substantial amount of work has already been done on coarse-grained time series data and measuring correlations within them. In 1980 Kedem published *Binary Time Series* [7] which provided an extensive treatment of the properties of binary time series data, including non-information-theoretic methods of measuring correlation and predicting future values in such series. Wentian Li's 1989 paper titled "Mutual Information Functions versus Correlation Functions" [8] derived an exact relationship between the mutual information function $M(d)$ and the correlation function $\Gamma(d)$ for binary time series. Li also discussed the effects of finite sample size on the correlation returned by the mutual information function. More recently papers by Herzel and Große [9] and Benedetto, Caglioti, and Loreto [10] have discussed the application of information theoretic analysis methods to other types of coarse-grain series. Herzel and Große use DNA strands as their data, while Benedetto, Caglioti, and Loreto analyzed and classified written languages.

Finally, Mark Roulston [11] generated statistical data based on several probability distributions, and analyzed the deviation of the Shannon entropy of the resulting data from the expected values based on the probability distributions. This experiment was similar to the one performed in this project, except that here other information theoretic measures will be used instead of the Shannon entropy. Also, Roulston used relatively large series (sets of 100 and 1000 data points) for his calculations. This project will explore series with ten

elements or fewer.

3 The Numerical Experiments

There were two goals for this project. The first was to evaluate the mutual information as a measure of statistical correlation between short, coarse-grained times series. This was to be accomplished by observing the deviation of the mutual information from some expected value as the number of symbols used to compute the mutual information decreased. The second goal was to evaluate the transfer entropy as a measure of statistical correlation under the same conditions. These tasks were to be performed by a computer program.

3.1 The Systems

In order to evaluate the mutual information and transfer entropy as measures of correlation, a way of generating pairs of coupled time series was required. The decision was made to use first-order Markov processes to generate the time series. There were two reasons for this choice. The first was that the properties of Markov processes have been characterized and they are well understood. The second was that because the tests for this project were to be performed by computational simulation, it would be useful to use algorithms that would be easy to simulate and debug. Markov processes are relatively easy to simulate and maintain.

Although Markov processes are well characterized, an important component of this project was to characterize them from an information theoretic point of view. To determine how far off a measure of correlation is from some expected value, one needs to have the

expected value to compare. For the measures of correlation considered in the project, that expected value would be the value of the correlation measure applied to two very long time series. Conveniently, many of the long-term statistical properties of the output of a Markov process may be determined from the process itself, without generating long sequences of outputs. For example, the probability distribution of the symbols in a long output may be found by computing the eigenvector of the transition matrix for the Markov process. A large portion of the work for this project went to developing algorithms to extract from the Markov processes the long-term, or "population", values of the various measures of correlation. Hand-in-hand with this characterization, work was done to parameterize the Markov processes to allow the correlation between the two output sequences to be varied in a controlled fashion.

The output of the Markov processes used in this project were two sequences of binary symbols. These are the two artificial time series. It was decided that the sequences would be binary (as opposed to some other coarse-graining, such as a three or four state system) for two reasons. One, Kedem [7] and Li [8] had already done substantial work on the properties of binary time series, so their properties were already understood. Two, because this project was concerned in part with measuring deviations due to coarse-graining, it was believed that selecting the coarsest graining possible would enhance any observable effects. Each of the sequences is of the same length, called the "window size", and denoted by the parameter *w*. As an example, consider the sequences of ones and zeros used for the calculations in Sections 1.4 and 1.5. These are two binary sequences, and the window size for the pair, *w*, equals 22. As another example, consider Table 3. Here, **x** and **y** represent the two different binary sequences. Each element of each sequence may be a zero or a

k	0	1	2	...	n	...	$(w-1)$
x	1	0	1	...	$\{0,1\}$	...	1
y	0	1	1	...	$\{0,1\}$	...	0

Table 3: Another example of a time series

one. The particular elements of each sequence may be referenced by the variable n where $0 < n < w$. In Table 3, $x_1 = 0$ and $y_{w-1} = 0$.

3.2 Mutual Information

The Markov model for the mutual information was relatively easy to construct. Recall the definition of the mutual information from Section 1.4. Using the index n introduced in Table 3, this definition may be rewritten as

$$M = \sum_{n=0}^{w-1} p(x_n, y_n) \log\left(\frac{p(x_n, y_n)}{p(x_n) \cdot p(y_n)}\right) \tag{59}$$

Notice that the mutual information only depends on $p(x_n, y_n)$, $p(x_n)$, and $p(y_n)$, where x_n and y_n are elements of the two time series **x** and **y**. Observe that because zero and one are the only values that may appear in either **x** and **y**, there exists the equalities

$$p(0_x, 0_y) + p(0_x, 1_y) = p(0_x) \tag{60}$$

$$p(1_x, 0_y) + p(1_x, 1_y) = p(1_x) \tag{61}$$

$$p(0_x, 0_y) + p(1_x, 0_y) = p(0_y) \tag{62}$$

$$p(0_x, 1_y) + p(1_x, 1_y) = p(1_y) \tag{63}$$

$p(0_x, 0_y)$	$p(0_x, 1_y)$	$-p(0_x)$
$p(1_x, 0_y)$	$p(1_x, 1_y)$	$-p(1_x)$
$-p(0_y)$	$-p(1_y)$	1

Table 4: Static Probability Table for the Mutual Information. The subscripts indicate which system is contributing which symbol to the probabilities.

Further

$$p(0_x) + p(1_x) = 1 \tag{64}$$

$$p(0_y) + p(1_y) = 1 \tag{65}$$

Also, it may be seen that the value of the mutual information does not depend on the order of the pairs (x_n, y_n). Any two pairs may swap position without changing the mutual information between the two series. Because the mutual information is not dependent on the order of the symbols, it does not encode any of the dynamics of the system. Therefore, instead of representing the Markov process for this system with a transition matrix, it may be represented as a table of relationships between static probability values as shown in Table 4, where the sum of the elements of any row or column is zero. Note that although certain probabilities appear in a negative form in this table, the probabilities themselves are not negative; all probabilities are positive. They appear in a negative form in this table to demonstrate the relationships between the values.

Using these relationships, it is straightforward to compute the mutual information of a time series. First, initialize all of the joint probabilities to zero, and then for each pair (x_n, y_n) in the time series, add one to the appropriate joint probability entry. When all the

points have been read in, divide each joint probability by the total number of pairs of points, w. Then each joint probability is exactly the probability of that pair of symbols occurring in the time series, and the probability of any particular symbol occurring in an array may be solved for using the equalities. Once that is done, all that remains is to evaluate Equation 59 using the values specified in the table.

Creating a model to generate time series with a particular mutual information was also relatively straightforward. Given the above relationships, the user could have been asked to supply the various values of the probabilities, thus generating a model with a specific mutual information. Even though there are eight variables in the probability matrix, only three of them are free parameters. This still provides for a large number of different types of matrices to be tested. In order to reduce the number of calculations, two further constraints were placed on the user model. The first constraint was that each of the individual symbols must occur with equal frequency.

$$p(a) = p(b) = p(c) = p(d) = \frac{1}{2} \tag{66}$$

The second constraint defined $p(0_x, 0_y)$ to be $\gamma/2$, where γ is a free, real-valued parameter which may range from 0.0 to 1.0. Solving the equalities results in the joint probabilities that have the values

$$p(0_x, 0_y) = \frac{\gamma}{2} \tag{67}$$

$$p(0_x, 1_y) = \frac{1-\gamma}{2} \tag{68}$$

$$p(1_x, 0_y) = \frac{1-\gamma}{2} \tag{69}$$

$$p(1_x, 1_y) = \frac{\gamma}{2} \tag{70}$$

These equations may also represented in a table format, as shown in Table 5.

$\gamma/2$	$(1-\gamma)/2$	$-1/2$
$(1-\gamma)/2$	$\gamma/2$	$-1/2$
$-1/2$	$-1/2$	1

Table 5: Constrained Mutual Information Static Probability Table

These constrains limit the number of free parameters in the model to just one, γ. However, even though it only has one parameter, the model covers a full range of possible mutual information values. If $\gamma = 0.0$, then only symbol combinations $(0_x, 1_y)$ and $(1_x, 0_y)$ will have non-zero probabilities of occurring. Conversely, if $\gamma = 1.0$, then only symbol combinations $(0_x, 0_y)$ and $(1_x, 1_y)$ will occur. In both of these cases, the mutual information values computed from the model are equal, with a value $M = 1.0$ when computed using $\log_2$. In addition, if $\gamma = 0.5$, then all combinations are assigned the same probability of occurring, $p = 0.25$, and $M = 0.0$. Further, since 0_x and 0_y can be homomorphically transformed to 1_x and 1_y respectively, the mutual information for this system should be distributed symmetrically around $\gamma = 0.5$ as $\gamma : 0.0 \rightarrow 1.0$.

