DISCOVERY OF CAUSALITY AND ACAUSALITY FROM TEMPORAL SEQUENTIAL DATA

A Thesis

Submitted to the Faculty of Graduate Studies and Research

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in Computer Science

University of Regina

By

Kamran Karimi

Regina, Saskatchewan

July, 2005

Abstract

In this thesis, we present a solution to the problem of discovering rules from sequential data. As part of the solution, the Temporal Investigation Method for Enregistered Record Sequences (TIMERS) and its implementation, the TimeSleuth software, are introduced. TIMERS uses the passage of time between attribute observations as justification for judging the causality of a rule set. Given a sorted sequence of input data records, and assuming that the effects take time to manifest themselves, we merge the input records to bring potential causes and effects together in the same record. Three tests are performed using three different assumptions on the nature of the relationship: instantaneous, causal, or acausal. The temporal reversibility of a relationship in time is used to judge the relationship as potentially acausal, while reversibility is considered as evidence for judging the relationship as potentially causal. To visualise the attributes' influence on each other, the thesis introduces dependence diagrams, which are graphs that connect condition attributes to decision attributes. We performed a series of comparisons between TIMERS and other causality discoverers, and also experimented with both synthetic and real temporal data for the discovery of temporal rules. The results show an improvement in the quality of the rules discovered with TIMERS.

Acknowledgements

I would like to express my gratitude to my supervisor, Dr. Howard Hamilton, for all the help and guidance he has provided me with over the years. I appreciate his kindness, friendliness, compassion, and sense of humour, as much as his clever and thoughtful technical remarks. His high academic standards have helped me to greatly improve my research abilities.

Many thanks to my PhD committee members, Dr. Cory Butz, Dr. Andrei Volodin, and Dr. Yiyu Yao, for their helpful suggestions and constructive criticism. This work has benefited greatly because of them. Thanks to Dr. Brien Maguire, head of the Department of Computer Science, and Ms. Donalda Kozlowski, the departmental secretary, who often helped me with paper work.

I also thank the Faculty of Graduate Studies and Research, the Department of Computer Science, NSERC, and Dr. Hamilton for providing me with the financial support needed to do this research.

Finally, thanks to my family and friends for their unconditional support. I am where I am now because of the help that I have received from so many people in my life. I am grateful to all of them.

Post Defence Acknowledgement

I am grateful for the thoughtful, fair, and objective criticism and comments of the external examiner, Dr. Oliver Schulte. His knowledge in multiple disciplines and his questions and comments during and after the defence has resulted in many improvements to the thesis. I also wish to thank Dr. Christopher Yost for presiding over the defence.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter provides an introduction to this thesis. Section 1.1 introduces some of the terms that are used in the thesis. Section 1.2 defines the problem that is being solved. Section 1.3 identifies the main contributions made by the thesis. To provide a general understanding of the solution, an overview of the approach is presented in Section 1.4. Finally, Section 1.5 provides an outline of the remainder of the thesis.

1.1 Basic Definitions

Suppose we have a set of attributes (variables) that describe a system. We may wish to know if the value of one attribute can be determined (predicted) by the values of the other attributes. The attribute whose value is being predicted is called the *decision attribute*, while the attributes whose values are used for prediction are called the *condition attributes*. A rule that relates certain values of the condition values to a value of the decision attribute is called a *decision* (or *classification*) rule. A decision rule takes the form of: if {<relation between condition attributes' values>} then (<decision attribute's

predicted value>), where the condition attributes appear on the left hand side of the rule, and the decision attribute appears on the right hand side. The placement of brackets is arbitrary and meant to increase the readability of the rules. An example rule is

if {($a > 2$)} then (b = true).

The data used to derive the rules are called *training data*. A rule can be *executed* when the conditions in the left hand side of the rule are satisfied. If so, the rule yields the decision attribute's predicted value.

To determine the quality of the decision rules, we measure their accuracy. After the rules are derived, we can try them on the same data to see whether or not the values predicted for the decision attribute match the data. The results are called the *training accuracy*. For example, if the rules correctly predict the decision attribute's value for training data 80% of the time, then the training accuracy is 80%. Alternatively, we can try the rules on data that were not used to derive the rules, and obtain the *predictive accuracy*. Predictive accuracy is usually lower than training accuracy because of the general tendency of rules to match the training data better than other datasets. This property is usually called *overfitting*. The data that are used to measure the predictive accuracy are called *testing* (or *unseen*) data. Both training and predictive accuracy can be used to measure the quality of the rules.

A set of rules designates a *relationship* between the values of the condition attributes on one hand, and the value of the decision attribute on the other hand. In this thesis we propose techniques for discovering the nature of this relationship.

1.2 Problem Statement

Given the sometimes-immense amounts of data gathered automatically by sensors in domains of scientific research and engineering, understanding relationships among the attributes can be a difficult undertaking. We may be interested in discovering properties of a system that is either inaccessible or very complex. Example systems include the weather, ocean currents, and environments on other planets. We may not have a theory for explaining the observations, or the theory may be hard to apply, as in the case of chaotic systems. In such cases, one can observe only some of the attributes, while the others remain hidden. It may not be possible to change some attributes, so experimentation may be limited or not possible.

This thesis proposes a method for answering the following question: Given an ordered sequence of data, how is the value of a certain attribute related to the values of other attributes? Is it potentially caused by them? Or are they merely observed together (perhaps at the same time, or perhaps with time delays in between)? Is the relationship reversible? In other words, given the next values, can we *retrodict* (predict in reverse) the current values?

More specifically, we consider the problem of discovering a relationship, in the form of a *set of rules*, from *sequential data records*. Each record contains the values of a set of attributes. The records come from the same source and are sorted according to a temporal (increasing/decreasing time) or linear (increasing/decreasing coordinate value) order. The aim is to investigate the relationships between the attributes of a system whose internal workings are not known. Such an investigation may be required when the system is a black box with hidden internals, or is very complex, or there is no access to the system,

for example, because it is too far from us. In such a case, a formal analysis of the system and the relationships implied by it may be hard or impossible.

It may also be hard or impossible to influence the system in any way. So we may not be able to change some input parameters of the system to observe the effects on the other attributes. If so, then we call the system *uncontrolled*. For the problem being solved here, we do not require that the values of certain attributes be fixed in order to see how the other attributes change. However, if the system is controllable, then the user may wish to perform experiments by influencing certain attributes in a regimented fashion.

If experimentation is possible then we can distinguish between input and output attributes, and we can change the input and observe the effects on the output. In such a case, if the values of the input attributes are determined independently of any output attribute, then we can restrict the set of attributes whose values are predicted (the decision attributes), to include only the output attributes.

An example problem with an uncontrolled system is that of predicting the weather. The data may consist of value for several attributes, recorded every minute, hour, or day. In this case, most if not all of the attributes are outside our control. Nonetheless, we may want to predict the value of one attribute using the others.

A common assumption in rule discovery is that the value of a decision attribute depends on the value of other attributes in the same record, often gathered at the same time. In this thesis, we assume that the value of the decision attribute may be better determined by referencing the values of condition attributes in the preceding or following records. In our weather example, the temperature now may depend more on the wind speed an hour ago than the wind speed now. With this assumption, the rules that are

created predict the value of a decision attribute at some time point, by referencing observations made before and/or after that time point.

Considering that a sequence of data records has a direction of progression, i.e., the order in which it has been sorted, the rules we derive from them can execute in forward and/or backward directions relative to the sorting order. Such rules are called *sequential* in general. In the weather example, the records are ordered temporally. If the data records signify temporal events, as in the values of the attributes of a system observed over time, then the resulting rules are *temporal*. Temporal rules imply potential causal or acausal relations.

As will be seen in the thesis, we define causality according to a common sense understanding of the term, but other definitions are possible. In the common sense definition, causes always precede the effects temporally. To simplify the analysis we assume that any time we see the causes we can expect to see the effects. Of course, sometimes we may see the effects without first observing the causes. This is a syntactic definition of causality, as we are only concerned with the form of a causal relationship, and not it meaning or semantics.

In this thesis, when we refer to a relationship as *causal* if the decision attribute's current value is determined by the value of at least one attribute observed in the past. This view is in line with many people's intuitive understanding of causality. When we refer to a relationship as *acausal*, then we consider it possible that the decision attribute's value is not caused by other attributes' values, but happens to be seen together, with some time interval between them. There may be hidden common causes that are producing the observed temporal pattern.

We represent a causal relationship with a causal ruleset, which is a set of rules where in each rule the previous values of the condition attributes are used to predict the current value of the decision attribute. Similarly, an acausal relationship is represented by an acausal ruleset, where previous values of condition attributes may be used to predict the value of the decision attribute, but, relative to the decision attributes' time of appearance, at least one condition attribute's value should come from the a succeeding time step. In both causal and acausal rule sets no rule references the current value of any condition attribute.

As an example, it may be possible to predict the wind speed at any hour by using the recorded value of the wind speed from the preceding or succeeding hour, but one value may not be causing the other. In this case, the wind speeds at two consecutive hours form a temporal pattern that could be the result of meteorological factors that have escaped our observation. So the wind speeds are not independent of each other, but they have no causal relationship.

Sequential data records can also take the form of a sequence of records collected along a linear path through space, for example, along a road or going down a well. We do not look for causality in such a non-temporal context because unlike the direction of the arrow of time, which appears to point in one direction, one can traverse a spatial line in either direction. Thus, there is no need to distinguish one direction as more significant [39]. In a linear spatial domain, assuming the system is not changing during the observation period, we could have observed the same data, but in reverse order, if we had started from the other end.

As an example, suppose that we are moving along a straight road, and observe the tar content of the asphalt every 2 metres. At the arbitrarily-named point 1, we measure the tar contents to be 70%, and at point 2 (two metres away), it is measured to be 65%. The result is the sequence of observations <…, 70%, 65%, …>. If we had started from the opposite end, we would have observed point 2 before point 1, and the sequence would contain the values in reverse: <…, 65%, 70%, …>. Observing events in reverse order is not possible in a temporal domain.

1.3 Contributions of the Thesis

This thesis makes five primary contributions, which we identify in this section. They are explained further in the text.

First, the thesis defines the problem of discovering sequential rules from sequential data, where each sequential rule refers to condition attributes that appear in a record other than that of the decision attribute. This rule discovery process can be performed in the context of a complex, closed, or perhaps otherwise inaccessible system. A sequential rule can be derived from sequential data records gathered from spatial (one-dimensional) or temporal data. If the data are of a temporal nature, then the results are *temporal rules*. Thus, according to our definitions, a temporal rule is a restricted type of a sequential rule.

Temporal rules are often created in a form that refers to previous or current attribute values in order to predict the decision attribute's value at a future time. The second contribution of this thesis is a generalisation of the direction of time, that allows a temporal rule to refer to condition attributes' values that are observed in the preceding or succeeding records relative to the record where the value of the decision attribute is

observed. Referring to the next records may not be suitable for real-time execution of temporal rules, because when such rules are executed they are not able to give a verdict concerning the current situation until sometime in the future. However, rules that refer to the succeeding values are applicable in cases where stored temporal data is available and rules with the highest possible training accuracy or predictive accuracy are desired. In such a case, for most records, the future and past observations are available during processing. Example applications are the repair of faulty observations, and the filling of missing observations. For linear spatial data, the same technique can be used by substituting notions of nearby previous and next locations (neighbourhood), for the past and future.

The third contribution is the introduction of a set of tests to judge the potential causality of a relationship that is expressed as a set of rules. It is often hard to ascertain that there is a causal relationship between two events, so our judgment in this regard should be considered as a hint. Knowing the nature of a relationship helps us to understand, and possibly better control, the phenomenon that is under investigation. Our tests can be applied to sequential temporal data.

Fourth, we show that the problem of temporal rule discovery is similar to that of determining whether or not a relationship is time-reversible. The reversibility of a system is of importance in fields such as automata theory, and especially in quantum computing. Reversible processes exhibit unique characteristics. When done slowly, they require very little energy consumption. They are also of essential value in quantum computing. Verifying the reversibility of the processes in a system is thus of importance from both theoretical and practical viewpoints.

The user may want to verify whether a process is reversible or not, but a formal analysis may be hard or impossible. We determine the reversibility of a relationship in simple sequences by attempting to predict a decision attribute's value using only the previous or only the next values of the condition attribute. By our definitions, a causal relation can be reinterpreted as being irreversible, while an acausal relation can be reinterpreted as being reversible. Our treatment of time, which allows us to reference both the preceding and the succeeding records in the same rule, is more general than that normally considered in the literature on reversibility of temporal relationships, where one is assumed to be able to reference either the past or the future, but not both.

The fifth contribution is the presentation of a graphical method, called the *dependence diagram*, to enable the user to view how the value of a decision attribute is determined by the condition attributes. It provides information concerning the attributes involved in the decision making process, and the degree of involvement. Decision rules usually concern only a single decision attribute. When given a number of input attributes, determining which one to choose as the decision attribute can be difficult. The user can try different candidates as the decision attribute, and may want to know which choice is best. A dependence diagram can help the user in this regard by making it easier to understand the decision rules.

1.4 Overview of the Approach

Part of the solution that we propose to the problems outlined in Section 1.4 is encapsulated in the Temporal Investigation Method for Enregistered Record Sequences (TIMERS). TIMERS begins by pre-processing the data to make them ready for use with

conventional rule or tree discovery methods. Then rules are generated, and in the case of temporal data, their quality (training or predictive accuracy) is used to form a judgement on the causal or acausal nature of the relationship that they express.

TIMERS relies on a rule/tree generator for functioning. To demonstrate that the algorithm works regardless of the underlying rule or tree generator, we perform experiments using C4.5 [66] for classification, and CART [10] for both classification and regression.

TIMERS allows the user to perform a number of tests, and based on the results, decide on the nature of the relationship. The conditions that TIMERS requires for the tests to be meaningful are that the data originate from the same source and be sorted. The user must ensure that these conditions are satisfied before proceeding with the tests.

To provide an informal introduction to TIMERS, we proceed in three stages. First, we describe the discovery of decision rules in general. Then we describe the discovery of temporal decision rules. Finally, we explain how an analysis of sets of temporal decision rules can form the basis for determining the acausality or causality of a relationship.

A common form of input in many data mining and machine learning problems is a *dataset* consisting of a series of records or instances. Each record contains the value of several attributes. An example dataset is shown in Table 1.1.

Table 1.1 Four records that contain values for Outlook, Temperature, and Play

Outlook	Temperature	Play
Sunny	25	Yes
Rainy	13	No
Overcast	20	Yes
Sunny	10	No

Table 1.1 contains records that show the values of the attributes Outlook, Temperature, and Play. Extracting information from such records has been an active and

well established area of research, referred to as concept learning, classification, or learning decision rules. The assumption is that there may be relations among the different attributes in each record. In a classification problem, the aim is to determine the value of the decision attribute, using the values of the condition attributes. For Table 1.1 we could set the decision attribute to be Play and try to determine when we can play, by using the values of Outlook and Temperature. The process of selecting a value for Play can be performed by creating a decision tree or a set of decision rules. A *decision tree* is a tree structure where at each node one or more attributes' values are tested, and based on the result, a branch is selected [56]. Following that branch, we may come to another node, and perform another test. Finally we will reach a leaf, and then a value for the decision attribute is determined. Figure 1.1 shows an example decision tree.