While this model represented a very convenient parameterization of the mutual information calculations, there was one problem. The all of the entries in the table are static probability values, and it proved impractical to generate dynamic time series from these static values. Instead, an ensemble of time series of some user-specified window size w was generated. Each realization of the ensemble was one of the pairs of time series as shown in Table 3 with length w. The entire ensemble consisted of all possible combination of ones and zeros for each of the time series. For each member of the ensemble, it was possible to compute the mutual information as detailed above. Based on the user-specified

value for γ, a Markov probability table was generated. Only one value of γ was used for the entire ensemble. Observe that the probability of the model generating a particular pair of time series $\mathbf{x}$ and $\mathbf{y}$, $p(\mathbf{x},\mathbf{y})$, may be computed as

$$p(\mathbf{x},\mathbf{y}) = \prod_{n=0}^{w-1} p(x_n, y_n) \tag{71}$$

where $p(x_n, y_n)$ may be computed from the user-specified Markov probability table.

Therefore, for any realization of the ensemble, there exists a tuple (M, p), where M is is the mutual information computed for the realization, and p is the probability that the user-specified model will generate that particular realization. Since $p(\mathbf{x},\mathbf{y})$ was evaluated for all the members of the ensemble

$$\sum_{x,y} p(\mathbf{x},\mathbf{y}) = 1 \tag{72}$$

Because the sum of the probabilities is one, it is possible to generate a cumulative distribution function from the probabilities of the various realizations of the ensemble. Using the cumulative distribution, it is possible to compute the 50%-ile of the mutual information. That is, the value of mutual information that is greater or equal to the mutual information of at least half the ensamble.

3.3 Transfer Entropy

Devising a Markov model from which transfer entropy values may be computed was slightly more difficult than devising the comparable matrix for the mutual information. Recall Schreiber's equations for the transfer entropy (Equations 37 and 38)

$$T_{J\to I} = \sum_{x_{n+1},x_n,y_n} p(x_{n+1},x_n,y_n) \log\left(\frac{p(x_{n+1}|x_n,y_n)}{p(x_{n+1}|x_n)}\right)$$

$p(0'_x, 0'_y \mid 0_x, 0_y)$	$p(0'_x, 0'_y \mid 0_x, 1_y)$	$p(0'_x, 0'_y \mid 1_x, 0_y)$	$p(0'_x, 0'_y \mid 1_x, 1_y)$
$p(0'_x, 1'_y \mid 0_x, 0_y)$	$p(0'_x, 1'_y \mid 0_x, 1_y)$	$p(0'_x, 1'_y \mid 1_x, 0_y)$	$p(0'_x, 1'_y \mid 1_x, 1_y)$
$p(1'_x, 0'_y \mid 0_x, 0_y)$	$p(1'_x, 0'_y \mid 0_x, 1_y)$	$p(1'_x, 0'_y \mid 1_x, 0_y)$	$p(1'_x, 0'_y \mid 1_x, 1_y)$
$p(1'_x, 1'_y \mid 0_x, 0_y)$	$p(1'_x, 1'_y \mid 0_x, 1_y)$	$p(1'_x, 1'_y \mid 1_x, 0_y)$	$p(1'_x, 1'_y \mid 1_x, 1_y)$

Table 6: Transfer Entropy Transition Matrix (primed values are from observation $n+1$, unprimed values are from observation n)

$$T_{I\to J} = \sum_{y_{n+1},x_n,y_n} p(y_{n+1},x_n,y_n)\log\left(\frac{p(y_{n+1}|x_n,y_n)}{p(y_{n+1}|y_n)}\right)$$

These equations contain explicit references to the the current set of observations (x_n, y_n) and the future set of observations (x_{n+1}, y_{n+1}). Therefore these equations can record some of the dynamics of what is being observed. As a consequence, the Markov model for the system must be represented as a transition matrix. There are four possible sets of observations that may be made at time n: $(0_x, 0_y)$, $(0_x, 1_y)$, $(1_x, 0_y)$, $(1_x, 1_y)$. These same four series of observations may also be made at time $n+1$. Thus the Markov process for the transfer entropy may be described by a four-by-four transition matrix shown in Table 6.

While the matrix in Table 6 can properly describe a Markov process from which a transfer entropy may be calculated, it is not the most useful presentation of the data. Instead, suppose each conditional probability in the table was converted into a four-element joint probability by multiplying the conditional probability with an appropriate two-element joint probability (using the relation $p(a,b,c,d) = p(a,b|c,d)\cdot p(c,d)$). The result would be the joint probability matrix show in Table 7.

$p(0'_x, 0'_y, 0_x, 0_y)$	$p(0'_x, 0'_y, 0_x, 1_y)$	$p(0'_x, 0'_y, 1_x, 0_y)$	$p(0'_x, 0'_y, 1_x, 1_y)$
$p(0'_x, 1'_y, 0_x, 0_y)$	$p(0'_x, 1'_y, 0_x, 1_y)$	$p(0'_x, 1'_y, 1_x, 0_y)$	$p(0'_x, 1'_y, 1_x, 1_y)$
$p(1'_x, 0'_y, 0_x, 0_y)$	$p(1'_x, 0'_y, 0_x, 1_y)$	$p(1'_x, 0'_y, 1_x, 0_y)$	$p(1'_x, 0'_y, 1_x, 1_y)$
$p(1'_x, 1'_y, 0_x, 0_y)$	$p(1'_x, 1'_y, 0_x, 1_y)$	$p(1'_x, 1'_y, 1_x, 0_y)$	$p(1'_x, 1'_y, 1_x, 1_y)$

Table 7: Transfer Entropy Probability Matrix

Recall that Equations 39 and 40

$$
\begin{aligned}
T_{J\to I} &= \sum_{x_{n+1},x_n,y_n} p(x_{n+1},x_n,y_n)\log\left(\frac{p(x_{n+1},x_n,y_n)\cdot p(x_n)}{p(x_n,y_n)\cdot p(x_{n+1},x_n)}\right)\\
T_{I\to J} &= \sum_{y_{n+1},x_n,y_n} p(y_{n+1},x_n,y_n)\log\left(\frac{p(y_{n+1},x_n,y_n)\cdot p(y_n)}{p(x_n,y_n)\cdot p(y_{n+1},y_n)}\right)
\end{aligned}
$$

were a modified form of Schreiber's transfer entropy equations from which all of the conditional probabilities had been eliminated. As the following equations show, it is possible to compute all of the values necessary to evaluate $T_{J\to I}$ from the probability entries in Table 7.

$$p(0'_x, 0_x, 0_y) = p(0'_x, 0'_y, 0_x, 0_y) + p(0'_x, 1'_y, 0_x, 0_y) \tag{73}$$

$$p(0'_x, 0_x, 1_y) = p(0'_x, 0'_y, 0_x, 1_y) + p(0'_x, 1'_y, 0_x, 1_y) \tag{74}$$

$$p(0'_x, 1_x, 0_y) = p(0'_x, 0'_y, 1_x, 0_y) + p(0'_x, 1'_y, 1_x, 0_y) \tag{75}$$

$$p(0'_x, 1_x, 1_y) = p(0'_x, 0'_y, 1_x, 1_y) + p(0'_x, 1'_y, 1_x, 1_y) \tag{76}$$

$$p(1'_x, 0_x, 0_y) = p(1'_x, 0'_y, 0_x, 0_y) + p(1'_x, 1'_y, 0_x, 0_y) \tag{77}$$

$$p(1'_x, 0_x, 1_y) = p(1'_x, 0'_y, 0_x, 1_y) + p(1'_x, 1'_y, 0_x, 1_y) \tag{78}$$

$$p(1'_x, 1_x, 0_y) = p(1'_x, 0'_y, 1_x, 0_y) + p(1'_x, 1'_y, 1_x, 0_y) \tag{79}$$

$$p(1'_x, 1_x, 1_y) = p(1'_x, 0'_y, 1_x, 1_y) + p(1'_x, 1'_y, 1_x, 1_y) \tag{80}$$

$$p(0'_x, 0_x) = p(0'_x, 0_x, 0_y) + p(0'_x, 0_x, 1_y) \quad (81)$$

$$p(0'_x, 1_x) = p(0'_x, 1_x, 0_y) + p(0'_x, 1_x, 1_y) \quad (82)$$

$$p(1'_x, 0_x) = p(1'_x, 0_x, 0_y) + p(1'_x, 0_x, 1_y) \quad (83)$$

$$p(1'_x, 1_x) = p(1'_x, 1_x, 0_y) + p(1'_x, 1_x, 1_y) \quad (84)$$

$$p(0_x, 0_y) = p(0'_x, 0_x, 0_y) + p(1'_x, 0_x, 0_y) \quad (85)$$

$$p(0_x, 1_y) = p(0'_x, 0_x, 1_y) + p(1'_x, 0_x, 1_y) \quad (86)$$

$$p(1_x, 0_y) = p(0'_x, 1_x, 0_y) + p(1'_x, 1_x, 0_y) \quad (87)$$

$$p(1_x, 1_y) = p(0'_x, 1_x, 1_y) + p(1'_x, 0_x, 1_y) \quad (88)$$

$$p(0_x) = p(0_x, 0_y) + p(0_x, 1_y) \quad (89)$$

$$p(1_x) = p(1_x, 0_y) + p(1_x, 1_y) \quad (90)$$

A similar set of equations shows it is possible to compute all of the values necessary to evaluate $T_{I \to J}$ from the probability entries in Table 7.

$$p(0'_y, 0_x, 0_y) = p(0'_x, 0'_y, 0_x, 0_y) + p(1'_x, 0'_y, 0_x, 0_y) \quad (91)$$

$$p(0'_y, 0_x, 1_y) = p(0'_x, 0'_y, 0_x, 1_y) + p(1'_x, 0'_y, 0_x, 1_y) \quad (92)$$

$$p(0'_y, 1_x, 0_y) = p(0'_x, 0'_y, 1_x, 0_y) + p(1'_x, 0'_y, 1_x, 0_y) \quad (93)$$

$$p(0'_y, 1_x, 1_y) = p(0'_x, 0'_y, 1_x, 1_y) + p(1'_x, 0'_y, 1_x, 1_y) \quad (94)$$

$$p(1'_y, 0_x, 0_y) = p(0'_x, 1'_y, 0_x, 0_y) + p(1'_x, 1'_y, 0_x, 0_y) \quad (95)$$

$$p(1'_y, 0_x, 1_y) = p(0'_x, 1'_y, 0_x, 1_y) + p(1'_x, 1'_y, 0_x, 1_y) \quad (96)$$

$$p(1'_y, 1_x, 0_y) = p(0'_x, 1'_y, 1_x, 0_y) + p(1'_x, 1'_y, 1_x, 0_y) \quad (97)$$

$$p(1'_y, 1_x, 1_y) = p(0'_x, 1'_y, 1_x, 1_y) + p(1'_x, 1'_y, 1_x, 1_y) \quad (98)$$

$$p(0'_y, 0_y) = p(0'_y, 0_x, 0_y) + p(0'_y, 1_x, 0_y) \quad (99)$$

$$p(0'_y, 1_y) = p(0'_y, 0_x, 1_y) + p(0'_y, 1_x, 1_y) \quad (100)$$

$$p(1'_y, 0_y) = p(1'_y, 0_x, 0_y) + p(1'_y, 1_x, 0_y) \quad (101)$$

$$p(1'_y, 1_y) = p(1'_y, 0_x, 1_y) + p(1'_y, 1_x, 1_y) \quad (102)$$

$$p(0_y) = p(0_x, 0_y) + p(1_x, 0_y) \quad (103)$$

$$p(1_y) = p(0_x, 1_y) + p(1_x, 1_y) \quad (104)$$

Three observations may be made about the relationship between the various entries of Table 7. One is that the calculations for the joint probabilities of the type $p(x_n,y_n)$ are the same for both $T_{I\to J}$ and $T_{J\to I}$, so they were not repeated in the second set of equations. The second is that computing $p(x_n,y_n)$ is equivalent to summing a column of the matrix, it is also equivalent to summing a row of the matrix. Summing over the first row is equivalent in summing over the x_n and y_n indices, which yields $p(0'_x, 0'_y)$. However, because the probabilities under consideration are time-shift invariant, this is equivalent to $p(0_x, 0_y)$ obtained from summing the first column of the matrix. A similar logic applies to the other rows and columns. Finally, note that summing over the elements of the first two columns

is equivalent to summing over the x_{n+1}, y_{n+1}, and y_n indices, resulting in the probability $p(0_x)$. $p(1_x)$ may be determined by summing over columns three and four. Summing over the first two and second two rows yield the same results.

Using these relationships, it is possible to compute the two transfer entropies of a time series, in a manner similar to computing the mutual information. First, initialize all of the joint probabilities to zero, and then for each quadruplet $(x_{n+1}, y_{n+1}, x_n, y_n)$ in the time series, add one to the appropriate joint probability entry. When all the points have been read in, divide each joint probability by the total number of quadruplets of points, *w*. Then each four-element joint probability is set, and Equations 39 and 40 may be solved.

Developing a model to generate time series with a particular set of transfer entropies has been much harder then developing one for the mutual information, and has yet to be completed as of this writing. Obviously, the user could supply all sixteen entries for Table 7, but this would result in a huge number of possible matrices to evaluate. However, because one of the goals of this project was to compare the the mutual information and transfer entropy as measures of correlation, it was decided to consider those system with no mutual information but with non-zero transfer entropy.

One way to force a system to have zero mutual information is to ensure the equality

$$p(x_n, y_n) = p(x_n) \cdot p(y_n) \tag{105}$$

always holds. This may be accomplished by setting the following

$$p(0_x) = p(1_x) = p(0_y) = p(1_y) \quad = \quad 0.5 \tag{106}$$

$$p(0_x, 0_y) = p(0_x, 1_y) = p(1_x, 0_y) = p(1_x, 1_y) \quad = \quad 0.25 \tag{107}$$

Applying these constrains to the joint probability matrix yields Table 8, where the sum of

$p(0'_x, 0'_y, 0_x, 0_y)$	$p(0'_x, 0'_y, 0_x, 1_y)$	$p(0'_x, 0'_y, 1_x, 0_y)$	$p(0'_x, 0'_y, 1_x, 1_y)$	-0.25
$p(0'_x, 1'_y, 0_x, 0_y)$	$p(0'_x, 1'_y, 0_x, 1_y)$	$p(0'_x, 1'_y, 1_x, 0_y)$	$p(0'_x, 1'_y, 1_x, 1_y)$	-0.25
$p(1'_x, 0'_y, 0_x, 0_y)$	$p(1'_x, 0'_y, 0_x, 1_y)$	$p(1'_x, 0'_y, 1_x, 0_y)$	$p(1'_x, 0'_y, 1_x, 1_y)$	-0.25
$p(1'_x, 1'_y, 0_x, 0_y)$	$p(1'_x, 1'_y, 0_x, 1_y)$	$p(1'_x, 1'_y, 1_x, 0_y)$	$p(1'_x, 1'_y, 1_x, 1_y)$	-0.25
-0.25	-0.25	-0.25	-0.25	1

Table 8: Constrained Transfer Entropy Probability Matrix

all the rows and columns must be zero.

The addition of these constraints reduces the number free parameters in the matrix from sixteen to nine, however that was still too many free parameters to model. As a way to eliminate additional variables, it seemed reasonable to determine those conditions for which the transfer entropy of the matrix would be zero, and then to try and perturb away from those conditions. Consider the $T_{J\to I}$ transfer entropy. While in principle any combination of probabilities might result in it having a zero value, it is reasonable to look at a subset of combinations. If every term in the sum for $T_{J\to I}$ is zero, then the total will also be zero. There are two ways for a term of the sum to be zero, either

$$p(x_{n+1}, x_n, y_n) = 0 \tag{108}$$

or

$$\frac{p(x_{n+1}, x_n, y_n) \cdot p(x_n)}{p(x_n, y_n) \cdot p(x_{n+1}, x_n)} = 1 \tag{109}$$

Equation 108 is the trivial case. This implies that no instance of (x_{n+1}, x_n, y_n) exists in the input. While this could be true for a special case, in general it is not likely to happen.

Instead, consider Equation 109. In order for it to hold

$$p(x_{n+1},x_n,y_n)\cdot p(x_n) = p(x_n,y_n)\cdot p(x_{n+1},x_n) \tag{110}$$

Since the constraints that $p(x_n) = 0.5$ and $p(x_n,y_n) = 0.25$ were already applied, this is equivalent to

$$2\cdot p(x_{n+1},x_n,y_n) = p(x_{n+1},x_n) \tag{111}$$

However, if a sum is taken over all possible values of a variable in a joint probability, that variable may be eliminated from the joint probability. Therefore

$$p(x_{n+1},x_n) = \sum_{y_n} p(x_{n+1},x_n,y_n) \tag{112}$$

Because y_n was defined to have only two possible states, there are only two terms in the the summation, and Equation 112 may be rewritten as

$$p(x_{n+1},x_n) = p(x_{n+1},x_n,y_n) + p(x_{n+1},x_n,\tilde{y_n}) \tag{113}$$

where $\tilde{y_n}$ is the compliment of y_n. That is, whatever value y_n takes, $\tilde{y_n}$ takes the other possible value. Substituting this into Equation 111 yields

$$2\cdot p(x_{n+1},x_n,y_n) = p(x_{n+1},x_n,y_n) + p(x_{n+1},x_n,\tilde{y_n}) \tag{114}$$

$$p(x_{n+1},x_n,y_n) = p(x_{n+1},x_n,\tilde{y_n}) \tag{115}$$

Again noting that the sum over all possible values of a variable in a joint probability eliminates that variable, Equation 115 may be written as

$$\sum_{y_{n+1}} p(x_{n+1},y_{n+1},x_n,y_n) = \sum_{y_{n+1}} p(x_{n+1},y_{n+1},x_n,\tilde{y_n}) \tag{116}$$

$$v = \begin{bmatrix} p(0'_x, 0'_y, 0_x, 0_y) \\ p(0'_x, 1'_y, 0_x, 0_y) \\ \vdots \\ p(1'_x, 0'_y, 1_x, 1_y) \\ p(1'_x, 1'_y, 1_x, 1_y) \end{bmatrix} \tag{117}$$

Table 9: Vector v

The last expression states that the first sum of the first two cells in the first column of the matrix must equal the sum of the first two cells in the second column of the matrix (or the sums of the first two cells in the third and fourth columns must be equal, and so on). Similarly, the equations may be solved for $T_{I\to J}$, yielding similar results, only acting on the first and third and second fourth cells of the appropriate columns. Note that this expresses a linear relationships between probability values. Although the transfer entropy equation is nonlinear, the equations that are being used are linear, and may be analyzed using linear algebra.