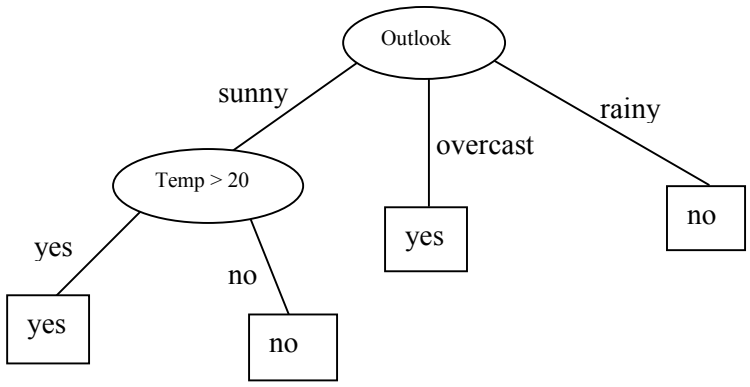


Figure 1.1 A decision tree for the records in Table 1.1

In a decision rule, values of certain attributes are tested at the left hand side, known as the *antecedent* of the rule. If all the conditions are met, then a value for the decision attribute is proposed by the right hand side, or *consequent*, of the rule. An example decision rule is:

if {(Outlook = sunny) and (Temperature > 20)} then (Play = yes). Rule 1.1

Decision trees and decision rules are closely related, and one can create one representation, given the other. C4.5, which is a widely referenced and commonly used classification program, constructs decision trees first, and then derives decision rules from them. Classical examples of data used for classification include the iris and soybean datasets [8], which have been used for classifying the decision attribute (type of the iris or sickness of a soybean plant).

Other related problems include discovering association rules, where the appearance of certain values together are discovered, but no classification is performed. When discovering an association rule, no single attribute is set as the decision attribute. A prominent representative of this problem domain is market basket analysis [11], which concerns the discovery of relations among different items sold together at a store. An example would be the observation that certain food items are usually bought together.

The usual assumption in datasets for classification and association rule mining is that their records are not related to each other in terms of: (1) the time of observation, and (2) the source or origin. For example, in the market basket analysis (introduced in Chapter 2), the records come from different customers (different sources) and with no particular order in time (no temporal ordering). The same applies for census data, where a number of attributes are filled with values pertaining to different people or things, possibly over a period of time.

These approaches do not require any constraints on the input dataset, and hence work in more cases than the temporal approach presented here, but as we will see, given the proper input, the temporal approach works better than the alternatives.

We take a classification approach, in that we investigate the relationship of a decision attribute with other condition attributes. The traditional approach is to look for a relationship between the decision attribute and other attributes within the same record. One example is Rule 1.1, as previously presented. This method may not produce good results if there is an *inter-record* relationship among the attributes, i.e., a relationship where the value of one attribute is related to a value in another record. For an inter-record relationship to be meaningful, we require that the records be produced from the same source.

Given the assumption that the records are ordered, we can investigate to see if the previous records affect the current value of the decision attribute. The past affecting the present follows the normal direction of time. This direction coincides with our everyday observations of causality. If using the past condition attributes results in better decision making, then there may be a causal relation at work. Another possibility is that we may be observing a temporal pattern, which is not necessarily causal. For example, two attribute's values may be related to each other over time (first, one is observed, and after a while the other one's value is observed), but neither is causing the other. They may both have a hidden common cause that has escaped our attention. So the observation that a temporally ordered relationship exists between some attributes by itself does not justify the conclusion that causality is present.

With our approach, temporal decision-making can be performed either with a decision tree or a set of decision rules. In either case the attributes should be qualified with their time of observation, because the same attribute can appear more that once in a rule, but at different times. A portion of a *temporal decision tree* [32] is shown in Figure

1.2, where yesterday's value of Outlook has been used to predict today's value of Outlook.

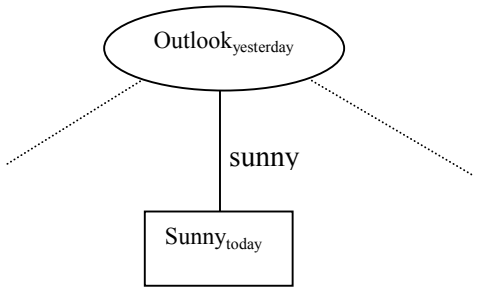


Figure 1.2 A portion of a temporal decision tree

An example temporal rule is shown in Rule 1.2 below.

if {(Outlook$_{yesterday}$ = sunny)} then (Outlook$_{today}$ = sunny). Rule 1.2.

The time of occurrence of the decision attribute is called the *current time*. Extracting such a rule requires an input dataset in which records are ordered temporally, and observations are made on a regular basis, such as once a day. This may not necessarily be the case in Table 1.1. Also, as can be seen in Rule 1.2, the decision attribute can participate in the rule as a condition attribute at times other than the current time. Rule 1.2 only covers a span of two days, hence "today" and "yesterday" are enough to qualify the attributes' time of observation. But if the rule involved more days, we use the current time as a reference. In this case we would obtain Rule 1.3 below.

if {(Outlook$_{today-1}$ = sunny)} then (Outlook$_{today}$ = sunny). Rule 1.3.

We say that a temporal relationship that is not causal is *acausal*. We also call it a *temporal association* because the value of the decision attribute and the condition attributes are associated together over time. This informal definition is different from the

mainstream definition of a temporal association rule, where the attributes all appear at the same time, but the association is valid only during a certain time interval [50]. Formal definitions of these concepts will be presented in Chapter 3.

To investigate the causality or acausality of a temporal relation, we consider the possibility that the decision attribute's value is determined by attributes not only in the previous records, but also in the next records, or both previous and next records. One example rule in this context would be Rule 1.4 below.

if $\{(\text{Outlook}_{\text{yesterday}} = \text{overcast})$ and $(\text{Outlook}_{\text{tomorrow}} = \text{rainy})\}$ then $(\text{Outlook}_{\text{today}} = \text{rainy})$. Rule1.4

Again, the attributes' time of observation could be set relative to the decision attribute's time (current time) as in Rule 1.5(a), and in a more general notation, in Rule 1.5 (b), where time T is assumed to always denote the time when the decision attribute's value is observed.

if $\{(\text{Outlook}_{\text{today-1}} = \text{overcast})$ and $(\text{Outlook}_{\text{today+1}} = \text{rainy})\}$ then $(\text{Outlook}_{\text{today}} = \text{rainy})$, Rule 1.5(a)

if $\{(\text{Outlook}_{T\text{-}1} = \text{overcast})$ and $(\text{Outlook}_{T+1} = \text{rainy})\}$ then $(\text{Outlook}_{T} = \text{rainy})$. Rule 1.5(b)

The unit of time progression depends on the rate at which observations were made to create the dataset. In Rules 1.4 and 1.5, the unit is a day, reflecting the assumption that the data was gathered daily. As an alternative, one can read "T+1" as "next" and "T-1" as previous, and use the same method for any "$T \pm n$" (read: n observations after or before).

To summarise this informal presentation, we assume three possibilities for a relationship. (1) A relationship is *non-temporal* if the values of condition attributes and the decision attribute are observed at the same time. In this case the relationship is best described as a traditional decision rule. Such a relationship is labelled *instantaneous* by our method. (2) A relationship is *causal* if only condition attributes from the past are used to predict the value of the decision attribute at the current time. This temporal requirement matches our intuitive understanding of causation. (3) A relationship is *acausal* if events from the future are used to predict the decision attribute at the current time. In such a relationship, the condition attributes's values can belong to the past or future, but at least one value belongs to the future.

To detect the nature of the relationship concerning a decision attribute, we perform three tests to see if the relationship is instantaneous, acausal, or causal. In the instantaneous case, we provide condition attributes from the same time as the decision attribute, and extract decision rules from the data. For the causality test, we include condition attributes from the past, and generate decision rules. For the acausal test, we include condition attributes from the past, as well as the future, and then generate rules to predict the decision attribute. The resulting rules' qualities (accuracy values) are then compared to each other, and the method that results in highest quality is chosen as an indication of the nature of the relationship.

The main drawback of TIMERS is the constraint that the input data must be produced sequentially by the same system. As we will explain in Chapter 6, under certain conditions our method can make mistakes in declaring a relationship to be causal or acausal.

1.5 Thesis Outline

The material in the thesis is presented according to the *practice and theory* principle. Namely, we have tried to introduced the concepts and methods informally first, and then present them formally later in the text. This approach was chosen in the hope of making the material easily understandable.

The remainder of the thesis is organised as follows. Chapter 2 overviews the problem of temporal and causal knowledge discovery, and presents some existing approaches that solve aspects of this problem. Section 2.1 overviews the main approaches for discovering temporal patterns and relations. Section 2.2 describes how the concept the causality has been approached from physical and statistical perspectives. Section 2.3 presents the main approaches to causal discovery in computer science and introduces two programmes for causal discovery that use different methods for causal discovery from TIMERS.

Chapter 3 presents the problem and the method in both formal and informal languages. In Section 3.1 we define what we mean by sequential data, temporal causality, and acausality. We also introduce temporalisation, which is the procedure we use to merge consecutive records together in different ways. This procedure allows us to bring the causes and the effects into the same record. As a result, we can use existing machine learning and data mining tools, which ordinarily do not search for relationships between records. Section 3.2 presents an algorithm for temporalisation, and Section 3.3 presents the TIMERS algorithm. Section 3.4 shows how by measuring the acausality or causality of a relation we are determining the reversibility or irreversibility of the system that generated the input data.

Chapter 4 describes the generation and presentation of temporal relations, which are a special form of sequential relations. Section 4.1 introduces TimeSleuth, the software that implements TIMERS. Section 4.2 introduces the dependence diagram, which is a visualisation method for better understanding the relationships among attributes in TIMERS' output. These diagrams are meant for human use. In Section 4.3 we show how temporal classification rules, converted to Prolog [80] statements, can be used for planning purposes.

Chapter 5 presents the results of experimenting with the TIMERS method. Section 5.1 compares TIMERS with other causal discovery software. TIMERS accepts a more limited type of input than other causality discoverers described in the thesis, but TIMERS can achieve higher accuracy than other method on that kind of input. In Section 5.2 we apply classification and regression to temporal and spatial data. We show that regression and classification give consistent results. The similarity between temporal data and sorted one-dimensional spatial data is explored in this section by experiments on spatial data obtained from a well-drilling dataset.

Chapter 6 discusses advantages and disadvantages of the proposed method and summarises the thesis. Section 6.1 describes when the introduced method can be applied. Section 6.2 details the strengths and limitations of TIMERS in the light of the lack of agreement on the concept of causality, and the consequent absence of an objective measure for determining causality. Section 6.3 summarises the findings of this thesis.

Chapter 2

Background Knowledge

In this chapter, we present some of the major tendencies and research areas in the subjects of discovering temporal and causal relations. Section 2.1 is concerned with approaches to discovering temporal rules, relations, or patterns without any claims as to any possible causality. In Section 2.2 we show that no clear consensus exists regarding the definition of causality. However, we see a general trend towards computable models of causality. Section 2.3 discusses the major approach in computer science to discovering causality, which is based on causal Bayesian networks. We introduce two different programmes, namely TETRAD and CaMML and demonstrate how they work by using examples. These programmes are used for comparison purposes in Section 5.

2.1 Temporal Discovery

The arrow of time is a unidirectional part of the spacetime continuum, as verified subjectively by the observation that we can remember the past, but not the future, or that we get older, but not younger. Objectively, there are a number of physical phenomena

that point to this special property of time. A prominent example is the entropy of any closed system, which cannot decrease [23].

Temporal data are often represented as a sequence, sorted in a temporal order. Examples of studies of sequential data and sequential rules are given in [3, 20, 69]. For example, in [20], the authors provide a genetic algorithm solution to the problem of detecting rules that manoeuvre an airplane that is being chased by a missile in a two dimensional space. Discrete attributes such as speed, direction of the missile, turning rate of the airplane, etc. are measured during 20 time steps. It is assumed that after 20 steps the missile will stop the chase. The rules discovered in that paper form part of a plan, and the genetic algorithm changes parts of the plan to make them better suited to solving the problem. The rules are then used in a simulator to measure their effectiveness. Time is obviously the sequencing factor in this example.

There are a number of general fields in the study of sequential data. A *time series* is a time-ordered sequence of observations taken over time [6, 12]. An example is the series of numbers <1, 3.5, 2, 1.7, …>. In a *univariate time series*, each observation consists of a value for a single attribute, while in a *multivariate time series*, each observation consists of values for several attributes. Most research on time series has assumed the presence of a distinguished attribute representing time, and numeric values for all other attributes. Attempts have been made to fit constant or time-varying mathematical functions to time series data [6]. A time series can be *regular* or *irregular*, where in a regular time series data are collected at predefined intervals. An irregular time series does not have this property, and data can arrive any time, with any temporal gap in between. A *deterministic time series* can be predicted exactly, while the future values in a *stochastic time series*

can only be determined probabilistically. The former is a characteristic of artificial and controlled systems, while the latter applies to most natural systems. Simple operations like determining the minimum or maximum values of certain attributes, finding trends (such as increases or decreases in the value of stocks), cyclic patterns (such as seasonal changes in the price of commodities), and forecasting are common applications of time series data.

Many approaches to the discovery of rules from time series data involve pre-processing the input by extracting features from the data. Global features include the average value, or the maximum value, while local features include an upward or downward change, or a local maximum value [30]. Another example of discovering temporal traits by pre-processing time series data is the discovery of increasing or decreasing trends before rule extraction [27]. While the study of time series is pursued widely, Keogh argues that the common method of using a window to extract information from a time series may not be useful or even meaningful, as similar results can be obtained from a randomly generated time series [51].

In [61], the phrase *multiple streams of data* is used to describe simultaneous observations of a set of attributes. The streams of data may come from different sensors of a robot, or the monitors in an intensive care united, for example. The values coming out of the streams are recorded at the same time, and form a time series. The data represented in Table 1.1 is an example of multiple streams of data, where three attributes are observed over time. In [54], an algorithm is presented that can find rules (called "structures" by the authors) relating the previous observations to the future observations.

Such temporal data appear in many application areas and a good overview can be found in [67].

An *event sequence* is a series of temporally ordered events, with either an ordinal time attribute (which gives the order but not a real-valued time) or no time attribute. Each *event* specifies the values for a set of attributes. A recurring pattern in an event sequence is called a *frequent episode* [53]. Recent research has emphasised finding frequent episodes with varying number of events between the key events that identify the event sequence. Algorithms such as *Dynamic Time Warping* and its variants measure the similarity of patterns that are stretched differently over time [43]. These methods have not been applied to searching for relations in causal data. No claim is made as to whether or not they represent causal relationships. The main difference between an event sequence and a time series is that a time series is a sequence of real numbers, while an event sequence can contain attributes with symbolic domains.

A market basket is traditionally defined as a set of Boolean attributes, each specifying if a particular item exists in the basket or not [10]. For example, given three products *A*, *B*, and *C*, an item can be <1, 1, 0> which means that the buyer picked products *A* and *B* but not *C*. In market basket research, the aim is to find associations (patterns) among the attributes of interest. Unlike the frequent episodes or the time series, in this case there is no interest in discovering temporal patterns, since it is assumed that the market basket items, which form an *itemset*, come from different sources, thus one cannot assume any meaningful order among the items. This assumption puts market basket research in a category of its own. Market basket data can be used for the discovery of frequent

episodes if they are generated by the same system. [26] contains an extensive review of the methods of discovering knowledge from sequential data.