To do this, the relationships may be represented as a matrix. Table 7 may be written as a vector v with sixteen entries as shown in Table 9. A second matrix A may be constructed which summarizes the constrains to be imposed on v, as shown in Table 10. The first eight rows of A represent the constraint that the sum of each row and each column of Table 8 must be 0.25 to fix the mutual information at zero. In this case, these rows are set so that when they act on the appropriate vector, the first eight terms of the result will be ones. The second eight rows of A represent the constraint that the various sets of cells must satisfy so that there is zero transfer entropy. These rows are set so that the second eight terms of

$$A = \begin{bmatrix}
4 & 4 & 4 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 4 & 4 & 4 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 4 & 4 & 4 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 4 & 4 & 4 \\
4 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \\
0 & 4 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 4 & 0 & 0 \\
0 & 0 & 4 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 4 & 0 \\
0 & 0 & 0 & 4 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 4 \\
 & & & & & & & & & & & & & & & \\
1 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & -1 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & -1 & -1 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1
\end{bmatrix} \tag{118}$$

Table 10: Constraint Matrix for Transfer Entropy Calculation. The empty space separates the first eight and last eight rows.

the result vector will be zero if there is zero transfer entropy. A non-zero result in result elements nine though twelve indicates $T_{J\to I}$ is nonzero. A non-zero result in elements thirteen through sixteen indicates that $T_{I\to J}$ is nonzero.

Solving the linear equation $Av = 0$ reveals the null space of the matrix has five dimensions, indicating that there should be five free parameters in Table 8 similar to γ from the mutual information calculations. However, there does not appear a simple interpretation for them. In order to display the parameters, it is possible to solve the linear equation $Av = b$, where $b = [1111111100000000]^{\mathrm{T}}$ indicating the result for a system with no transfer entropy. Solving this equation using the `Maple` computer algebra program yielded a sixteen element vector parameterized with five variables. It is easier to show the results if the sixteen-dimensional vector v is converted back into a set of four-by-four matrices as shown in Table 11. For these matrices the j values represent the five free parameters. Matrices that meet the imposed conditions may be constructed from linear combinations of the matrices.

The set of matrices shown in Table 11 represent the extent of the work that has been accomplished on this part of the project. As can be seen, the matrices do not show any simple pattern that would indicate how to divide up the five parameters in a manner similar to what is seen in the mutual information calculations. However, there are still some constraint equations which have not been incorporated into these calculations. For the entries of v (Table 8) to be interpreted as probabilities, they all must lie between zero and one. Any set of values for the j parameters will allow v to meet the restrains imposed on it, including those that result in elements of v that are not interpretable as probabilities. Because the full development of this model has not been completed, it was not possible to evaluate the

$$
v = j_1 \begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 \end{bmatrix} + j_2 \begin{bmatrix} -1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 1 & 0 & -1 & 0 \\ -1 & 0 & 1 & 0 \end{bmatrix}
$$

$$
+ j_3 \begin{bmatrix} 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + j_4 \begin{bmatrix} 1 & 1 & -1 & -1 \\ 0 & 0 & 0 & 0 \\ -1 & -1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}
$$

$$
+ j_5 \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0.25 \\ 0 & 0 & 0.25 & 0 \\ 0 & 0.25 & 0 & 0 \\ 0.25 & 0 & 0 & 0 \end{bmatrix}
$$

Table 11: Constraint Matrices for v

Mutual Information	Probability
0.000	0.317
0.123	0.173
0.311	0.374
0.811	0.078
1.000	0.058
Total	**1.000**

Table 12: Probability Distribution for a system where $w = 4$, $\gamma = 0.4$

transfer entropy as a measure of statistical correlation for short, coarse-grained time series at the time of writing.

4 Numerical Results

The only results that have been collected by this project have been for the evaluation of the mutual information as a measure of statistical correlation for short, coarse-grained time series. A computer program was written that accepts a model matrix and a window size, and performs the operations described in Section 3.2. For example, when the program was set to compute the mutual information for time series of length $w = 4$, where the model parameter $\gamma = 0.4$, $4^4 = 256$ mutual information/probability pairs were generated. Table 12 shows the results of sorting those pairs and summing the probability of those pairs with the same mutual information.

Figure 1 showing the fiftieth-percentile cumulative mutual information values for mul-

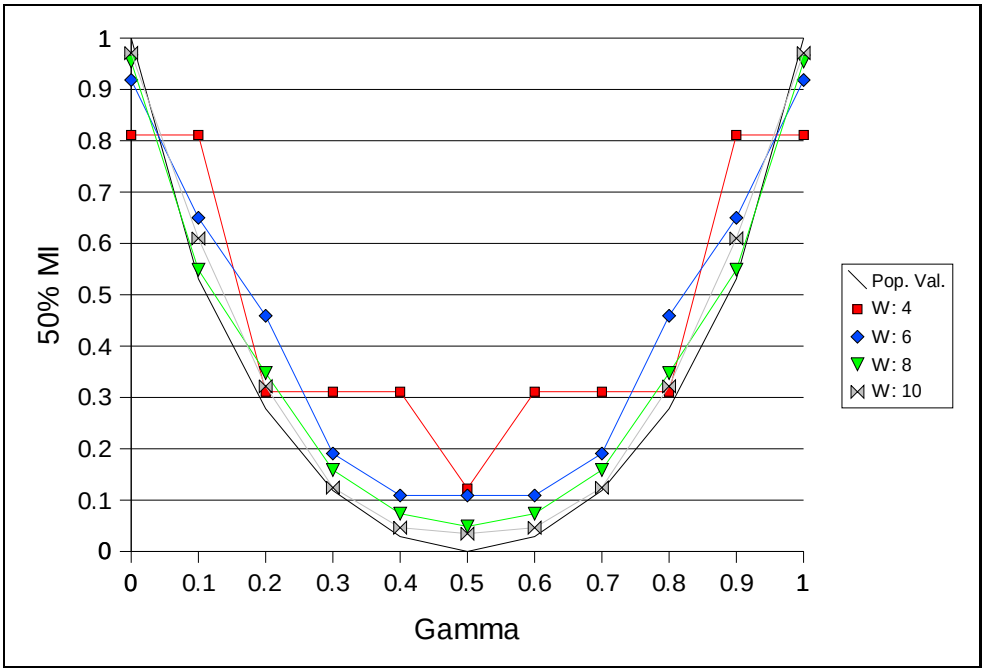

Figure 1: Graph of Many Mutual Information Values

tiple models and windows sizes by plotting the fiftieth-percentile values against the γ parameter for the various window sizes. This graph is symmetric around $x = 0.5$ as was expected, because the systems have the same symbol statistics on either side of $x = 0.5$, they just change which symbols are associated with which probabilities. As the window size increases, the fiftieth-percentile curves do seem to be approaching the population value, which was also expected. As the sample size increases, it is expected that measures of statistical information would tend to approach the population values. To highlight this trend, an RMS-type value was computed. Let the population mutual information for a particular system x be denoted by $M_{pop.}(x)$, and let the fiftieth-percentile sample mutual information for that same matrix with a window size of w be denoted by $M(w,x)$. The deviation $D(w)$ is computed just as the sample standard deviation, where the mean value is given by $M_{pop.}(x)$,

Window Size	Deviation	Best-Fit Value *D(w)*
4	0.21	0.21
6	0.11	0.11
8	0.05	0.06
10	0.04	0.04

Table 13: RMS Deviation from Population MI with Best-Fit Values

and the sample value is given by $M(w,x)$.