Despite having different terminology, all the representations described so far in this section have the common characteristic of recording the values of some attributes and placing them together in a record. Time series, event sequences, and streams of data all attempt to find temporal rules (called patterns, episodes, and structures, respectively) from the input data. Each of these representations uses a different set of algorithms to solve its problems.

2.2 Causal Discovery

Modern physics is not based on intuitive ideas about time: There is no universal clock, and the only constant is the speed of light [22]. Each observer perceives time differently according to his speed relative to another observer. Ordinarily, physicists consider the speed of light to be an upper limit for the speed with which an event *A* can cause another event *B* [23]. There are theories about particles that can move faster than light. One example is the tachyon, which was first introduced by Feinberg [17].

In the macroscopic world, moving faster than light may lead to contradictions, as the effect *B* can appear before the cause *A*. This possibility is a contradiction to the intuitive understanding of causality, and leads to a paradox: if a person can move faster than light, then perhaps he can return to his past and change the causes of his travel in time. An effect happening before its cause is called *backward causation*. As shown in [79], some philosophers do not consider this impossible or paradoxical. It is possible that in the absence of free will, we could go back to our past and not change anything that interferes

with our time travel. An implication of this assumption is that since we are not able to change anything, we would travel to the past again and again, and thus be stuck in a never-ending time loop.

In this thesis we refer to apparent backward causation as *acausality*, and consider it to imply the presence of hidden common causes. The effects of such a common cause are spread over time, and so form a temporal association. This interpretation of the philosophical notion of backward causation acknowledges the temporal characteristics of the relationship, and avoids any temporal paradox.

A physical concept related to this thesis is that of time-reversibility of physical laws. Most laws in physics are expressed in a *time-reversible* manner, meaning that they can be applied in either temporal direction. An example is the relationship between force and acceleration, $f = m \times a$, which does not enforce any restrictions on the direction of time. While the previous formula might suggest that the force f is caused by the mass and the acceleration, we can re-arrange the formula to read $a = f / m$ and interpret the formula as saying that force causes acceleration. The fact that many physical laws do not exhibit temporal asymmetry has prompted some researchers to consider them as incomplete approximations [64]. Others assume that there are two distinct types of physical laws. *Time-symmetrical* laws hold backwards in time, while *asymmetrical* laws are valid in only one direction [4].

However, another possible view is that a formula such as $f = m \times a$ shows an instantaneous physical relationship: the force at any one moment is related to the acceleration at that same moment. In other words, they are created together and one cannot exist without the other. An example shows why the common perception is that

force causes acceleration: when driving a car, pressing on the gas pedal may be interpreted as exerting more force. The resulting acceleration in then considered an effect of the force. However, changing the position of the gas pedal only starts a process that changes the amount of fuel delivered to the engine, which then increases the force generated by the engine. The instantaneous relationship between the force and the acceleration remains, though the relationship between pressing down the gas pedal and observing a change in speed is temporal (causal).

A physical event that is asymmetrical in time is a mug falling from the table and breaking. This event is observed a lot more often than the reverse. This phenomenon is explained by the second law of thermodynamics, which states that the entropy of any closed system does not decrease. In physics, entropy by definition makes time one-directional, as we define the direction of time to be that of increasing entropy.

In classical Newtonian physics, causality is well-defined. As exemplified in a statement by Laplace, it was believed that if one knows the initial states of all the particles in the Universe, plus the applicable rules, one can predict the future or retrodict the past perfectly [59]. More specifically, given the current position value (x, y, z), the current momentum vector $\boldsymbol{p}$, plus the forces acting on a macroscopic particle, in classical physics one can plot the trajectory of the particle and predict the future position and momentum [59]. Quantum theory, however, has placed severe restrictions on causality in the classical physics sense. According to the Heisenberg uncertainty principle, one cannot know both the position and the momentum of a particle with arbitrary precision. More precisely, $\Delta x \times \Delta \boldsymbol{p} > 0$, where the Δ operator denotes the uncertainty in the value of its operand [59].

There is another example of how quantum physics is in disagreement with the intuitive understanding of causality. While in classical physics the interaction of a cause and its effect requires spatial proximity, quantum physics allows apparently instantaneous causal interactions between particles that are at arbitrary distances from each other. As an example, given a set of two entangled electrons, measuring the spin direction of one of them instantly determines the spin direction of the other one, thus establishing causal non-locality [52].

Granger causality, used mainly in economics contexts, puts clear emphasis on temporal precedence [19]. A *Granger causal relationship* exists when previous values of some attributes improve the prediction of a decision attribute's value. Suppose we are observing two attributes x_t and y_t over time, and A is a set of attributes considered relevant to y_t. We say x_t granger-causes y_t if there is a natural number $h > 0$ such that $P(y_{t+h} \mid A) < P(y_{t+h} \mid A \cup \{x_t, x_{t-1}, \ldots\})$, where $P(a \mid b)$ is the probability of event a happening, given that even b has happened.

For example, knowing the previous value of the attribute x_t = "Was there a political scandal today?" during the past few days may increase our ability to predict the value of the attribute y_t = "Does the stock market lose value today?". Granger causality is subject to errors. For example, if the event "person A leaves the building" is always followed 10 minutes later by "Everybody leaves the building" then Granger causality will consider person A's leaving the building as a cause for everybody leaving the building.

Statistics, which provides computational methods of judging causality, has become a popular method of causal investigation. Probabilistic causality can be defined as follows. A is considered to be a cause of B if we have $P(B \mid A) = 1$ and $P(B \mid \sim A) = 0$, where $P(B$

|A) measures the conditional probability of B given A, i.e., the probability of event B happening, given that event A has already happened [89]. This formula implies temporal precedence of the cause with regards to the effect. In practice, however, this definition is too brittle, and few real-world data would satisfy it, so one says that if P(B | A) > P(B | ~A) then A is a cause of B. An example of a weakness of this second definition comes into view if A = seeing lightning, and B = hearing thunder. Suppose we hear thunder only after seeing lightning, then we have P(B | A) > P(B | ~A), which would lead us to believe that the act of seeing the lightning cause the hearing of thunder. For this reason in the probabilistic approach there is an added assumption that there should not be any hidden common causes at work. In this example, the lighting and the thunder are created at the same time by a discharge of electricity.

Temporal order does not necessarily exist in statistics. In our description of the statement P(B | A), event A was considered to have already happened, but this consideration does not mean that every conditional probability implies a temporal ordering. Bayes' rule states P(B | A) = P(A | B) × P(B) / P(A). In the left hand side of the equation event B is assumed to have already happened, while in the right hand side event A is assumed to have happened. In other words, using algebraic manipulation we can change the order in which the events are supposed to have happened, and thus reverse the original, and possibly natural, temporal order of the events.

When a conditional probability value expresses a reverse temporal ordering, then it is called the *likelihood* [90]. Suppose we create a model M from some data D and then measure the conditional probability of the data given the model P(D | M). Since the data existed before the model, this probability is called the likelihood of D given M, and

indicates reversing the temporal order because now the model explains the data, instead of the other way around. For more discussions about discovering causality, especially from a statistical point of view, refer to [18, 28, 44, 78, 85].

2.3 Causality in Computer Science

Automatic discovery of causal relations among a number of attributes has been an active research field. More specifically, automated methods have been applied to determining whether or not the value of an attribute is caused by the values of the condition attributes.

The prevalent approach to the discovery of causality is to consider the problem to be that of creating a graph, where the parent nodes denote causes, while the children denote effects. Conditional independence plays a great role in the construction of these causal graphs. Two tools that were designed for performing unsupervised search for causal relations are TETRAD [70] and CaMML [42, 82]. They look for relationships among all attributes, resulting in a non-linear increase in running time as the number of attributes increase. Although these tools were not designed for the exact problem of finding atemporal and temporal rules from temporally ordered data from a single source, they can be applied to this problem and provide the most appropriate existing techniques for comparison purposes with our proposed approach.

Currently TETRAD is the de facto causality discoverer, and is widely discussed in the literature. CaMML, has received less attention, and fewer technical details are available for it.

TETRAD

TETRAD is a well-known causality discoverer that uses Bayesian networks [24, 62] to find causal relations. Bayesian networks are directed acyclic graphs that represent the conditional dependency of the attributes. A Bayesian network uses conditional probability distributions at each node [25]. It is advocated in [84] that a Bayesian network is a generalisation of a relational database. A *causal Bayesian network*, as used in TETRAD, assumes that the links in the graph denote causal relationships. TETRAD at first assumes that all attributes are causaly related, so at the beginning the causal network is fully connected. Then it uses conditional independence tests to remove or revise edges. The remaining edges form a *causal Bayesian network*. It has its own notation for displaying the discovered causal relations. For example, $A \rightarrow B$ means that A causes B and $A \leftrightarrow B$ means that both A and B have a hidden common cause. Unfortunately TETRAD's results are not always precise. A o$\rightarrow B$ means that either A causes B, or they both have a hidden common cause, A o–o B means that A causes B or B causes A, or they both have a hidden common cause (usually considered to mean the same thing as co-occurrence). This ambiguity about the exact nature of the relationship opens the door to different interpretations of the results. From the examples in [9, 76], it appears that TETRAD discovers causal relations that are subject to debate.

The main trend in causality mining involves using the statistical concept of conditional independence as a measure of the control one attribute may have over another [63]. For example, given the three attributes x, y, and z, if y is independent from z given x, that is, $P(y, z \mid x) = P(y \mid x)$, then we can conclude that y is not a direct cause of z. In other words, *x separates y* and z from each other. This basic concept is used to build Bayesian

networks, which show the conditional dependence of attributes as arcs. These arcs are then *interpreted* as signifying causal relations and the resulting Bayesian network is considered causal.

A joint probability is the probability of two or more events happening together, as in P(a, b). A marginal probability distribution is the probability of some events happening, while ignoring the other events. P(a), which ignores the attribute b, is a marginal probability value. In constructing a Bayesian network, we simplify joint probability distributions by using products of marginal probability distributions and conditional probabilities. As a simple example, P(a, b) can be written as P(b | a) × P(a). In this case the corresponding Bayesian network is a graph consisting of two nodes a and b, where a is the parent of b (node a points to node b), and there is a directed link from node a to node b. Bayesian networks are acyclic.

In general, the chain rule of probability is written as $P(x_1, \ldots, x_n) = P(x_n \mid x_{n-1},\ldots,x_1) \times P(x_{n-1} \mid x_{n-1},\ldots, x_1) \times \ldots \times P(x_2 \mid x_1) \times P(x_1)$, where × denotes multiplication. This formula can be simplified when we have information about the conditional independence of the attributes, which can be either given by the user, or computed from the data. For example, suppose we want to determine what causes the city's fire department to send out a fire rescue team to a home. We represent this event with r. A user can come up with a number of possibilities for the causes. The main cause can be identified as the home's alarm system a, which in turn may be caused by a fire f or by an accident t. But it also possible that the alarm system is not working, and the rescue team was sent because of a passer by's phone call c. A graph showing these relations comes is shown in Figure 2.1.

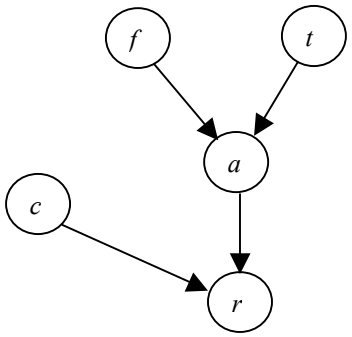


Figure 2.1 A Bayesian network for the fire rescue team example

Such graphs appear in Belief networks, probabilistic networks, knowledge maps, and causal networks [68]. This Bayesian network implies the property that any node is independent of its non-descendant nodes when conditioned on its parents. This implication is called the *causal Markov condition* [91]. Table 2.1 summarises the differences between a Bayesian network and a causal Bayesian network. As can be seen, the move from a Bayesian network to a causal Bayesian network mostly involves a change of interpretation and terminology.

Table 2.1 Differences between a Bayesian network and a causal Bayesian network

Bayesian network	Causal Bayesian network
An edge denotes a probabilistic dependence.	An edge denotes a direct causal relationship.
A node is independent of its non-descendants given its parents (Markov condition).	A node is independent of its non-descendants given its direct causes (Markov condition).
The network is used for probabilistic reasoning.	The network is used for causal inference.

Nodes *t*, *f*, and *p* do not have any parents, so their probabilities can be expressed by the marginal probabilities P(*t*), P(*f*), and P(*p*). Event *a*, on the other hand, has two parents, so its probability distribution is determined only by referring to the parents *t* and *f*. Using the chain rule of probability we have $P(f, t, a) = P(a \mid f, t) \times P(t \mid f) \times P(f)$. Now we

incorporate the conditional independence $P(t \mid f) = P(t)$ as expressed by the network, and we get: $P(f, t, a) = P(a \mid t, f) \times P(t) \times P(f)$.

For all attributes, according to the *chain rule of probability*, also known as the *product rule* or the *multiplication rule* [7], $P(f, t, c, a, r) = P(r \mid f, t, c, a) \times P(a, \mid f, t, c) \times P(c \mid f, t) \times P(t \mid f) \times P(f)$. The assumption of conditional independence of each node from other nodes, given the parents, allows us to make these substitutions: $P(t \mid f) = P(t)$, $P(c \mid f, t) = P(c)$, $P(a \mid f, t, p) = P(a \mid f, t)$, and $P(r \mid f, t, c, a) = P(r \mid c, a)$, so can simplify the joint probability formula: $P(f, t, c, a, r) = P(r \mid a, c) \times P(a \mid f, t) \times P(t) \times P(f) \times P(c)$.

Intervention is of great importance in this method of causal discovery. By fixing a node at a certain value, one investigates its effect on the descendent nodes. For example, we can manipulate the graph of Figure 2.1 by setting the alarm to be on, and hence create the new probability distribution: $P_{alarm}(f, t, c, a, r) = P(r \mid a, c) \times P(c) = P(c, a, r)$. As can be seen, an intervention removes the ancestors of the manipulated node. Pearl introduces the *do*() operator for manipulation of nodes. Fixing a descendant node should have no effect on its parents, which makes causality an asymmetric property by Pearl's definition because causality flows from parents to the children and not vice versa.

A graph denotes the order in which causality takes effect. In Pearl's scheme, the starting nodes take their values first, and then the descendants are affected. As an example, an equation such as $f = m \times a$ by itself does not show causality, but in a graph such as $f \rightarrow a$, we know that f comes first.

We associate a conditional probability table with each node of a Bayesian network. For this example we may have the following probability tables, as presented in Table 2.2. Accident t, Fire f, and Call c are not conditioned on any other attributes, while Alarm a

and Rescue *r* are represented by conditional probability distributions. As can be seen, the assumption of conditional independence reduces the size of the joint probability distribution from the potentially intractable 2^n, where *n* is the number of binary attributes. At the other extreme, where all attributes are independent, we would need a table of only size 2*n* to store the probabilities.

Table 2.2 The (conditional) probability tables for the nodes of a Bayesian graph

Accident		Fire		Alarm				Rescue Team			
True	False	True	False	Accident	Fire	True	False	Alarm	Call	True	False
0.2	0.8	0.01	0.99	false	false	0.01	0.99	false	false	0.01	0.99
Call				false	true	0.9	0.1	false	true	0.9	0.1
True	False			true	false	0.9	0.1	true	false	0.9	0.1
0.01	0.99			true	true	0.99	0.01	true	true	0.95	0.05

Each row in a conditional probability table should sum to one. A node with no parents has a table with a single row, which contains the prior probabilities of its values. By definition, the prior probability of an event does not depend on other events. Conditional probability tables can be either provided by a domain expert or computed from data. Here is a general procedure for constructing a Bayesian network [68]:

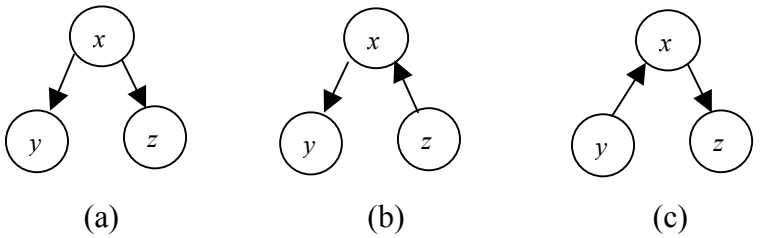
Procedure *P*2.1:

1. Choose a relevant set of attributes *X*.
2. Choose an ordering for the attributes.
3. While there are attributes left in *X*,
    a. Pick an attribute x_i and add a corresponding node to the network
    b. Set Parents(x_i) $\subseteq \{x_{i-1}, \ldots, x_1\}$ to some minimal set of nodes already in the net such that the conditional independence property $P(x_i \mid x_{i-1},\ldots,x_1) = P(x_i \mid \text{Parents}(x_i))$ is satisfied.
    c. Define the conditional probability table for x_i.

This procedure is simple and guarantees an acyclic graph, but the results depend on the order in which the nodes are added.

Bayesian networks can be used to consider different possibilities. For example, suppose that the rescue team is present. What is the probability that the *cause* was an alarm and not a call? A posterior probability is simply a conditional probability, such as P(A |B), where A has happened after (posterior to) B. We can use Bayes' rule to discover the posterior probabilities, as follows: P(a = true | r = true) = P(a =true $\wedge$ r = true) / P(r = true). A similar formula calculates P(c = true | r = true). Even though a and c are considered independent, they become dependent when conditioned on the observation of their effect r. For example, P(c = true | r = true $\wedge$ a = true) is different from that of P(c = true | r = true).

In a causal Bayesian network, the parents of a node are interpreted to be its causes, and the children of a node are considered the effects. To help distinguish between a mere association and causality, a causal Bayesian network measures the relationship of three attributes. One attribute acts as the control for the two other attributes. Suppose that we have three attributes x, y, and z. If the control attribute x separates the other attributes y and z, then y and z are independent given x. In this case x is either the parent of y and z, or x comes in between y and z, as in Figure 2.2. In all three cases y and z are not reachable given x. If both z and y are parents of x, then these two causes are not independent, as mentioned in the fire rescue team example. TETRAD generates a list of all the individual links of a causal graph, from which the user can derive the graph.



(a) (b) (c)

Figure 2.2 *y* and *z* are conditionally independent given *x*

The fourth possibility, that of nodes *y* and *z* pointing to node *x* (*y* and *z* are both causes of *x*), would create a *collider* at node *x*. Each of the three possibilities in Figure 2.2 has a corresponding factorisation of the probability joint distribution according to the Markov condition. For 2.2(a) we have p(*x*, *y*, *z*) = p(*x*) × p(*y* | *x*) × p(*z* | *x*). For 2.2(b), p(*x*, *y*, *z*) = p(*z*) × p(*x* | *z*) × p(*y* | x), while for 2.2(c) we have p(*x*, *y*, *z*) = p(*y*) × p(*x* | *y*) × p(*z* | *x*). The graph with the joint distribution that better matches the data is then selected to represent the data. TETRAD can leave the edge connecting two nodes as undirected when the data does not allow for a direction to be set. Alternatively, it may flag an edge as contradictory when there is some evidence supporting one direction, and some supporting the other direction.

Given a causal graph, the *d-connectedness* (dependence- or directional-connectedness) criterion can be used to determine if two nodes are independent from each other given a third node [63]. In this method independence and separation in the graph are considered to imply each other. In Figure 2.2, *y* and *z* are d-separated given *x*. One can generalise the d-separation (and d-connectedness) to sets of attributes by defining the sets of nodes *x* and *y* to be d-separated (d-connected) given *z*, if every member of *x* and y are d-separated (d-connected) given *z*.

The notion of conditional independence as defined in statistics is devoid of time. Pearl recommends that temporal information, if available, be used to place constraints on the relationships among the attributes (if we know *x* always happens before *y*, then *y* cannot be a cause of *x*) [63]. However, temporal order is not essential to the working of such algorithms.

TETRAD is proven to give correct results if the input data satisfies the Markov condition, as well as the *faithfulness* condition. A dataset generated from a set of causally sufficient attributes satisfies the faithfulness condition if the probability distribution entailed by it matches that of the graph that generated the data [92].

There has been work on introducing more temporal information in constructing Bayesian networks, as described by Shafer in [74], where the concept of a causally subsequent Bayesian network is explained (The variable *x* is subsequent to variable *y* when *y* precedes it *x*). Shafer argues that a probability tree, such as the one depicted in Figure 2.3 is more general than a Bayesian network. More details on Shafer's work come in [75].

Causal Bayesian networks are sensitive to the correctness of the input data. In other words, a causal Bayesian network is as reliable as the prior data that was used to generate it. Also, when presented with data that violate the original probabilities used to construct the network, the inferences may be wrong. However, the main practical problem with Bayesian networks is that constructing a network from observed data is an NP-Hard task [13], and becomes increasingly time-consuming, or impossible, as the number and combination of attributes increase.

Here is a simple example of how to generate a causal Bayesian network with only two variables. The first event e_1 is the probability of rain today, and the second event e_2 is the probability of rain tomorrow. We expect from experience that these two events be related to each other, meaning that if it rains today, it is more probable to rain tomorrow. Suppose we observe some weather data and construct the joint and marginal probability table shown in Table 2.3.

Table 2.3 The joint and marginal probabilities for the rain events

	Rain tomorrow (e_2)	No rain tomorrow ($!e_2$)	Marginal probability of raining today
Rain today (e_1)	0.15	0.10	0.25
No rain today ($!e_1$)	0.15	0.60	0.75
Marginal probability of raining tomorrow	0.30	0.7	

To generate a network representing the probabilities, we start with a node that denotes event e_1, and create its two child nodes, denoted as e_1 and $!e_1$. Then for each child node we create further children to denote e_2 and $!e_2$, as shown in the Figure 2.3.

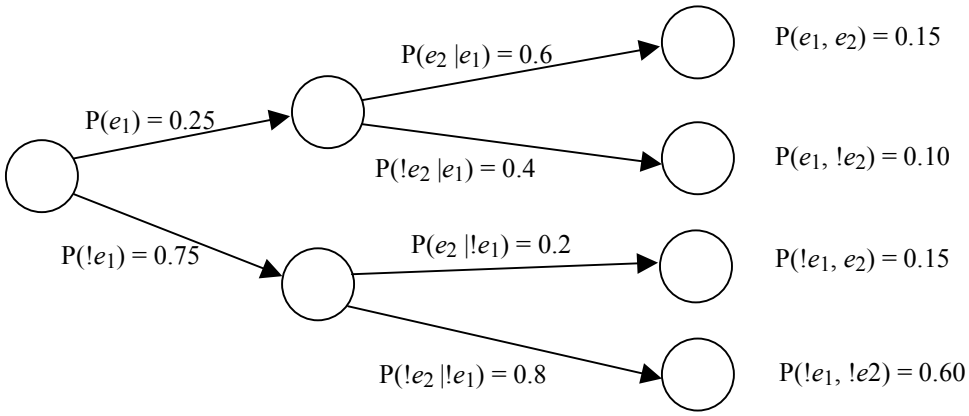


Figure 2.3 A probability tree representing two attributes

To create a causal Bayesian network in TETRAD we start from a fully connected graph, where every node is dependent on every other node. We then prune the nodes as we establish conditional independence among the nodes. After this phase TETARD

chooses a direction for each remaining link based on conditional dependence values obtained from both possibilities.

Here is how TETRAD handles the simple rain example. TETRAD starts with a graph containing both events connected by a link. If e_1 and e_2 are independent, it removes the link between them. It computes $P(e_1, e_2) = 0.15$, while $P(e_1) \times P(e_2) = 0.25 \times 0.3 = 0.075$, and so $P(e_1, e_2) \neq P(e_1) \times P(e_2)$, which by definition indicates that the events are not independent. S we leave the link in place. Both possibilities for the direction of the link are shown in Figure 2.4.

Figure 2.4 Two possible causal Bayesian networks for the rain example

Now the programme has to decide on a direction. We compute the conditional dependencies in both cases: $P(e_1 \mid e_2) = P(e_1, e_2) / P(e_2) = 0.15 / 0.30 = 0.5$, and in a similar fashion, $P(e_2 \mid e_1) = 0.6$. In this case it seems more plausible that e_1 should be the parent of e_2 than the other way around.

We needed to compute both possibilities in the case of a simple example with two attributes. In general, finding the best network is of exponential nature, and heuristics have to be devised to create trees in practical time limits. Procedure *P*2.1 is an example of such heuristic methods, but as we have already mentioned, it is sensitive to the order in which the variables are added.

CaMML

CaMML (Causal Minimum Message Length) is a Minimum Message Length based causal discovery system that creates a causal Bayesian network. It measures the goodness-of-fit of a causal model to the data [82]. Given a set of observed attributes, CaMML finds causal relationships between one or more causes and a single effect. As an example, in CaMML's notation (A, $B \rightarrow C$) means that A and B are causes of C.

CaMML tries to learn the best causal Bayesian structure to explain some observed data, using a Minimum Message Length (MML) metric for selecting a model. It searches the space of possible causal models using the Markov Chain Monte Carlo Method (MCMC) [55], and finds the one that best explains the data. In a Markov process, the transition from one state to the next depends on the current state only. In other words, any transition in a Markov process does not depends on the history of the moves that lead to the current state. A Markov chain is characterized by a transition matrix than gives the probability of moving from one state of the system to the next one. Starting from an initial state and multiplying by the Markov transition matrix enough number of times, we settle in a final state. Monte Carlo methods work by simulating an unknown function using probabilistic means. They sample values from a probability distribution and compute a function at those points. In MCMC, to obtain a specific probability distribution, one generates a Markov chain whose long-term equilibrium is that distribution.

MML was first introduced by Wallace and Boulton [81] and is based on maximizing the posterior probability of the model. If we want a good model M which is fitted to the

data D, then we want to maximize the posterior probability P(M | D). According to Bayes' rule we should maximize P(D | M) × P(M) / P(D). We recall that P(D | M) is the likelihood of D given M. Since D is given, we consider P(D) a constant, so the model M must be chosen such that the numerator is maximized. We follow the information theory's [77] method of taking the negative of the natural logarithm of the probability values, and convert the problem to that of minimizing the expression –ln(P(D | M)) – ln(P(M)). In information theory, this formula expresses the minimum length necessary to encode the model M, which is –ln(P(M)), plus the minimum length necessary for encoding any exceptions to the rules in the data, expressed by –ln(P(D | M)). Hence the name MML.

We now consider the simple rain example. The calculations performed may not be the same as those performed by CaMML, but the general outline is similar. For the rain example, we first must define rules that state the rainy days to be independent (resulting in two simple rules: e_1, e_2), or either event can cause the other one, expressed as either $e_1 \rightarrow e_2$ or $e_2 \rightarrow e_1$. We now attempt to select one rule set from the space of rule sets {<e_1, e_2>, <$e_1 \rightarrow e_2$>, <$e_2 \rightarrow e_1$>}. Since the size of the models are close, we ignore the model M in the equation and minimize –ln(P(D | M)).

In the first case, we have two rules, and consider their average probability value as the probability of the rule set. We calculate: P(D | e_1, e_2) = (P(e_1) + P(e_2)) / 2 = (0.25 + 0.3) / 2 = 0.225. From which we compute –ln(0.225) =1.49. Now we examine another rule set and compute P(D | $e_1 \rightarrow e_2$). To do so we add the correct predictions (P(e_2 | e_1) + P(!e_2 | ! e_1)) and subtract the probabilities of wrong predictions (P(e_2 | ! e_1) + P(!e_2 | e_1)). Note that this may result in values bigger than 1, but the formula will still work. In the

rain example we have: $P(e_2 \mid e_1) + P(!e_2 \mid !\ e_1) - P(e_2 \mid !\ e_1) + P(!e_2 \mid e_1) = 0.6 + 0.8 – 0.4 – 0.2 = 0.8$. We obtain $–\ln(0.8) = 0.22$. For $P(D \mid e_2 \rightarrow e1)$, we compute $P(e_1 \mid e_2) + P(!e_1 \mid !\ e_2) – P(e_1 \mid !\ e_2) – P(!e_2 \mid e_1) = 0.5 + 0.86 – 0.5 – 0.14 = 0.72$. We obtain $–\ln(0.72) = 0.32$. So MML also suggests that we should choose $e_1 \rightarrow e_2$.

The main difference between TETRAD and CaMML is that TETRAD aims to first discover the probabilistic dependency structure of the variables, and then infer a causal model to explain the dependencies. CaMML, on the other hand, considers a model first, and then sees how well the data can be fitted to it, as explained before.

There are some common characteristics for Bayesian learners such as CaMML and TETRAD. One is that they consider all the available attributes in the process of causal discovery. In other words, they try to find causal relationships among all the attributes, making the problem exponentially harder as the number of attributes grows. We have shown that this can result in very long execution times [38]. The other common consideration is that the input records are considered to be independent of each other, and no assumptions are made as to when or where they may have been obtained. The records could have come from different sources or different times. Assuming no temporal relationship among the records allows these approaches to work on many datasets.

Chapter 3

Knowledge Discovery from Sequential Data

In this chapter, we present our approach to the discovery of sequential rules, and show how we can use tools that were not designed to handle temporal data to process and generate temporal rules.

Section 3.1 presents the formal definition of causality and acasuality, and defines the problem we are solving. We also introduce the form of input data appropriate for investigation with TIMERS. This section explains how the data is pre-processed in different forms, how rules are generated and their quality measured. Sections 3.2 and 3.3 present the temporalisation and the TIMERS algorithms along with other sub-algorithms employed by them. In Section 3.4 we show how the problem of discovering rules reference attribute values in different time steps is related to that of traversing an automaton's graph of state transitions in forward or backward directions of time. We show that TIMERS can determine to what extend an automaton is reversible.

3.1 The Representation of the Problem

In this section we present the main ideas behind the TIMERS method, including a formal definition of what is meant by causal, acausal, and instantaneous relations.