$$D(w) = \sqrt{\frac{1}{n-1}\sum_{x}\left[M(w,x) - M_{pop.}(x)\right]^2} \tag{119}$$

In this equation, the summation is taken over all basis matrices defined by x, and n is the number of basis matrices used. Table 13 lists the deviations calculated for the curves presented in Figure 1. Figure 2 shows a graph of the deviations as well as the best fit curve. The best-fit curve for this data was computed using the `Axum 6` computer graphing package. It is

$$D = 1.628\left(w^{-1.369}\right) - 0.034 \tag{120}$$

The magnitude of the deviation decreases as the window size increases, which was expected. Unfortunately, because the curve is only based on four data points, not much information can be gleaned from it. It would be useful to know whether the deviation asymptotically approaches zero as the window size increases, or whether it goes to some other value. The former would seem to be the case as the constant offset of the curve is relatively close to zero, but it is not possible to tell. Also, it is unknown whether the relatively large difference between the best-fit value and the actual data for a window size of 8 is the

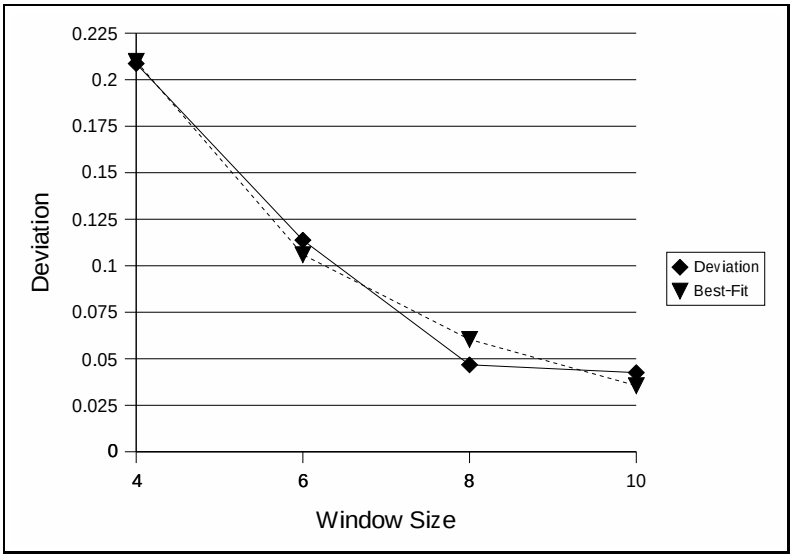

Figure 2: Graph of Deviation from Population

result of a poor fit or whether there is a change in the system properties at this point.

5 Conclusions and Further Work

Unfortunately not many conclusions may be drawn from this project. It was hoped that it would be possible to finish the transfer entropy modeling, and be able to provide a comparison between the performance of the mutual information and transfer entropy, but this was not accomplished. The results of the analysis of the mutual information do provide some useful information. Based on the trails run, a time-series length of only ten symbols provides a relatively close approximation of the population mutual information for the system, with an average deviation of around 0.04. This is useful for many situations because it allows one to expect good results from mutual information calculations even when working with limited data sets.

This project suggests a large number of possible ways to extend the research. The first would be to complete the mathematical analysis of the transfer entropy model, and evaluate its performance in measuring correlations. Also, no work was done in looking at how the coarseness of the data reflects on the performance of the measurement functions. It would also be interesting to see the relationship between Li's [8] work and the results returned by this project.

Further, one of the difficulties encountered in this project was the rapid increase in computation times for evaluating the mutual information as the window size increased. Because evaluating one mutual information value requires evaluating every possible combination of two binary strings of length w, increasing the length of the strings multiplies the number of combinations by four. That is, the total number of calculations shows exponential growth with respect to the sequence length, specifically $c = 4^w$, where c is the number of calculations and w is the window size. Because of this fantastic growth rate, the run time for the program quickly becomes enormous. Using a timer function, the run-time of the program was evaluated with various window sizes, as shown in Table 14.

In order to estimate the run time for the program for arbitrary window sizes, the parameter equation

$$y = \frac{k^x}{a} + b \tag{121}$$

was chosen as a model. In this equation, y is the run time, k is the exponent base, and x is the window size. a and b are free parameters determined by the program characteristics. If a were a regular multiplying constant, then from dimensional analysis a should be a Skinner constant with units of seconds per operation. However, it is easier to think of the

Window Size	**Num. Ops. (4^w)**	**Average Time**
1	4	0.01
2	16	0.02
3	64	0.03
4	256	0.07
5	1024	0.20
6	4096	1.07
7	16384	4.39
8	65536	17.83
9	262144	74.58
10	1048576	305.18

Table 14: Program Run Times given in seconds

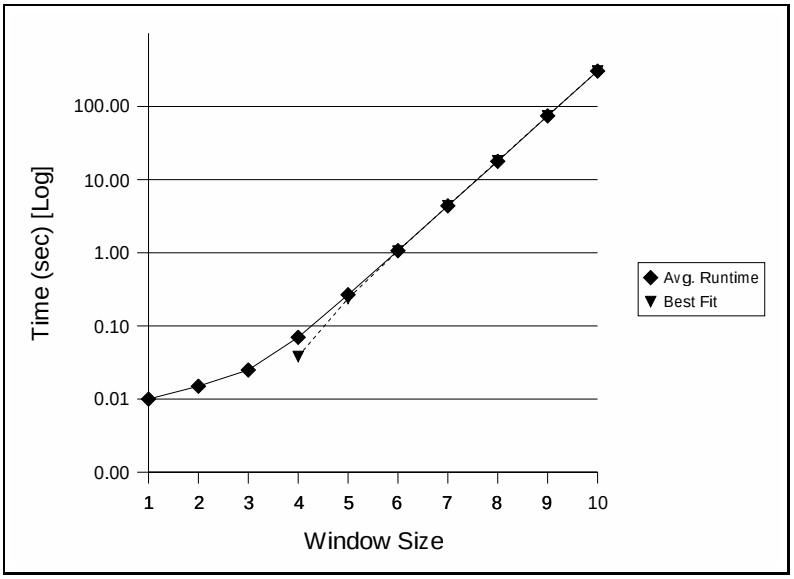

Figure 3: Graph of Program Run Times

inverse (operations per second), so a was chosen to be a dividing constant. b, with units of seconds, represents the fixed processing overhead that the program performs, such as opening and closing files. Using this model, a best-fit equation for the data was computed using `xmgr`. The final solution, with an accuracy of no less than 0.01 on any value, was as follows.

$$y = \frac{4.10^x}{4390} - 0.03 \tag{122}$$

Figure 3 shows a graph of Table 14 with this best-fit line included.

For a ten-bit-length sequence, the average run time for the program on one computer was 3.05×10^2 seconds, or a little over five minutes. Based on equation 122, the same program on the same machine processing a twenty-symbol strings would take about 4.09×10^8 seconds, or almost thirteen years. Obviously, the brute force methods to used in this project rapidly become untenable for "large" window sizes. However, there are several other approaches. For large window sizes ($w >= 100$), the small sample effects are much less

significant, and it is possible to use statistical sampling techniques. This is the method Roulston [11] used to evaluate the Shannon entropy values for series of length 100 and 1000. Work is still being done to determine the appropriate methods for evaluating information theoretic values for windows sizes greater than ten elements, but less than one hundred elements. Some of these methods include Monte Carlo-type sampling techniques, and being studied by Professor Dennis Weaver of Saint Leo University at the time of writing.

Finally, during the course of this experiment, software was developed to calculate mutual information and transfer entropy values for arbitrary datasets. It should be easy to convert that software to process real-time data, with the intention of being able to control a system in real time using the information theoretic measurements to detect changes in the system state.

In many ways, this project has served as test-bed for different ideas. While it has answered some questions, its greatest value should be to provide direction for further inquiries.

References

[1] G. Moore, “Cramming more components onto integrated circuits” Electronics **38** (1965).

[2] *Moore’s Law*, <http://www.intel.com/research/silicon/mooreslaw.htm>

[3] L. Gonick and W. Smith, *The Cartoon Guide to Statistics* (HarperCollins Publishers Inc., New York, NY, 1993).

[4] *Pick 4*, <http://www.state.nj.us/lottery/games/1-4_pick4.shtml>

[5] C. Shannon, “A Mathematical Theory of Communication” The Bell System Technical Journal **27**, 379 (1948).

[6] T. Schreiber, “Measuring Information Transfer” Physical Review Letters **85**, 461 (2000).

[7] B. Kedem, *Binary Time Series* (Marcel Dekker Inc., New York, NY, 1980).

[8] W. Li, “Mutual Information Functions versus Correlation Functions” Journal of Statistical Physics **60**, 823 (1990).

[9] H. Herzel and B. Große, “Measuring correlations in symbol sequences” Physica A **216**, 518 (1995).

[10] D. Benedetto, E. Caglioti, and V. Loreto, “Language Trees and Zipping” Physical Review Letters **88**, 048702 (2002).