3.1.1 Overview

Imagine a person is faced with the problem of learning the effects of his or her actions on a black box system. Given an unknown system with a set of knobs (inputs) and gauges (outputs), the person starts by observing the position of the knobs and the gauges. The knobs and gauges may be connected together, via feedback loops, which blurs a strict input-output notion. These observations would allow the person to deduce the relationships between the different input and output states. After a while, if possible he may start manipulating the knobs and reading the output, to see if it is possible to set the outputs to desired values.

As a more specific example, suppose a person can control the temperature of a cylinder, and can measure the temperature T, pressure P, and volume V of the cylinder. From the formula $P = nRT / V$, we know that at any instant, the three measurements are related to each other. However, there will be a time interval between turning the knob to increase the temperature, and the observation of the new values for pressure and volume. This dependence is captured in any sequential data that is made of the observables of the system, and one can say that turning the temperature knob causes the volume or pressure to change.

The above scenario makes explicit the importance of the temporal order among observations. As a side effect, we cannot mix data coming from different systems together, as that would invalidate any strict temporal order among the observations. In our method, any conclusion about the attributes' temporal effects on each other is derived directly from the data.

The problem we are considering is to classify the relation between a distinguished decision attribute and a number of condition attributes as one of *instantaneous*, *causal*, or *acausal* based on temporal data. To solve this problem we propose the TIMERS algorithm. TIMERS assumes the passage of time between the input records, and differs from the other methods in two main ways: First, it does not try to create a graph of causal relations, where all attributes take part in a hierarchy of causal relationships. Instead it focuses on the relationship between a decision attribute and the other attributes, to see if there is a causal relation among them. It is possible to run TIMERS several times with a different target (decision) attribute each time, but the results are not to be combined into a graph (though as will be seen, we can combine the results into "dependence diagrams"). Second, it assumes that the input records are temporally sorted and come from the same source. This temporal characteristic of the data is the basis for the justification of causal discovery in the presented method.

While TIMERS is fast and can handle many more attributes in the record than other methods [38], proper input is less widely available. However, when applicable, the results are meaningful, because with temporal decision rules the user can answer questions about "what" is related to what, as well as "how." For example, an attribute may appear in all causal rules that determine the value of a decision attribute, implying a stronger

relationship than another attribute that appears sporadically. This idea is the basis for dependence diagrams, as explained in Section 4.2.

Temporal sequences are often considered to be passive indicators for the presence of temporal structure in data [1, 21, 53, 61], but when causal relations exist in the domain, a temporal rule can be interpreted as a plan and executed. TIMERS' implementation optionally converts the output rules to PROLOG statements, which can then be executed directly [34, 35].

3.1.2 Problem Statement

We now state the problem formally. Sequential data are formed by a series of records that appear one after the other, and follow a definite order. In the case of observations made from a single system, the order may be temporal, in which case the records appear according to the time they were generated. We assume that a total of T observed records are present in the input data $\boldsymbol{D} = \{d_1, \ldots, d_T\}$. In other words, we gather data during T time steps. While we are using a set notation, we consider the order of the records in $\boldsymbol{D}$ to be unchangeable.

Since we are interested in the attribute values over time, we distinguish the time of observation of each attribute value in $\boldsymbol{D}$ by the first index. In practice this time stamping may be explicit or implicit. We do not rely on explicit time stamping, and consider the order in which the records appear to represent their time. For any t, $1 \leq t \leq T$ we distinguish between the current time step t and the other time steps. The value of the attributes at the current time step may depend on the attribute values that appear before or after the current time step. We assume that a time step takes long enough for any changes

in the system to have become manifest. The time steps do not have to be of equal length. If we consider the system to be an automaton, and each record to represent a state, then each record is generated after a state transition, which may take different amounts of time depending on the source and destination states.

Each record $d_t = <d_{t1},\ldots, d_{tm}>$ represents the values of m different attributes, taken from the set of attributes $V = \{v_1,\ldots, v_m\}$, which are observed at the same time step t. We consider one attribute v_j, $1 \le j \le m$ to be of special interest, and designate it as the decision attribute. In many contexts, the decision attribute is selected by the domain expert. Given the above data, the problem is to determine if the value of attribute v_j is causally related to the other attributes or not.

3.1.3 Temporalisation

We define the set

$$P = \{d_{ki} \mid 1 \le k \le T, 1 \le i \le m\}$$

to represent all observations made from time 1 to time T. For practical reasons, we concentrate on a limited *window size* of w observations. For any given time step t, the window includes observations in 1 time step, called the *current time step*, plus $w-1$ neighbouring time steps. The neighbouring time steps can appear before or after the current time step. We assume that only the information in this window is relevant to predicting the value of decision attribute v_j. The *window set*

$$P_w(t) = \{d_{ki} \mid t \le T - w + 1 \ \&\ t \le k < t + w, 1 \le i \le m\}$$

represents all observations in the window, starting at time step t and ending at $t + w - 1$. $P_w(t)$ merges a number of input records together.

When considering a window size of w, there are w possibilities for the position of the decision attribute in $P_w(t)$, ranging from position t to $t + w - 1$. We aim to reset the positions in the merged records, so the time-indices start from 1. This resetting is done by simply subtracting the value $(t-1)$ from each field's time step. The process of merging a number of consecutive records together, which is called *temporalisation*, converts every w consecutive observation in the original data to w temporalised records, such that

for $1 \le r < w$, $\boldsymbol{z_r} = \{z_{ki} \mid p \ne r, d_{pi} \in P_w(t)\ \&\ k = p - t + 1\ \&\ z_{ki} = d_{pi}\} \cup \{d_{rj}\}$.

j denotes the decision attribute's index. z_r is derived from $P_w(t)$ with the following modifications. First, the time steps are set to start at 1, no matter where in the input data $\boldsymbol{D}$ the fields come from. Second, we do not include the values for the decision attribute except in time step r. For every window set $P_w(t)$, temporalisation produces w temporalised records, one for each possible position of the decision attribute: $1 \le r \le w$. Each temporalised record contains $m \times (w - 1) + 1$ fields. The "1" indicates the decision attribute's value. The other $m-1$ values of the record in which the decision attribute appears are not included in the temporalised record. Since we generate w records for every record t, $1 \le t \le T-w$, the total number of records is $w \times (T - w + 1)$.

The above method is called the *sliding position* temporalisation because the position of the decision attribute slides from the beginning of the merged record to the end of the window.

3.1.4 Causality and Acausality in TIMERS

In the following formal definitions of instantaneous, causal, and acasual sets of rules, $R = \{r_1, \ldots, r_n\}$ is a set of rules generated to predict the value of a decision attribute. All

the rules have the same decision attribute, as returned by the DECISION() operator. Hence, DECISION(r_1) = DECISION(r_n) = da_{t0}. da is the name of the decision attribute from the set of attributes V, and $t0$ is the time it was observed, where $1 \leq t0 \leq w$. Since the appearance of an attribute v_i at different times usually denotes different values, the temporal relation will have a bigger domain: $V_t = \{v_{ji} \mid 1 \leq j \leq w \ \& \ 1 \leq i \leq m\}$,

where w, the window size, indicates how many time steps are involved. All the condition attributes that are used in a rule are contained in the set: CONDITIONS(r) = $\{a_t, \ldots\} \subseteq V_t - \{ da_{t0}\}$.

All attributes in CONDITIONS() have a time step as index, to distinguish the same attribute observed at different times. We assume that DECISION(r) $\notin$ CONDITIONS(r). In other words, an attribute at a given time cannot be used to predict its own value at that time. Here we are concerned with the properties of sets of rules, not attributes, and assume the rule sets not to be empty, that is, we assume that for any rule set R, $\exists\ r \in R$, such that CONDITIONS(r) $\neq \varnothing$. If this condition is not met, then we have a prediction for the decision attribute that does not depend on any condition attributes. We cannot discuss the temporal characteristics of this single rule with no left hand side, so we will not consider it any further.

Instantaneous. An *instantaneous* set of rules is one in which the current value of the decision attribute in every rule is determined only by the values of the condition attributes observed in the same record as the decision attribute. An instantaneous set of rules is an *atemporal* one. Another name for an instantaneous set of rules is a (atemporal) *co-occurrence*, where the values of the decision attribute are associated with the values of the condition attributes.

Definition 3.1: Instantaneous. For any rule r in rule set R, if the decision attribute d appears at time $t0$, then all condition attributes should also appear at time $t0$, i.e.,

R is instantaneous iff $(\forall r \in R$, if da_{t0} = DECISION(r), then $\forall a_t \in$ CONDITIONS(r), $t = t0$).

Temporal. A *temporal* set of rules is one that involves attributes from different time steps. A temporal set of rules can be causal or acausal. We exclude any condition attribute that appears at the same time as the decision attribute so as to prevent a strong instantaneous relationship from showing itself in the temporal tests' results.

Definition 3.2: Temporal. For any rule r in the rule set R, if the decision attribute appears at time $t0$, then all condition attributes should appear at time $t \neq t0$, i.e.,

R is temporal iff $(\forall r \in R$, if da_{t0} = DECISION(r), then $\forall a_t \in$ CONDITIONS(r), $t \neq t0$).

We now define the two possible types of a temporal rule:

Causal. In a *causal* set of rules, the current value of the decision attribute relies only on the previous values of the condition attributes in each rule [72].

Definition 3.3: Causal. For any rule r in the rule set R, if the decision attribute appears at time $t0$, then all condition attributes should appear at time $t < t0$, i.e.,

R is causal iff $(\forall r \in R$, if da_{t0} = DECISION(r), then $\forall a_t \in$ CONDITIONS(r), $t < t0$).

Acausal. In an *acausal* set of rules, the current value of the decision attribute relies on the future value of at least one condition attribute [46].

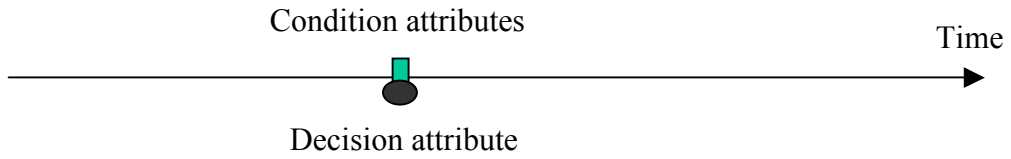
Definition 3.4: Acausal. For any rule r in the rule set R, if the decision attribute appears at time $t0$, then no attribute appears at time $t0$, and for at least one rule, at least one condition attribute should appear at time $t > t0$. i.e.,

R is acausal iff (1) $\forall r \in R$, if da_{t0} = DECISION(r), then $\forall a_t \in$ CONDITION(r), $t \neq t0$.

(2) $\exists r \in R$, if da_{t0} = DECISION(r), then $\exists a_t \in$ CONDITIONS(r), $t > t0$.

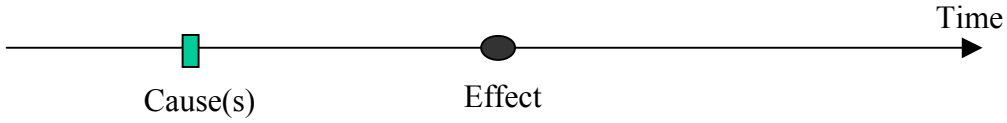
A pictorial representation of the instantaneous, causal, and acausal relations is provided in Figure 3.1. Condition attributes are represented by rectangles the decision attribute is shown by an oval.

In an **instantaneous** relationship, the decision attribute and the condition attributes are observed at the same time.



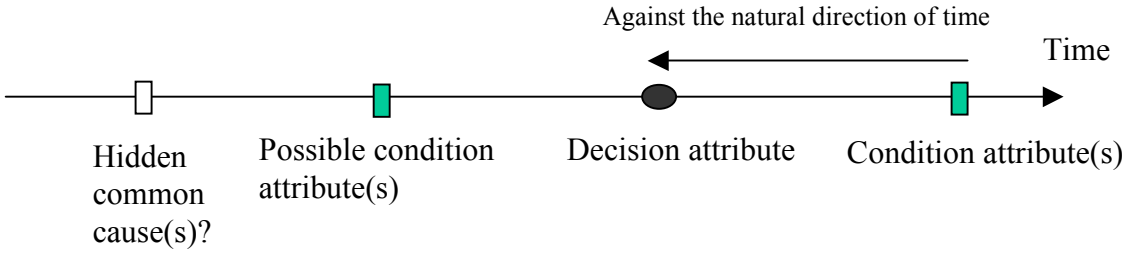
(a) Instantaneous relationship

In a **causal** relationship, the condition attributes occur before the decision attribute.



(b) Causal relationship

In an **acausal** relationship, at least one of the condition attributes occurs in the future.



(c) Acausal relationship

Figure 3.1 Temporal relationships between the attributes

We also put emphasis of temporal precedence between the causes and the effect, and use the ability to predict the value of the decision attribute to measure causality, which is similar to the case of Granger causality. However, Granger causality does not consider using future observations.

In a temporal context we look for rule sets that represent a relation R: $S \rightarrow \{v_{t0j}\}$. $S \subseteq V_t - \{v_{t0j}\}$, where v_{t0j} denotes the decision attribute at time $t0$. We consider S to be *minimal*, if there is no S' such that $S' \subset S$ and R: $S' \rightarrow \{v_{Tj}\}$. In other words, S represents the smallest set of attributes that can represent the relationship. If we cannot prove that set S is minimal, then a discovered temporal relation may actually be equivalent to an atemporal relation that results by eliminating "unnecessary" attributes. With this notation, in an atemporal relationship we have R: $S \rightarrow \{v_j\}$, $S \subseteq V - \{v_j\}$. In a temporal relationship we have: R: $S \rightarrow \{v_{Tj}\}$, $S \subseteq V_t - \{v_{t0j}\}$ & $S \cap (V_t - \{v_{t01}, \ldots, v_{t0m}\}) \neq \varnothing$.

The last condition makes sure that some attributes from time steps other than $t0$ are used.

TIMERS performs three tests: One for the instantaneous case, one for the causal case, and one for the acausal case. In each case we provide the rule discoverer with the appropriate condition attributes. In the instantaneous case, only attributes from the current time step are provided for rule generation. In the causal case, only attributes from the preceding time steps are provided, and in the acausal case, attributes from the preceding, as well as at least some attributes from succeeding time steps are involved in rule generation. The resulting rulesets are then evaluated, and the one with highest quality is considered to be the best description for the relationship. In this thesis we allow the user to choose either the training or the predictive accuracy values of the rules as the quality measure. It is possible to include other measures, such as the number of rules

needed for the rules (probably with lower number of rules being more desirable), in addition to or in place of accuracy. We leave the choice of quality measure to specific implementations.

When discovering acausality, the earlier versions of the TIMERS algorithm used the condition attributes only from the succeeding time steps [37], while in the current version TIMERS' rules can refer to attribute values in both previous and next records. This flexibility increases the likelihood of discovering better acausal rules, not only because more cases are investigated, but also because of the common sense notion that events happen gradually, and the neighbouring values in both directions are usually related to the value of the decision attribute.

We define an *order of conceptual simplicity* among the three types of the relations, with instantaneous being the simplest type of relationship, followed by acausality, followed by causality. Hence, instantaneous $<_{\text{simplicity}}$ acausal $<_{\text{simplicity}}$ causal. The intuition behind this ordering is that as we move from instantaneous to acausal and then to causal, more claims are being made about the relationship. As a principle we try to explain a relationship with the simplest possible type. As we will see, this ordering is used to choose a winning relations type when the results of the three tests are close.