[11] M. Roulston, “Estimating the errors on measured entropy and mutual information” Physica D **125**, 285 (1999).

Component	Name	Version
Processor	Intel® Pentium® 133 MHz	Family 5, Model 2, Stepping 12
Software Distribution	Red Hat	7.1
Operating System Kernel	GNU/Linux	2.4.2–2
Shell	GNU Bash	2.04.21(1)
C Library	glibc	2.2.2
C/C++ Compiler	GNU Compiler Collection	2.96-RH
Make	GNU Make	3.79.1

Table 15: System Requirements for `prob2`.

bintree.hh	exectimer.hh	mitest.hh	timeseries.hh	Makefile	basetest.hh	slog.hh
bintree.cc	exectimer.cc	mitest.cc	timeseries.cc	prob2.cc	problib.hh	

Table 16: Files Required to Build `prob2`.

A Appendix A: Computer Program Files

The following appendix contains the program source files required to compile the computer program `prob2`, Version 2, Block 2. This is the program that was used to evaluate the various statistical measures for this report.

A.1 Platform Requirements

All source computer code was written in the C++ computer programming language. A make file to control compilation was written to run with the GNU make program. Please consult Table 15 for a list of the hardware and software components that were used to compile and run the program.

A.2 Compilation and Operation Instructions

Please see Table 16 for a listing of the files needed to compile `prob2`. In order to compile the program, these file need to be placed into a common directory, and the Makefile invoked (this can usually be accomplished by typing `make` at the command prompt). Assuming a suitable C++ compiler is available, the files will be constructed, and the program `prob2` will be created.

At minimum to run `prob2`, you need to specify a source matrix file to read and process. There are also a number of switches that can be specified. Here is a listing of the switches, with their default values and functions.

- **-l**: Specifies the logarithm base to be used in calculations. This may be any real number, however *the program provides no error checking for nonsensical base values*, so

it is best to restrict this value to positive real values greater than zero. The default value is base 2.

- **-p**: Specifies the number of digits to display after the decimal point in calculations. This may be any integer, however *the program provides no error checking for non-sensical precision values*, so it is best to restrict this value to positive integers. The default value is 15 numbers after the decimal.

- **-t**: Specifies which calculation should be run on the input matrix. Current values associated with calculations are 1 for a χ^2 calculation, 2 for a mutual information calculation, and 3 for a transfer entropy calculation, however *only the mutual information calculation is implemented in* `prob2` *Version 2, Block 2*. If a value of 0 is specified, the program will probe the specified matrix file to determine for which calculate the matrix file is configured. If a particular calculation is specified, the matrix file will still be probed to determine if it is correctly configured, and the program will return an error if the matrix file number does not match the specified calculation number. Further, if an unsupported test is specified, the program will also generate an error. The default value is 0 for an autoprobe.

- **-w**: Specifies how many symbols should be used in evaluating the calculation, that is, the window size. If a window size of zero is selected, then the system will output the matrix specified by the matrix file, as well as the population calculation value for the matrix. Otherwise the program will evaluate the calculation for all possible binary, two-window combinations. Any unsigned long integer value is acceptable, but *no error checking is provided for negative values*. The default value is 0 for the system information.

Any of these switches may be specified in any order, provided all switches come before the filename. Any switches that come after the filename will be ignored. Duplicate switches are allowed, but only the last one will have any effect. Here are a few examples to illustrate various configurations.

- `$> prob2`
  This will fail because no matrix file name was specified.

- `$> prob2 matrix1.mat`
  This will run prob2 using the matrix1.mat file with the default values listed above.

- `$> prob2 -l4 -w 12 matrix1.mat`
  This will run prob2 using the matrix1.mat file with the logarithm base set to four and with the window size set to twelve (yes, a space is allowed between the number an the switch).

- `$> prob2 -l4 -w 12 matrix1.mat -l2`
  This will run prob2 using the matrix1.mat file with the logarithm base set to four and with the window size set to twelve. The second -l switch is ignored.

- `$> prob2 -l4 -w 12 -w6 matrix1.mat`
  This will run prob2 using the matrix1.mat file with the logarithm base set to four and with the window size set to six. The first window size switch is overridden by the second.

The remainder of this appendix will be a listing of the contents of the above files. While the program files will be slightly modified to accommodate printing requirements, the should be written an equivalently compilable form. *Please note that no testing has been done on the modified code contained herein, and while it should compile and run properly when entered into a computer system verbatim, no warranty or guarantee is made concerning the fitness or operability of this code.*

A.3 Source Code for `bintree.hh`

```
// Binary Tree Library File
// By Andrew Davis
// 2001-07-04

#ifndef BINTREE_HH
#define BINTREE_HH

//*******************************

#include <iostream.h>

//*******************************

struct treeptr {
  double xval, yval;
  struct treeptr *left, *right;
};

class bintree {
public:
  bintree() {root=NULL;} //Constructor method.

  void addelement (const double &newx, const double &newy);

  void writetree (ostream &outstr) const {if (root!=NULL)
inorderwritetree (outstr, root);}

  double cumdist () const;

  ~bintree() {if (root!=NULL) deleteelement (root);}
//Destructor method.

private:
  struct treeptr *root;
  struct treeptr *newelement (const double &newx,
const double &newy);
  void inorderwritetree (ostream &outstr,
struct treeptr *currpoint) const;
```

```
  void inordercumdist (struct treeptr *currpoint, double &xval,
double &sum) const;
  void deleteelement (struct treeptr *currpoint);
};

//*******************************

#endif
```

A.4 Source Code for `bintree.cc`

```
// Binary Tree Implimentation File
// By Andrew Davis
// 2001-07-04

#include "bintree.hh"

//*******************************

void bintree::addelement (const double &newx,
const double &newy){
  //This method adds an element to the binary tree.

  struct treeptr *currbranch=root;

  if (root==NULL){
    root = newelement(newx, newy);
    return;
  }

  for (;;){
    if (currbranch->xval == newx){
      currbranch->yval+=newy;
      return;
    }

    if (currbranch->xval < newx){
      if (currbranch->right==NULL){
        currbranch->right=newelement(newx, newy);
        return;
      }
      else currbranch=currbranch->right;
    }
    else{
      if (currbranch->left==NULL){
        currbranch->left=newelement(newx, newy);
        return;
      }
      else currbranch=currbranch->left;
    }
  }
}
```

```
//*******************************

void bintree::inorderwritetree (ostream &outstr,
struct treeptr *currpoint) const{
  if (currpoint->left!=NULL)
    inorderwritetree (outstr, currpoint->left);
  outstr<<currpoint->xval<<", "<<currpoint->yval<<'\n';
  if (currpoint->right!=NULL)
    inorderwritetree (outstr, currpoint->right);
}

//*******************************

double bintree::cumdist () const{
  double xval=0.0, sum=0.0;

  if (root==NULL) return (0);
  else inordercumdist (root, xval, sum);

  return (xval);
}

//*******************************

void bintree::inordercumdist (struct treeptr *currpoint,
double &xval, double &sum) const{
  if ((sum < 0.50) && (currpoint->left!=NULL))
    inordercumdist (currpoint->left, xval, sum);

  if (sum < 0.50){
    sum+=currpoint->yval;
    xval=currpoint->xval;
  }

  if ((sum < 0.50) && (currpoint->right!=NULL))
    inordercumdist (currpoint->right, xval, sum);
}

//*******************************

struct treeptr * bintree::newelement (const double &newx,
const double &newy){
  struct treeptr *newptr = new (struct treeptr);
  newptr->xval=newx;
  newptr->yval=newy;
  newptr->left=newptr->right=NULL;
  return (newptr);
}

//*******************************

void bintree::deleteelement (struct treeptr *currpoint){
  if (currpoint->left != NULL) deleteelement (currpoint->left);
  if (currpoint->right != NULL) deleteelement (currpoint->right);
```

```
  delete currpoint;
}

//*********************************
```

A.5 Source Code for `exectimer.hh`

```
// Program Timing Class Library
// By Andrew Davis, inspired by timing.c
// written by Robert Michael Lewis
// 2001-07-08

#ifndef EXECTIMER_HH
#define EXECTIMER_HH

//*********************************

class exectimer {
public:
  exectimer (bool printstate =1);

  void zerotimer () {time=get_prog_time();} // Set the timer.

  double gettime () {return (get_prog_time() - time);}
// Return the amount of time elapsed since timer object
// created or reset.

  void setprintstate (bool printstate =1)\
    {end_print = printstate;}
// This sets whether the destructor prints the time.
// 0=no print; 1=print.

  bool getprintstate () const {return (end_print);}
// Returns the value of the print state.
// See setprintstate() function.

  ~exectimer () {if (end_print)
    cerr<<"Time was: "<< gettime() << endl;}
// This is the destructor. If the print_state=1,
// it prints the elapsed time on the object at the
// time of destruction.

private:
  bool end_print;
  double time, sysspeed;
  double get_prog_time () const;
};

//*********************************

#endif
```

A.6 Source Code for `exectimer.cc`

```
// Program Timing Class Implementation
// By Andrew Davis, inspired by timing.c
// written by Robert Michael Lewis
// 2001-07-08

#include <iostream.h> // Cerr
#include <sys/times.h> // Times
#include <unistd.h> // sysconf (clock speed)
#include "exectimer.hh" // Library file

//*********************************************

exectimer::exectimer(bool printstate){
  //This is the constructor.

  sysspeed=sysconf(_SC_CLK_TCK); //Get the system speed
//in cycles per second. Must be called first in constructor
//as needed by zerotimer().
  end_print=printstate; //Set the print state.
//Defaults to 1=true=print in destructor.
  zerotimer(); //Initialize timer.
}

//*********************************************

double exectimer::get_prog_time () const{
// This is a private wrapper function for the times()
// function. For information on the times function and
// the tms struct, see "man 2 times".

  struct tms now; // Create tms struct.
  times(&now); // Call times to populate tms struct.
  return (((double) now.tms_utime +
    (double) now.tms_stime)/sysspeed);
// Return runtime. tms struct values in clock ticks,
// so need to divide by sysspeed to get seconds.
// sysspeed initialized to sysconf(_SC_CLK_TCK) by constructor,
// and is system clock tics per second. See "man 3 sysconf"
// for more information.
}

//*********************************************
```

A.7 Source Code for `mitest.hh`

```
// Mutual Information Test Class Library
// By Andrew Davis
// 2001-07-10

#ifndef MITEST_HH
#define MITEST_HH
```

```
//*********************************

#include "basetest.hh"
#include <fstream.h>

//*********************************

class mitest : public basetest {
public:
  mitest (ifstream &mdata);
  virtual void outputpopval (double logbase);
  virtual void initsampval (short p1,  w_size val);
  virtual r_size runpoptest (double logbase)
    {return (runtest (popprobmatrix, logbase));}
  virtual r_size runsamptest (double logbase) const;
  virtual r_size getsampprob () const;
  virtual void pokesampval (short *dataset1, short *dataset2,
    w_size index, short delta)
    {sampcntmatrix [dataset1[index]][dataset2[index]]+=delta;}
  virtual void outputsampval () const;

private:
  r_size runtest (r_size amatrix[2][2], double logbase) const;
  r_size popprobmatrix [2][2];
  w_size sampcntmatrix [2][2];
};

//*********************************************

#endif
```

A.8 Source Code for `mitest.cc`

```
// Mutual Information Test Class Implementation
// By Andrew Davis
// 2001-07-10

//*********************************************
// Include Files

#include <stdio.h>
#include "mitest.hh"
#include "slog.hh"

//*********************************************

mitest::mitest(ifstream &mdata){
  // This constructor loads population data from the file.

  r_size probA0, prob0, probA;

  mdata >> probA0 >> prob0 >> probA;
```

```
  popprobmatrix [symA][sym0]=probA0;
  popprobmatrix [symB][sym0]=prob0-probA0;
  popprobmatrix [symA][sym1]=probA-probA0;
  popprobmatrix [symB][sym1]=1-prob0-probA+probA0;
}

//***********************************************

void mitest::outputpopval (double logbase) {
  // This method displays the object information.

  printf("1 | %.4lf | %.4lf |\n"
 "--+--------+--------|\n"
 "0 | %.4lf | %.4lf |\n"
 "--+--------+--------|\n"
 "  |    A   |    B   |\n", popprobmatrix [symA][sym1],
popprobmatrix [symB][sym1], popprobmatrix [symA][sym0],
popprobmatrix [symB][sym0]);

  logbase++;

  fflush(NULL);
}

//***********************************************

void mitest::initsampval (short p1, w_size val){

  sampcntmatrix [symA][sym0]=0;
  sampcntmatrix [symA][sym1]=0;
  sampcntmatrix [symB][sym0]=0;
  sampcntmatrix [symB][sym1]=0;

  sampcntmatrix [p1][p1]=val;
}

//***********************************************

r_size mitest::runsamptest (double logbase) const{
  // This method returns the mutual information of cntmatrix.
  r_size sampprobmatrix[2][2];
  w_size totalcnt=sampcntmatrix[symA][sym0]
+sampcntmatrix[symA][sym1]+sampcntmatrix[symB][sym0]
+sampcntmatrix[symB][sym1];

  sampprobmatrix[symA][sym0]=((double)sampcntmatrix[symA][sym0]
/(double)totalcnt);
  sampprobmatrix[symA][sym1]=((double)sampcntmatrix[symA][sym1]
/(double)totalcnt);
  sampprobmatrix[symB][sym0]=((double)sampcntmatrix[symB][sym0]
/(double)totalcnt);
  sampprobmatrix[symB][sym1]=((double)sampcntmatrix[symB][sym1]
/(double)totalcnt);
```

```
  return (runtest(sampprobmatrix, logbase));
}

//*********************************************

r_size mitest::getsampprob () const{
  return(
    pow(popprobmatrix[symA][sym0],sampcntmatrix[symA][sym0])
 *pow(popprobmatrix[symA][sym1], sampcntmatrix[symA][sym1])
 *pow(popprobmatrix[symB][sym0], sampcntmatrix[symB][sym0])
 *pow(popprobmatrix[symB][sym1], sampcntmatrix[symB][sym1])
  );
}

//*********************************************

r_size mitest::runtest (r_size amatrix[2][2],
double logbase) const{
  // This method returns the mutual information of amatrix.
  r_size prob0, prob1, probA, probB, sum=0.0;

  prob0=amatrix[symA][sym0]+amatrix[symB][sym0];
  prob1=amatrix[symA][sym1]+amatrix[symB][sym1];
  probA=amatrix[symA][sym0]+amatrix[symA][sym1];
  probB=amatrix[symB][sym0]+amatrix[symB][sym1];

  if (amatrix[symA][sym0]!=0.0) sum+=amatrix[symA][sym0]
*slog(amatrix[symA][sym0] / (probA * prob0),logbase);

  if (amatrix[symA][sym1]!=0.0) sum+=amatrix[symA][sym1]
*slog(amatrix[symA][sym1] / (probA * prob1),logbase);

  if (amatrix[symB][sym0]!=0.0) sum+=amatrix[symB][sym0
]*slog(amatrix[symB][sym0] / (probB * prob0),logbase);

  if (amatrix[symB][sym1]!=0.0) sum+=amatrix[symB][sym1]
*slog(amatrix[symB][sym1] / (probB * prob1),logbase);

  return (sum);
}

//*********************************************

void mitest::outputsampval () const{
  // This method displays the object information.

  printf("\n"
 "1 | %ld | %ld |\n"
 "--+--------+--------|\n"
 "0 | %ld | %ld |\n"
 "--+--------+--------|\n"
 "  |    A   |    B   |\n\n", sampcntmatrix [symA][sym1],
sampcntmatrix [symB][sym1], sampcntmatrix [symA][sym0],
```

```
sampcntmatrix [symB][sym0]);

  fflush(NULL);
}

//*********************************************
```

A.9 Source Code for `timeseries.hh`

```
// Time Series Class Library
// By Andrew Davis
// 2001-07-04

#ifndef TIMESERIES_HH
#define TIMESERIES_HH

//*********************************

#include "problib.hh"

//*********************************

class timeseries {
public:
  timeseries(w_size windowsize);
  void resetdata (short val =0);
  void setval (w_size offset, short val =1);
  void pokeval (w_size offset, short val);
  short int checkval (w_size offset) const
    {return (series[offset]);}
  short int *getdata () {return (series);}
  w_size getnumzeros () const {return (numzeros);}
  w_size getsize () const {return (seriessize);}
  ~timeseries() {delete [] series;}

private:
  short *series;
  w_size numzeros, seriessize;
};

//*********************************

#endif
```

A.10 Source Code for `timeseries.cc`

```
// Time Series Implimentation File
// Andrew Davis
// 2001-07-04

#include "timeseries.hh"

//*********************************************
```

```
timeseries::timeseries (w_size windowsize){
  // This constructor initializes timeseries with a
// length and sets all values to zeros.

  seriessize=windowsize;
  series=new short int [seriessize];
  resetdata ();
}

//***********************************************

void timeseries::resetdata (short val){
  // This method sets all entries to one value
  for (w_size c=0; c < seriessize; c++) series[c]=val;
  if (val==0) numzeros=seriessize;
  else numzeros=0;
}

//***********************************************

void timeseries::setval (w_size offset, short val){
  //This function sets a particular element of the timeseries.

  if (series[offset] == 0) numzeros--;
  if (val==0) numzeros++;
  series[offset]=val;
}

//***********************************************

void timeseries::pokeval (w_size offset, short val){
  //This function sets a particular element of the timeseries.

  if (series[offset] == 0) numzeros--;
  series[offset]+=val;
  if (series[offset] == 0) numzeros++;
}

//***********************************************
```