3.1.5 Spatial Sequential Data

So far we have focused on discovering rules that involve time, i.e. the condition attributes appear at different times than the decision attribute. In this section, we take a step beyond time and generalise our method to include one-dimensional rules.

Modern physics has established time and space as a unity, where one is inconceivable without the other. However, time remains an anomaly because unlike the spatial dimensions, it seems that one cannot move back in time, although experiments have shown that at the particle level, this is in fact possible [23]. Discovering temporal associations that predict the future, based on past observations, is possible, and one can conceptually use the same idea for one-dimensional space as well. Our work has used the distinction between moving back and forth in time as the basis of distinguishing causality on one side, and acausality (or temporal co-occurrence) on the other side. We can explore the same idea in a one-dimensional space.

Consider the problem of drilling an oil well. The well can be regarded as a one-dimensional (linear) entity. As the drill is making its way through the ground, new points are explored and registered. When we stop, we have a series of records that follow each other along a line. While the data may seem to have been produced in a certain temporal order, one could argue that if the drilling were started from the opposite side, then we would be encountering the points in the reverse direction of time. It makes perfect sense to analyse the drilling data in either direction, with the results being valid in both cases. The linear order does not appear to offer any idea relevant to cause and effect, because what happens to precede something in one direction will be following it in the opposite direction. Thus, for spatial data, we refrain from labelling the type of relationship as causal or otherwise, and concern ourselves with finding the best possible rules for predicting the decision attribute's value. We still perform three tests, but the ruleset selected by the tests is considered simply as the best ruleset for predicting the decision

attribute. No judgement is suggested concerning the nature of the relationships between the attributes.

3.2 The Temporalisation Algorithm

Traditionally, to determine the value of a decision attribute we use the condition attributes from the same record. Here we provide a simple example to explain the reasons for temporalisation. Consider the following sequence of <position, direction> records: <3, Left>, <2, Left>, <1, Right>, <2, Right>, etc. Each record indicates the current position along a line, and the direction of movement at that position, assumed to be determined randomly. Every movement changes the position by one unit to the left or right. If the order of the records was ignored, the data would be atemporal. In such a case, we could form atemporal rules, for example by using the current movement direction (the condition attribute) to determine the current position (the decision attribute). The results would be instantaneous rules, but we can tell intuitively that they probably would not have good accuracy values, because there is no inherent relationship between a position and the randomly-chosen direction of movement at that position.

To explore causality, we use the intuitive notion that the condition attributes' effects take time to appear, and thus are seen in the later records. In this example, the next position depends on the previous position, plus the previous direction of movement. Given a temporally sorted sequence of records, we merge subsequent records into one record, bringing the possible causes and the effect together by temporalisation. The window size in temporalisation expresses our belief about how long it takes for effects to manifest themselves. Temporalisation enables us to use standard rule learning tools and

programmes, which do not consider the passage of time, for the purpose of temporal analysis of data.

An example record sequence with a window size of two in the forward direction of time would be <3, Left, 2, Left>, <2, Left, 1, Right>, <1, Right, 2, Right>, etc., where each record includes data from two time steps. Here we have the previous position, and the movement direction, as well as the current values. Obviously, given the previous position and the previous direction of movement, it is easy to determine the current position.

After pairs of consecutive records have been merged, each temporalised record may contain a cause and its effect. If the rules derived from these temporalised records result in a better accuracy value than rules derived from the original records, then we declare the relationship between the current position and other attributes as causal. In this example, we can expect very good results because, assuming a deterministic world where actions do not fail, when we know the past position and the past movement direction, we can identify the next position with certainty.

However, we may be dealing with a temporal relation that is not causal. For this reason, we should also consider the possibility that the next position and the next direction of movement allow reliable prediction of the current position. A temporalised record would now look like: <2, Left, 3, Left>. Of course, in this particular example, this acausal hypothesis is not as good as the causal one, as knowing where we are in the future is not sufficient to predict where we are now. There are two possibilities: currently we can be to the left or the right of the future position. This example shows clear signs of causality. As previously mentioned, at the current time we leave out all attributes with the

exception of the decision attribute. In the causal temporalisation, for example, the first record would thus be <3, Left, 2>. This is to prevent a strong instantaneous relationship from skewing the results of the causal and acausal tests.

The temporalisation technique prepares the data for rule extraction, and the final judgment about the type of the relationship is based on the quality of the rules. The quality can be measured using the rules' accuracy value. For the instantaneous test, no temporalisation is performed. Alternatively, one could say we temporalise with a window size of 1. For the causal (forward) test with a window size w, temporalisation involves merging every w consecutive records together, and setting the decision attribute to be that of the last record. For the acausality (backward) test, we attempt to predict values in the current record from values in the following records. So to use the same method as for the causal test, w consecutive records are merged and the decision attribute is set to be that of the first record (the current time).

With any fixed window size w, the sliding position temporalisation algorithm first places the current decision attribute at position one, and uses the next $w - 1$ records to predict its value. This corresponds to backward temporalisation in the previous versions of TIMERS [37]. Then the current attribute is set at position 2, and the previous record (position one) and the next $w - 2$ records are used for prediction. This case has no correspondence in the previous versions, as presented in [37]. This movement of the decision attribute's position continues until finally it is set to w, and the previous $w - 1$ records are used for prediction. This corresponds to forward temporalisation in the previous versions.

As an example, consider four temporally consecutive records, each with four fields: R_1: <1, 2, 4, true>, R_2: <2, 3, 5, true>, R_3: <6, 7, 8, false>, R_4: <5, 2, 3, true>. Suppose we are interested in predicting the value of the last (Boolean) attribute. Using a window size of 3, we can merge them as in Table 3.1. The decision attribute is indicated in boldface. When it comes to the record involving the decision attribute, we do not consider any condition attributes in the same record as the decision [37]. The *Record.value* notation means that we are only including the decision attribute. For example, <R_1, R_2, R_3.**false**> would contain <1, 2, 4, true, 2, 3, 5, true, **false**>, where *false* is the decision attribute in R_3. This omission makes sure that minimum amount of data is shared between the original (instantaneous) record and the temporalised record. In Table 3.1, there is a temporal order between the records in the first column, but there are no such relationships in other columns. The temporal order has been moved *inside* the temporalised records. The temporalised records can thus be used by conventional data mining tools that ignore any temporal order between the input records.

Table 3.1 Temporalisation with the forward, backward, and sliding position methods

Instantaneous. $w = 1$ (original data)	Forward (Causality). $w = 3$	Backward (Acausality). $w = 3$	Sliding position. $w = 3$
R_1 = <1, 2, 4, **true**>	<R_1, R_2, R_3.**false**>	<R_3, R_2, R_1.**true**>	<R_2, R_3, R_1.**true**>
R_2 = <2, 3, 5, **true**>	<R_2, R_3, R_4.**true**>	<R_4, R_3, R_2.**true**>	<R_1, R_3, R_2.**true**>
R_3 = <6, 7, 8, **false**>			<R_1, R_2, R_3.**false**>
R_4 = <5, 2, 3, **true**>			<R_3, R_4, R_2.**true**>
			<R_2, R_4, R_3.**false**>
			<R_2, R_3, R_4.**true**>

For the acausal test, we can have a mixture of previous and next attribute values. Given a window size w, p previous records and f future records can be involved, with the decision attribute happening in between. So we have $p + 1 + f = w$. The "1" in this equation indicates the location of the decision attribute at the current time. The requirement is that f be at least 1 (at least one record from the future for the acausality test

to be valid), so we have $1 \le f \le w-1$, and $0 \le p \le w-2$. The decision attribute's position slides in the merged records. It moves from being in the first position (no previous records) to being in record number $w-1$ ($w-2$ previous records, 1 next record). The sliding position temporalisation operator is presented in Figure 3.2.

The temporalisation operator *Temporalise*(w, pos, $\boldsymbol{D}$, d) takes as input a window size w, the position of the decision attribute within the window pos, the set of input records $\boldsymbol{D}$, and the decision attribute d, and outputs temporalised records. D_i returns the ith record in the input **D**. *Field*() returns a single field in a record, as specified by its first attribute. The += operator stands for concatenating the left hand side with the right hand side, with the results going to the left hand side attribute. <> denotes an empty record.

```
for (i = 0; i ≤ |D| − w; i++) {
        temporalisedRecord = <>
        for(j = 1; j < pos, j++)              // previous records
            temporalisedRecord += D_{i+j}
        for(j = pos + 1; j ≤ w, j++)          // next records
            temporalisedRecord += D_{i+j}
        temporalisedRecord += Field(d, D_{i+pos})     // the decision attribute
        output(temporalisedRecord)
}
```

Figure 3.2 The sliding position temporalisation method

This algorithm covers all three temporalisation methods: (1) For the instantaneous test, we provide it with a window size of 1 and a position of 1. Alternatively we could refrain from using the algorithm and simply employ the original input data. (2) For the causality test, window size w would be any desired value bigger than 1, and the position would be w too (last record). (3) For the acausality test, the window size could be set to any value bigger than 1, and the position would change between 1 and $w-1$.

Temporalisation results in a set of records with no temporal relationship among them. There is, however, an implicit temporal relationship within the fields in each

temporalised record. A traditional tree or rule generator considers all the attributes to be available at the same time, and the Temporalise() function allows such rule generators to still work, though the output of a traditional tree or rule generator does not consider the attributes to have been observed at different times.

We can remedy this shortcoming at either the rule or tree generation level. At the rule generation level the attributes in the output rules can be rearranged according to their time of appearance. We would thus re-temporalise the output rules. Another approach is to create a temporal decision tree instead of rules. Normal decision tree builders such a C4.5 rank the condition attributes according to how suitable they will be for expanding the tree at each step. For a temporal decision tree, the attributes should be ranked according to their temporal order as well as their suitability for expanding the tree. This is a stricter requirement than in the case of the rules, because unlike the rules, one cannot reorder the branches in a tree without altering the tree. To sort the condition attributes, they are partitioned according to their time of encounter. So attributes that are encountered at time step t of the temporalised records go into set V_t. If at a node of the tree a condition attribute from the set V_t is used, then the children of that node can only use condition attributes from the sets V_j ($j \geq t$), even if doing so makes the tree sub-optimal. An example temporal decision tree is shown in the right part of Figure 3.3.

(a) A normal decision tree (b) A temporal decision tree

Figure 3.3 Normal and temporal decision trees

To see the effects of the window size on trees generated by C4.5, we interpret the data in a Letter Recognition database [8] as being temporal. The data consists of 20,000 records to classify the letters of the English alphabet. There are 16 condition attributes that are actually seen all at the same time. However, we could look at the data as if different parts were generated at consecutive time steps. The results are shown in Table 3.2, where a window size of 1 results in normal tree generation behaviour because all attributes are assumed to have been produced at the same time.

Table 3.2 The effects of the window size on the size and accuracy of the tree

Window Size	Before Pruning		After Pruning	
	Size	Error	Size	Error
1	26721	0%	25713	0.5%
3	6689	32.9%	6385	33.1%
6	6081	33.8%	5793	34.0%
8	6225	33.4%	5937	33.5%
16	1377	59.4%	1345	59.4%

In all the cases the data remain the same, and only their interpretation changes. In other words, in each case we consider different condition attributes to have happened at the same time. For example, with a window size of 4, each 4 neighbouring attributes are supposed to belong to the same time step. With a window size of 5, there are 3 condition attributes at each time step, with the last time step containing only 1 condition attribute. The size and the quality of the tree change as the window size changes, because the

condition attributes will move from one time step to another. Consecutive window sizes result in similar tree sizes and error values, hence some window values are not shown in the Table. There is a sharp drop in the quality and size of the tree as we move into a window value bigger than 1, because the algorithm cannot use the attributes as freely as before. At the last window size, there is a drop in the quality of the tree, because some important attributes could not be used ay more. Increasing the window size beyond 16 (the number of condition attributes) has no effect on the results. Building a temporal decision tree in this way is a constraint satisfaction problem [60], where the choice of one attribute limits the future choices in that branch of the tree.

3.3 The TIMERS Algorithm

To use TIMERS, the user chooses a number of input attributes, which come from data ***D***. They include the decision attribute in which the user is interested, *da*, and also the condition attributes that will be used for classification. The other attributes present in the data can be ignored. There is no need to temporalise the input data for testing the instantaneous case, but for both causal and acausal cases the algorithm needs a window size. Since it is usually hard for the user to determine the most appropriate window size value, TIMERS accepts a range of values, from a starting window size α to an end value β.

It is possible that the condition attributes have no significant relationship with the decision attribute. In such cases, the classification rules will probably be of low quality. To detect such an eventuality, the user provides a threshold accuracy value ac_{th}. The resulting rule sets' accuracy values are compared to this threshold value, and if all

accuracy values are lower, then the algorithm does not have reliable data, and refrains from making any judgment.

TIMERS performs the appropriate temporalisation, generates classification rules, and saves the best accuracy values for each of the causal and acausal tests. In deciding the best method for describing the relationship between the decision and condition attributes, the size of the rules in each rule set is saved along with the corresponding accuracy value because the size of each rule set is considered as partially determining its simplicity.

After this phase, TIMERS decides on the best relationship type. In the simple case where there is no overlap between the accuracy intervals, the method that results in the best accuracy value will be chosen. If there is an overlap, then the complexity of the methods and the size of the rule sets are considered, as explained later in this chapter. The TIMERS algorithm appears in Figure 3.4 below. RuleGenerator() creates classification rules from input data, and Temporalise() performs temporalisation on data given a window size and a position within the window.

Input: A sequence of sequentially ordered data records ***D***, minimum and maximum temporalisation window sizes α and β, where $0 < \alpha \leq \beta$, a minimum accuracy threshold ac_{th}, a decision attribute *da*, and a confidence level *cl*. The attribute *da* can be set to any of the observable attributes in the system, or the algorithm can be tried on all attributes in turn. *Preference* determines whether the user prefers higher accuracy or a simpler method.
Output: A set of accuracy values and a verdict as to the nature of the relationship between the decision attribute and the condition attributes. It could be instantaneous, causal, or acausal.
RuleGenerator() is a function that receives input records, generates decision trees, rules, or any other representation for predicting the decision attribute, and returns the training or predictive accuracy, as well as the number of rules generated.

```
TIMERS(D, α , β, ac_th, da, cl, preference)
{
  ac_i = RuleGenerator(D, da);  // instantaneous accuracy; window size = 1
  for (w = α  to β)
        for (pos = 1 to w)
            (ac_w,pos , ruleSize_w,pos) = RuleGenerator(Temporalise(w, pos, D, da), da)
        end for
  end for

  ac_c = max(ac_α,α, …, ac_β,β)  // best causal test
  ac_a = max(ac_α,pos1, …, ac_β,pos2),  ∀ ac_x,pos, 1 ≤ pos < x  // best acausal result

  // Is there is enough relevant information?
  if (max(ac_i, ac_c, ac_a) < ac_th) then stop.

  Verdict = "for attribute " + da + ", "
  Relation = RelationType(cl, (ac_i, ruleSize_i), (ac_a, ruleSize_a), (ac_c, ruleSize_c), preference)
  case relation of
       INSTANTANEOUS: verdict += "the relation is instantaneous"
       ACAUSAL: verdict += "the relation is acausal"  // an element from the future is present
       CAUSAL: verdict += "the relation is causal" // all condition attributes in the past
  end case
  return verdict.
}
```

Figure 3.4 The TIMERS algorithm

The memory space needed by TIMERS is computed as follows. For every run of the *Temporalise*() operator, we get a dataset of $|\boldsymbol{D}| - (w - 1)$ records, hence the total number of the output records created by the TIMERS algorithm is $\sum_{w=\alpha}^{\beta} |D| - (w - 1)$. For a window size of 1, the dataset already exists (the original dataset). There is no need to save each temporalised dataset after it has been used for rule generation. Thus there are a

maximum of $|\boldsymbol{D}| - (\beta - 1)$ temporalised records during any iteration. Considering that the number of attributes in each record is multiplied by the window size, the maximum number of fields in the temporalised dataset will be $(\beta - 1) \times (|\boldsymbol{D}| - (\beta - 1)) \times Length_{\text{Rec}} + 1$. where $Length_{\text{Rec}}$ is the number of fields in each original input data record. The expression $(\beta - 1)$ reflects the fact that we only include the decision attribute at the current time. The 1 at the end of the formula reflects the decision attribute at that time.