A.11 Source Code for `problib.hh`

```
// Problib library file
// Andrew Davis
// 2001-07-12

#ifndef PROBLIB_HH
#define PROBLIB_HH

//***********************************************
// Typedefs
```

```
typedef double r_size;
typedef unsigned long w_size;

//***********************************************
// Macros

#define symA 0
#define symB 1
#define sym0 0
#define sym1 1

#define CHOICE 0
#define X2TEST 1
#define MITEST 2
#define TETEST 3

#define AOK 0
#define ARG_ERROR -1
#define FILE_ERROR -2
#define TEST_ERROR -3

//***********************************************

#endif
```

A.12 Source Code for `prob2.cc`

```
// Probability Program 2 using C++
// By Andrew Davis
// 2001-06-05

//***********************************************

#include "bintree.hh"
#include "timeseries.hh"
#include "mitest.hh"
#include "exectimer.hh"

#include <string>
#include <unistd.h>
#include <sstream>

//***********************************************

void readcmdparams (int &error, int argc, char *argv[],
ifstream &datain, ofstream &dataout, ofstream &cdout,
w_size &windowsize, double &logbase, int &precision,
int &testval){

  int optval, testchk;

  if (argc<2) error=ARG_ERROR;
  else{
```

```
    do{
      optval=getopt(argc, argv, "+l:p:t:w:");

      switch (optval){
      case 'l': logbase=atof(optarg); break;
      case 'p': precision=atoi(optarg); break;
      case 't': testval=atoi(optarg); break;
      case 'w': windowsize=atol(optarg); break;
      }
    }
    while ((optval>0) && (optval!='?'));

    datain.open(argv[optind]);
    datain>>testchk;
    if (testval==CHOICE) testval=testchk;

    if (testval!=testchk) error=FILE_ERROR;
    else if (windowsize>0){
      ostringstream buffdata, buffcd;
      buffdata<<argv[optind]<<"."<<windowsize<<".data";
      buffcd<<argv[optind]<<"."<<windowsize<<".cd";

      string sdata (buffdata.str()), scd (buffcd.str());
      dataout.open(sdata.c_str());
      cdout.open(scd.c_str());
    }
    else{
      ostringstream buffdata;
      buffdata<<argv[optind]<<".pop.data";

      string sdata (buffdata.str());
      dataout.open(sdata.c_str());
    }
  }
}

//*********************************************
void generateData (basetest *thetest, bintree &datastore,
w_size windowsize, double logbase){
  short carry;
  w_size c, d;
  timeseries sensor1 (windowsize), sensor2 (windowsize);

  // Initialize timeseries and set sample value.
  thetest->initsampval (0, windowsize);

  do{
    // store data in tree
    for (d=0; d<windowsize; d++) cout<<sensor1.checkval(d);
    cout<<" ";
    for (d=0; d<windowsize; d++) cout<<sensor2.checkval(d);
    cout<<"\n";

    datastore.addelement(thetest->runsamptest(logbase),
```

```
      thetest->getsampprob());

    // Increment sensor 1 and update tables.
    for (c=0, carry=1; ((carry!=0) &&
        (c < sensor1.getsize())); c++){
      thetest->pokesampval(sensor1.getdata(),
        sensor2.getdata(), c, -1);
      sensor1.pokeval(c, carry);

      if (sensor1.checkval(c)>1){
  carry=1;
        sensor1.pokeval(c,-2);
      }
      else carry=0;

      thetest->pokesampval(sensor1.getdata(),
        sensor2.getdata(), c, 1);
    }

    // Increment sensor 2 and update tables.
    for (c=0; ((carry!=0) && (c < sensor2.getsize())); c++){
      thetest->pokesampval(sensor1.getdata(),
        sensor2.getdata(), c, -1);
      sensor2.pokeval(c, carry);

      if (sensor2.checkval(c)>1){
        carry=1;
        sensor2.pokeval(c,-2);
      }
      else carry=0;

      thetest->pokesampval(sensor1.getdata(),
        sensor2.getdata(), c, 1);
    }
  }
  while ((sensor1.getnumzeros()+sensor2.getnumzeros())
!=(windowsize*2));
}

//***********************************************

void printerror (int error){
  switch (error){
  case ARG_ERROR:  cerr<<"No test file listed."<<endl; break;
  case FILE_ERROR: cerr<<"Testval--file mismatch."<<endl;
    break;
  case TEST_ERROR: cerr<<"Illegitamate test selected."
    <<endl; break;
  default: cerr<<"Unknown Error."<<endl; break;
  }
}

//***********************************************
```

```
int main (int argc, char *argv[]){
  // Main function.

  // Set program timer.
  exectimer clock1;

  // Initialize Local Data Objects
  basetest *thetest=NULL;
  bintree datastore;
  w_size windowsize=0;
  double logbase=2;
  ifstream datain;
  ofstream dataout, cdout;
  int testval=AOK, precision=15, error=0;

  readcmdparams (error, argc, argv, datain, dataout, cdout,
    windowsize, logbase, precision, testval);

  if (error==AOK){
    switch (testval){
    case MITEST: thetest=new mitest(datain); break;
    default: error=TEST_ERROR; break;
    }
  }
  datain.close();

  // If window size is zero, show system diagnostic data,
  // else generate data.
  if (error != AOK){
    clock1.setprintstate(0);
    printerror (error);
  }
  else if (windowsize == 0){
    clock1.setprintstate(0);
    thetest->outputpopval(logbase);
    cout<<"Pop. Test Val.: "<<thetest->runpoptest(logbase)<<"\n";
    dataout.precision(precision);
    dataout<<thetest->runpoptest(logbase)<<"\n";
  }
  else {
    generateData (thetest, datastore, windowsize, logbase);

    dataout.precision(precision);
    datastore.writetree (dataout);

    cdout.precision(precision);
    cdout<<datastore.cumdist()<<"\n";
    cdout.close();
  }
  dataout.close();

  return (error);
}
```

A.13 Source Code for `slog.hh`

```
// Arbitrary Base Log Function
// Andrew Davis
// 2001-07-10

#ifndef SLOG_HH
#define SLOG_HH

//*********************************

#include <math.h>

//*********************************

inline double slog (double val, double base){
  return  (log(val) / log(base));
}

//*********************************

#endif
```

A.14 Source Code for `basetest.hh`

```
// Base Test Class Library
// By Andrew Davis
// 2001-07-04

#ifndef BASETEST_HH
#define BASETEST_HH

//*********************************

#include "problib.hh"

//*********************************

class basetest {
public:
  virtual void outputpopval (double logbase) =0;
  virtual void initsampval (short p1,  w_size val) =0;
  virtual r_size runpoptest (double logbase) =0;
  virtual r_size runsamptest (double logbase) const =0;
  virtual r_size getsampprob () const =0;
  virtual void pokesampval (short *dataset1, short *dataset2,
    w_size index, short delta) =0;
  virtual void outputsampval () const =0;
};

//*********************************

#endif
```

A.15 Source Code for `Makefile`

```
ERRS = -Wall
OPT1 = -O3 -s -march=i586
OPT2 = -fomit-frame-pointer -funroll-loops -fforce-addr
OPT = $(OPT1) $(OPT2)

#*********************************************************

btcc = bintree.cc
bto = bintree.o
bthh = bintree.hh

etcc = exectimer.cc
eto = exectimer.o
ethh = exectimer.hh

micc = mitest.cc
mio = mitest.o
mihh = mitest.hh basetest.hh slog.hh problib.hh

tscc = timeseries.cc
tso = timeseries.o
tshh = timeseries.hh

p2cc = prob2.cc
p2o = prob2.o
p2hh = problib.hh mitest.hh exectimer.hh bintree.hh

mkfl = Makefile

#*********************************************************

prob2 : $(bto) $(eto) $(mio) $(tso) $(p2o) $(mkfl)
$(CXX) $(ERRS) $(OPT) $(bto) $(eto) $(mio) $(tso) $(p2o) -o prob2

prob2.o : $(p2cc) $(p2hh) $(mkfl)
$(CXX) $(ERRS) $(OPT) -c $(p2cc)

bintree.o : $(btcc) $(bthh) $(mkfl)
$(CXX) $(ERRS) $(OPT) -c $(btcc)

exectimer.o : $(etcc) $(ethh) $(mkfl)
$(CXX) $(ERRS) $(OPT) -c $(etcc)

mitest.o : $(micc) $(mihh) $(mkfl)
$(CXX) $(ERRS) $(OPT) -c $(micc)

timeseries.o : $(tscc) $(tshh) $(mkfl)
$(CXX) $(ERRS) $(OPT) -c $(tscc)

#*********************************************************
```