Computation wise, the number of times that RuleGenerator() runs is equal to $1 + \sum_{w=\alpha}^{\beta} w = 1 + [\beta \times (\beta + 1) - (\alpha - 1) \times \alpha] / 2$. Hence, the time complexity of TIMERS is polynomially related to the time complexity of the RuleGenerator().

TIMERS uses a statistical test to see if the results of the three tests are close together, which results in a simpler method being suggested, even if its accuracy value is less than the more complex method. As we will explain later, simplicity depends on both conceptual simplicity (for example, an instantaneous relationship is conceptually simpler than a causal relationship) and also the number of rules needed to express a relationship (a lower number of rules denotes a simpler relationship).

In more detail, using the confidence level provided by the user in the *cl* parameter, TIMERS constructs a confidence interval for the accuracy, as in $\Pr[-z \leq f \leq z] = cl$. Normalisation leads to $\Pr[-z \leq (f - p) / (\text{sqrt}(p \times (1 - p) / N)) \leq z] = cl$. In this formula, f is the observed accuracy, N is the number of records, and p is the unknown actual accuracy value. Solving this equation results in upper and lower bounds for p [83]. The z values are determined by assuming a normal distribution. For example, a confidence level *cl* of 90% implies z is 1.65. After computing the interval, TIMERS checks to see if the

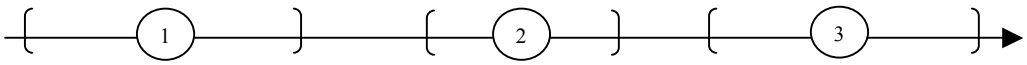
corresponding intervals overlap. If they do, the method with the simpler type of relationship is chosen. The intuition is that even if the simpler method has resulted in less accuracy, it could have *potentially* produced the same or better results.

As an example, suppose with a confidence level of 90%, we have: the instantaneous accuracy ac_{i} = 32.5%, $interval_{\mathrm{aci}}$ = [31%, 34%], the acausal accuracy ac_{a} = 35%, $interval_{\mathrm{aca}}$ = [33%, 37%], and the causal accuracy ac_{c} = 37%, $interval_{\mathrm{acc}}$ = [35%, 39%]. We assume all methods resulted in the same number of rules. Because the confidence intervals of the instantaneous method and the acausal methods intersect, instantaneous is chosen because it is considered simpler. Then we consider the causal case, and since the intervals of the instantaneous and causal methods do not overlap, the causal method is chosen as the final verdict because of its higher accuracy value.

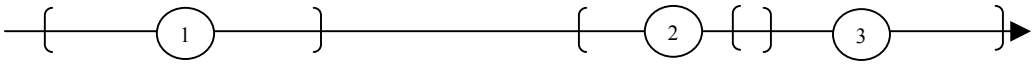
This example also demonstrates a special case where every two neighbouring intervals are overlapping. In this case, starting with the first two or the last two methods give different results. In the first case, as shown above, we choose the method with the highest accuracy. Here we started the comparisons from the left to the right, or lower accuracy to higher accuracy. But when starting from the right to the left (higher accuracy to lower accuracy) we end up choosing the simplest possible method. We leave the decision about which direction to follow to the user. In the TimeSleuth programme the user can choose between "Prefer simpler method" (right to left) and "Prefer higher accuracy" (left to right).

To clarify the previous paragraph, Figures 3.5(a) to 3.5(e) illustrate all possibilities of the accuracy intervals and the winning method in each case. A circled number from 1 to 3 represents the accuracy values of one of the methods: instantaneous, causal, and acausal.

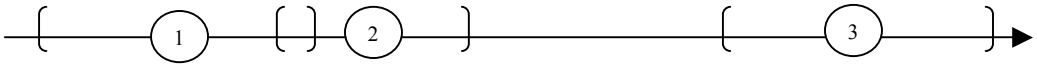
The accuracy values are sorted in an ascending order and the circled 3 represents the highest accuracy value, which could have been produced using any of the instantaneous, causal, or acausal methods. The brackets around each circle show the accuracy intervals. We assume that in all three cases the same number of rules was generated.



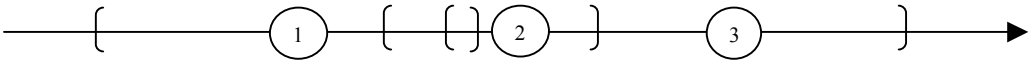
(a) No overlap. Method 3 is the winner.



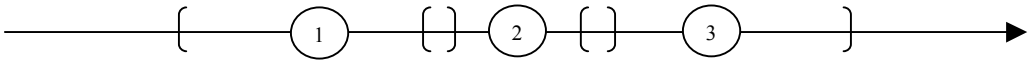
(b) Methods 2 and 3 overlap. The simpler of methods 2 and 3 is the winner.



(c) Methods 1 and 2 overlap. Method 3 is the winner



(d) All three methods overlap. The simplest method is chosen.



(e) Both pairs of neighbours overlap. If higher accuracy is preferred, 3 is chosen. Otherwise 1 is chosen.

Figure 3.5 Possibilities of accuracy intervals' relative positions

To determine which method/relation type to choose, we sort the accuracy values in either an ascending order (preferring the method with higher accuracy) or in descending order (preferring the simpler method). Figure 3.6 shows how the best method is selected.

Input: A confidence level *cl*, three accuracy values corresponding to the instantaneous, acausal, and causal methods: ac_i, ac_a, ac_c, and their corresponding number of rules: $nRule_i$, $nRule_a$, $nRule_c$, a preference *p* for higher accuracy versus a simpler method.
Output: A verdict as to the best relationship type.

```
//info[].method contains one of INSTANTANEOUS, CAUSAL, or ACSUAL.
//info[].Accuracy is the best accuracy value.
//info[].interval contains the interval of the accuracy value, computed using a confidence value

Function SelectRelationshipType(cl, (ac_i, ruleSize_i), (ac_a, ruleSize_a), (ac_c, ruleSize_c), p)
{
  // initialise the info[] structure
  forEach method ∈ { INSTANTANEOUS, ACAUSAL, CAUSAL }
     info[method] = (method , accuracy_method, ruleSize_method, Interval_method =
                                                  ComputeAccuracyInterval(accuracy_method))
  end forEach
  // if preference is given to higher accuracy, then start the search from lower accuracy values
  if (p = HIGHER_ACCURACY) then
    sort_Ascending(info[]); // sort in ascending order of accuracy.
  else  // SIMPLER_METHOD
    sort_Descending(info[])

  winner = 1
  for (count = 2 to 3)
     if (overlap(info[winner].interval, info[count].interval))
     {  // if there is  an overlap, then choose the simpler method
        if (info[count].method <simplicity info[winner].method and
            info[count].ruleSize ≤ info[winner].ruleSize) then
           winner = count
     }
     else
     { // if no overlap, choose the method with higher accuracy
        if (info[count].accuracy > info[winner].accuracy) then
           winner = count
     }
  end for
  return info[winner].method   //one of  INSTANTANEOUS, ACAUSAL, or CAUSAL
}
```

Figure 3.6 Selecting the best type of relationship

Starting with the two methods with the lowest (or highest) accuracy values, we test to see if there is an overlap among their confidence intervals. If so, then we choose the simpler method. The choice of the simpler method depends on both the conceptual complexity of the relation as defined above, and also the number of rules that are needed

to express the relationship. In our method the more rules needed to express a relationship, the more complex that relationship. This is reflected in our method in the following way: If a conceptually simpler method overlaps with a conceptually more complex method, but at the same time requires a bigger ruleset size, then priority is given to the method with the smaller ruleset size. In other words, for a simpler method to over-rule a more complex method, not only should there be an overlap between their accuracy intervals, but the simpler method should result in a smaller ruleset size. While our assumed order of complexity is subjective, including the size of rules adds an objective element to the complexity measure. If there is no overlap in the accuracy intervals, we choose the method with the better accuracy value. A winner is then selected among the first two methods. This winning relation type is then compared with the third method to determine the final method.

If needed, the algorithm in Figure 3.6 can also be used to select the best window size among a number of accuracy values obtained in either the acausal or casual case. In that case the order of simplicity is determined by the window size, with smaller window sizes being simpler.

3.4 (A)causality and (Ir)reversibility

The input to the TIMERS method is a set of sequential records. In general, we can represent the system under investigation as an automaton (function) with a transition function δ(). It is usual to assume that a transition function takes as input a current state and an event, where the event causes the state to change. The transition function then looks like δ(*q* = current state, *e* = current event), and the result is one (if the automaton is

deterministic) or more (if the automaton is non-deterministic) next states, chosen from the set of all the possible states Q.

In this paper, we do not see any need to distinguish between the current state and the current event. The current state is made up of a number of attributes, and collectively they lead to a change. In other words, the current event is considered to be part of the current state. A domain expert may decide to designate certain attributes as events if need be.

So a transition function is represented as $\delta(q) \in 2^Q$. In general, the reverse function δ^{-1} does not exist. In other words, we can only move forward in time with δ, and knowing the current state will not allows us to know the previous state. While this assumption is often made in the literature on automata, it is rarely specified explicitly. For this reason, a more appropriate name for δ would be a (unidirectional) "next state" function. In this case δ^{-1} would be called a "previous state" function.

If the transition function δ^{-1} exists, then we call the automaton *reversible*. In general, δ^{-1} exists only when the graph of the transitions has only one previous state for any current state. A time-reversible automaton can be deterministic or not, as long as the above condition is satisfied. Examples of a reversible function are $Inc(a) = a + 1$, and the logical *NOT* operator, where we can know the original argument if we are given the result. Examples of non-reversible functions are multiplication (given the output 6, what where the original operands?) and the logical *AND* operator. We now present these ideas formally.

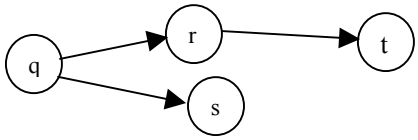
Consider a set of attributes $V = \{v_1,\ldots v_m\}$. We define the automaton $X(Q, \delta, Q_0)$, where $Q \subseteq V^m$ is a set of current states, δ: $Q \rightarrow 2^Q$ is the transition function, and $Q_0 \subseteq Q$

is a set of starting states. We do not include a $\sum$ set to denote the events for the sake of simplicity of the notation, and consider it integrated into the states. If needed, we can define a set Q' as the set of states, and a set $\sum$ to include the events, and then define $Q = Q' \times \sum$, and $Q_0 = Q'_0 \times \sum$. The automaton is called *time-reversible* if and only if there is a δ^{-1} function where

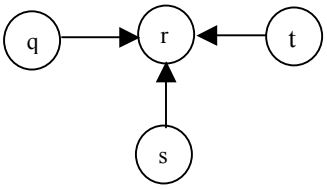
if $\forall q \in Q$, $\delta(q) \neq \varnothing$, then $q = \delta^{-1}(\delta(q))$.

It is proven that any δ that is a one-to-one (injective) function is reversible [16]. Such an automaton is backward-deterministic.

The above definition of δ^{-1} is different from an application of δ, where it is quite possible to have $q = \delta(\delta(q))$ for some or all states q. In such a case we are still traversing the graph of the automaton in the "forward" direction, while δ^{-1} allows us to *re-trace* a path in the graph in the backward direction. δ^{-1} answers the question "where we would have been before, if we wanted to be where we are now?." Figure 3.7(a) shows an example of a non-deterministic, but time-reversible graph. The graph is non-deterministic because there are two transitions out of state q. Figure 3.7(b) gives an example of a non-time-reversible graph, because when being in r, one cannot know with certainty where the previous state was. This situation is similar to the two-dimensional space where we could not be certain of how we ended up in a specific position (we could have moved into the current position from left, right, up, or down).



(a) A non-deterministic, time-reversible graph.



(b) A deterministic, non-time-reversible graph.

Figure 3.7 Examples of graphs with different reversibility properties

At each transition, we are only considering the information that is available at that point in time. So we do not remember the previous states or any events. Though automaton X forms a Markov chain [41] (meaning the next state depends only on the current state), a time-reversible automaton is not the same as a reversible Markov chain. In a reversible Markov chain the probabilities of a transition between nodes i and j is the same as those from node j to i [2], so the concept of a reversible Markov chain is devoid of time.

Reversible automata are closely related to the conservation of information and energy in physics. They are universal computationally [58]. Reversible automata are of important theoretical value, both in classical computer science where the problem of determining reversibility is non-trivial to solve in general, and is shown to be undecidable in the case of any cellular automata with a dimension of 2 or higher [31], and in quantum computing, where it is a requirement that the computation be reversible before it can be implemented by quantum computing methods [29] because the unitary operators [14] of

quantum computing are by definition reversible. It is shown that the more reversible a computation, that is, the more it is possible to recover past states from current states, the less complex, less costly, and less energy-consuming, the computation will be [5]. These results imply that there is an energy loss with information loss.

TIMERS' treatment of reversibility is different from the common notion explained above, as will be explained. TIMERS can generate rule sets that refer to both the previous and next values relative to the decision attribute, as shown in the sliding position temporalisation of Table 3.1. However, TIMERS can also use the forward and backward methods of temporalisations, with examples in Table 3.1.

Temporal rules generated in the forward direction of temporalisation refer to the previous observations to predict the current value of the decision attribute. This direction is reversed for rules generated after a backward temporlisation, where the next observations are used for predicting the current value of the decision attribute. The implication of this property is that TIMERS can be used to compare the ability of the condition attribute values to predict or retrodict the decision attribute's value.

If we consider the rules in the forward direction to represent the function $\delta()$, then the rules that determine the value of the decision attribute in the reverse direction can be represent the $\delta'()$ function. Given specific values for the condition attributes, one of the rules in the $\delta()$ function will fire and predict a certain value for the decision attribute. Running the rules in $\delta()$ on an input data sets allows us to gather statistics about how often each rule is run.

In general, there are more than one rule that determine the same value for the decision value. For example, the value *val* may be predicted or retrodicted by two or more rules.

In the $\delta^{-1}()$ function, given a value for the decision attribute, we can assign a probability value to each of the rules in $\delta()$, representing the likelihood that it could have been fired to produce that result. For example, suppose there are two rules for a certain value of the decision attribute, *val*, and given a dataset **D**, 60% of the rules that predict that value use rule number 1, while 40% use rules number 2. Now given that value *val* has been predicted, there is a 60% chance that the condition attributes match the left hand side of rule number 1, and a 40% chance that they match rule number 2.

Thus we see that the $\delta^{-1}()$ and $\delta'()$ functions are quite distinct from each other. $\delta^{-1}()$ does not contain the concept of running the rules backward in time, but the likelihood of being in a specific state. $\delta'()$, on the other hand, implies deriving and running rules that refer to the attributes in a backward order of time. In other words, in $\delta'()$ we are still using the values of the condition attributes to predict the value of the decision attribute, and the rules are referencing values backward in time.

Traditionally, if the $\delta^{-1}()$ function exists and is one to one, then we say we have a reversible relationship between the values of the condition attributes and the value of the decision attribute. TIMERS is not concerned with discovering this function, but it can produce both $\delta()$ and $\delta'()$, and derive their accuracy values. If $\delta()$ results in better accuracy values than $\delta'()$, then we define the relationship between the attribute values to be time irreversible. Our definition of irreversibility is analogous to the definition of causality, except we are using the forward temporalisation method, and not the sliding position temporalisation.

Similarly, if $\delta'()$ gives better accuracy values than $\delta()$, or they are close together, then we call the relationship between the values of the condition attributes and the decision

attribute's values as time reversible. Again, this is similar to the concept of acausality, but here we only use the backward temporalisation methods.

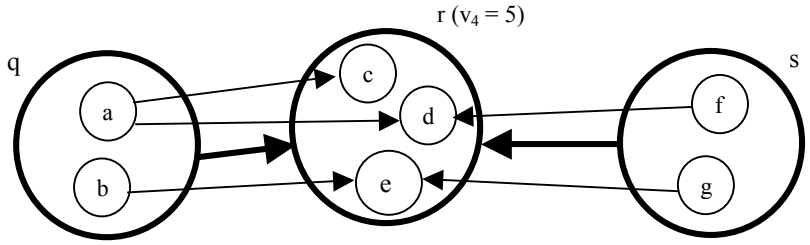
Now we establish a connection between state transitions and the TIMERS' sequential data. We can consider each line of input data in the input to TIMERS to be a state of the automaton, and the following line is considered the next state. Thus the input consists of a specific list of state transitions in the automaton, as generated by δ. We consider the normal direction of traversing a graph of state-changes to be the same as that of the temporal order of encountering the states in TIMERS' input. The reverse direction would be the opposite of the normal direction. Creating decision rules in the forward direction amounts to testing the causality of the data, while the decision rules that predict in the backward direction are a measure for acausality. This ability of TIMERS is especially of use for very complex or inaccessible systems where a formal investigation may be very hard or not possible, and we have to use observational data.

In this context, the sliding position temporalisation method is too general a treatment of time. For this reason we either reference the next attributes, or the previous attributes in testing for the reversibility of the data. TIMERS generates both $\delta()$ and $\delta'()$ to make a judgement about the time-reversibility of the relationship between the condition attributes and the decision attribute.

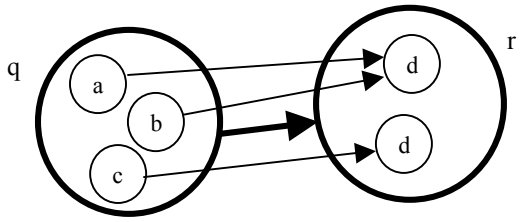
To see how TIMERS' $\delta()$ function is related to the original state transition function, we note the following. Normally, a state is represented by the values of all the attributes, while TIMERS uses classification of a single attribute. Suppose attribute v_i is set to be the decision attribute, and takes on values from the set Range(v_i). The resulting decision rules form the transition function δ: $Q \rightarrow$ Range(v_i). In general, Range(v_i) can be considered as

a merging (generalisation) of many states in V^m, where attributes other than v_i are ignored. Since the left hand side of the rules in δ may also miss certain attributes, generalisation is performed there too. Figures 3.8(a) and 3.8(b) show how in δ′ many states in δ can be merged into one. In these figures the original state transition function is shown with thinner lines, while δ is denoted with thicker lines. In δ we move from *q* to *r* whenever in the original function we move from any state in *q* to any state in *r*. Consider a decision rule such as if {(v_1 = 1) and (v_2 = 2)} then (v_4 = 5), where v_3's value is not important. The merged states in the right hand side of the transition in δ all have the same value for the decision attribute v_4 (v_4 = 5), and the rest of the attributes are ignored. The merged states in the left hand side all match the condition attributes in the left hand of a rule in δ, i.e., they all have v_1 = 1 and v_2 = 2. We assume that each original state can be applied to only one rule in δ. In other words, we assume that the rules in δ can be applied unambiguously.

Figure 3.8(a) shows two rules in δ that predict the same value for the decision attribute. So both rules have the same right hand side, as represented by the merged states of *r*. Each rule generalises different states of the original state transition function, as specified by the left hand side of the respective rule.



(a). An example of generalisation (merging) of states.



(b). δ is deterministic, but the original function may not be so.

Figure 3.8 Different mappings from the original function to δ

δ predicts the decision attribute's value using the condition attributes. One would expect δ′ to be obtained by re-tracing δ, which would lead us from the decision attribute's value back to the condition attribute's values, similar to the inverse of the original state transition function. This is not the case, and δ′ also predicts the value of the decision attribute using the condition attributes' values, but it refers to the next (and not the previous) condition attributes to do so.

This distinction means that unlike the usual case with the graph of an automaton, we should not literally go back along the δ transitions to create δ′. In other words, δ (from a causality test) and δ′ (from an acausality test) have distinct graphs, and should be traversed independently.

As in Figure 3.8(a), each rule discovered in δ or δ′ creates a link in its respective graph. Since we traverse the graphs in one direction only, there is no need for the graphs to be one-to-one. The graphs can be traversed deterministically because we assume the rules can be applied to the data with no ambiguity, so we always know where to go from any given state by finding the rule that applies to the data. This characteristic is reflected in Figure 3.8(b).

Considering that δ and δ′ are deterministic, it may seem that TIMERS allows us to get rid of any original non-determinism in the state transition function that generated the sequential data. Considering that each of δ and δ′ rule sets is associated with an accuracy value that reflects how often the rules make a correct prediction, any non-determinism is in fact reflected in the accuracy values of the rule sets, which could be less than 100%.

When given a series of sequential data, the user can choose a different decision attribute when running TIMERS, and each choice may result in a different verdict. The results are qualified with respect to the choice of the decision attribute, so one should say δ is (ir)reversible with respect to a decision attribute with accuracy *a*%. This verdict is then considered a hint at the (ir)reversibility of the original data.

Chapter 4

Derivation and Presentation of Temporal Rules

In this chapter, we show how temporal rules can be derived from sequential data. In Section 4.1 we introduce the *TimeSleuth* programme. TimeSleuth allows the user to select the decision and condition attributes, do discretisation and aggregation, derive rules and evaluate them, and represent the results in tabular forms. The tabular representations of the rules summarise the relationships and lays them out in a temporal order, making them perhaps easier for the user to understand [36].

Section 4.2 introduces a graphical method of representing rules, called a *dependence diagram*. Such diagrams are meant for better understanding of the relations among attributes and can represent any classification rule, and not just temporal ones. The tabular representations of TimeSleuth focus on a single decision attribute at a time, while a dependence diagram can show multiple decision attributes and their relationships together. However, unlike the tabular representations, a dependence diagram does not explicitly display temporal orders. A dependence diagram can be derived from the tables of TimeSleuth, but not vice versa.

An alternative form of presenting a temporal rule is as a Prolog statement, and is more suited for automatic usage. In Section 4.3 we introduce this possibility by way of an example in programming an artificial robot.

4.1 The TimeSleuth Software

TimeSleuth implements TIMERS II and provides a graphical user interface for the user to experiment with different settings and options. The user interface provides a number of tabbed panels. In each panel the user can perform a specific operation. In the Attributes panel, for example, the user selects the decision attribute(s) and the condition attributes, while in the Recommend panel the user is offered clues as to the nature of the rule set. TimeSleuth is written in Java and uses C4.5 as its rule generator. We chose C4.5 because it has become the default rule generator in the literature, and also because of its source code availability.

We have modified C4.5's source code so it can communicate with TimeSleuth, accepting new input commands and outputting relevant temporal data. As a result the user views C4.5 as integrated into TimeSleuth, even though C4.5 is called as an external programme. The modified source codes for compilation under different operating systems, as well as a precompiled version for use under 32 bit Microsoft Windows products are included in TimeSleuth's package. Detailed information about TimeSleuth can be found as online help files in the TimeSleuth package.

TimeSleuth can be used, with reduced abilities, even if C4.5 has not been modified for TimeSleuth. An option in TimeSleuth allows the user to inform it of this situation, so

it will not provide the non-modified C4.5 with options it cannot understand. In this case TimeSleuth mainly becomes a graphical interface for C4.5.

C4.5's input consists of at least two files. A .data file which contains the values of observed attributes, and a .names file that contains the names and possible values of the attributes. In the data file each line consists of the value of the condition attributes that determine the value of a single decision attribute. The decision attribute comes at the end of the line. The values are separated by comas (,). An example .data file looks as in Figure 4.1:

```
1, 0, 7, 8, 3, 4, 5
1, 0, 4, 13, 2, 5, 4
1, 0, 7, 8, 2, 4, 3
1, 0, 12, 0, 3, 3, 4
1, 0, 7, 8, 0, 4, 4
0, 0, 11, 8, 1, 4, 4
1, 0, 7, 8, 0, 4, 4
0, 0, 11, 8, 2, 4, 3
```

Figure 4.1 Contents of a .data file

There are 6 condition attributes (the first 6 attribute values in each line) and one decision attribute in these data. The decision attribute takes on the values 3, 4 and 5. C4.5 accepts continuous, discrete, and symbolic input values. After the rules are generated, C4.5 tries them on the contents of the .data file to measure the training accuracy value.

The decision attribute remains nameless in a .names file. By default C4.5 calls the decision attribute “class.” TimeSleuth calls the decision attribute simply "originalDecision." The range of the class come first in the .names file. The names and ranges of the condition attributes then follows. A matching example for the data in Figure 4.1 would be as in Figure 4.2:

```
1, 2, 3, 4, 5, 6
x: 0, 1
y: 0, 1, 2, 3
teta: continuous
zeta: continuos
u1: 0, 1, 2, 3, 4
u2: 3, 4, 5
```

Figure 4.2 Contents of a .names file

In this file we see that the decision attribute can take on discrete values from 1 to 6, and the names of the condition attributes are x, y, teta, zeta, u1 and u2. The mapping from names to data values for the first line of data is shown in Figure 4.3.

x	*y*	*Teta*	*Zeta*	*u*1	*u*2	*decision*
1	0	7	8	3	4	5

Figure 4.3 Attribute names and corresponding values

In addition to .data and .names files, C4.5 can be provided with a .test file. The format is exactly like a .data file. The contents of the .test file are not used during the tree and rule generation phase, but are consulted to test the generated trees and rules to measure the predictive accuracy of the tree or the rules.

All three files (names, data, test) should have the same name, and their extension determines their type. An example would be weather.names, weather.data, and weather.test.

Figure 4.4 shows a snapshot of TimeSleuth's window, showing the input handling panel. After reading the input files, a list of the discovered attributes is presented to the user. One can select a decision attribute by highlighting it. The user should inform TimeSleuth about the presence of a .test file by clicking on the appropriate checkbox.

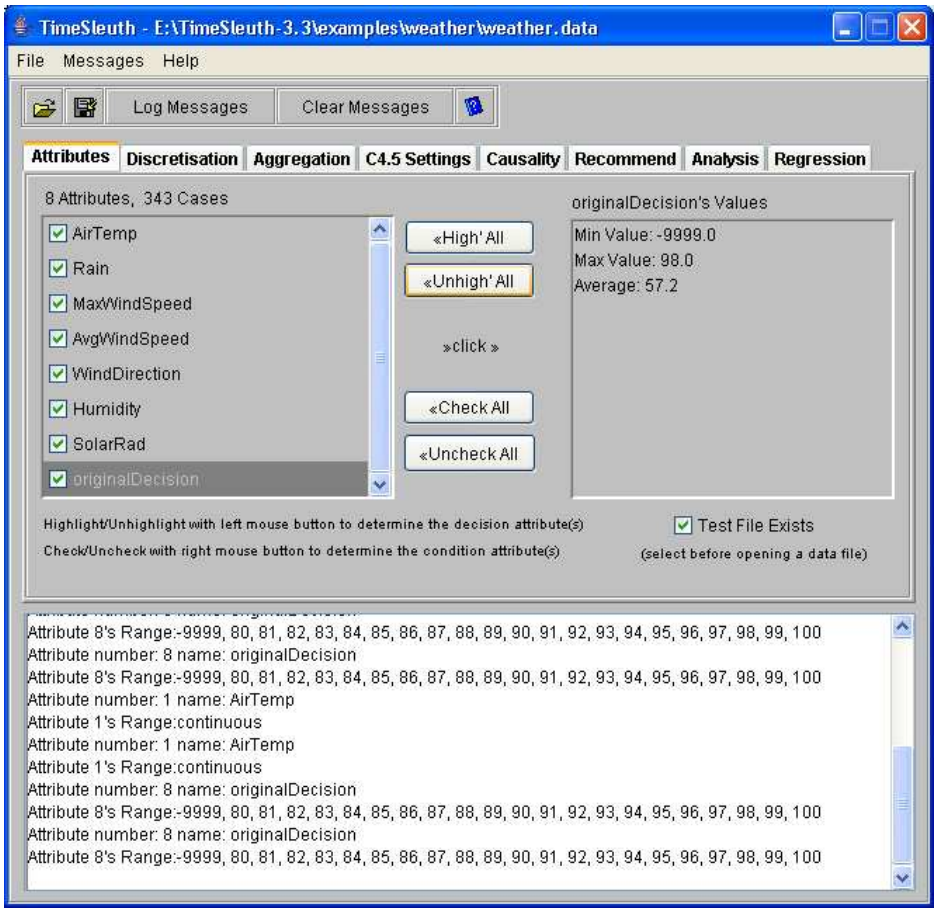


Figure 4.4 TimeSleuth 's input handling panel

The data used in Figure 4.4 and the following figures contains weather observations gathered hourly. The decision attribute, "originalDecision" in TimeSleuth, is the Soil Temperature.

Unlike with standard C4.5, in TimeSleuth the user can choose more than one decision attribute. In such a case, C4.5 is invoked multiple times, each time with a different attribute as the decision attribute. Using the original input files, TimeSleuth automatically generates the appropriate .data, .test, and .names files which contain the temporalised data and attribute names.

The user can choose to discretise the attributes using the "Discretisation" panel as shown in Figure 4.5. Two methods are available. In Method one, the attribute's range of values, as present in the input data file, is divided into segments of equal length. Method 2 takes into consideration the distribution of the values, and divides the range into segments that contain the same number of values. This consideration ensures the values in a denser region of the attribute's range are better represented. The steps taken in the discretisation process are shown in Figure 4.5.

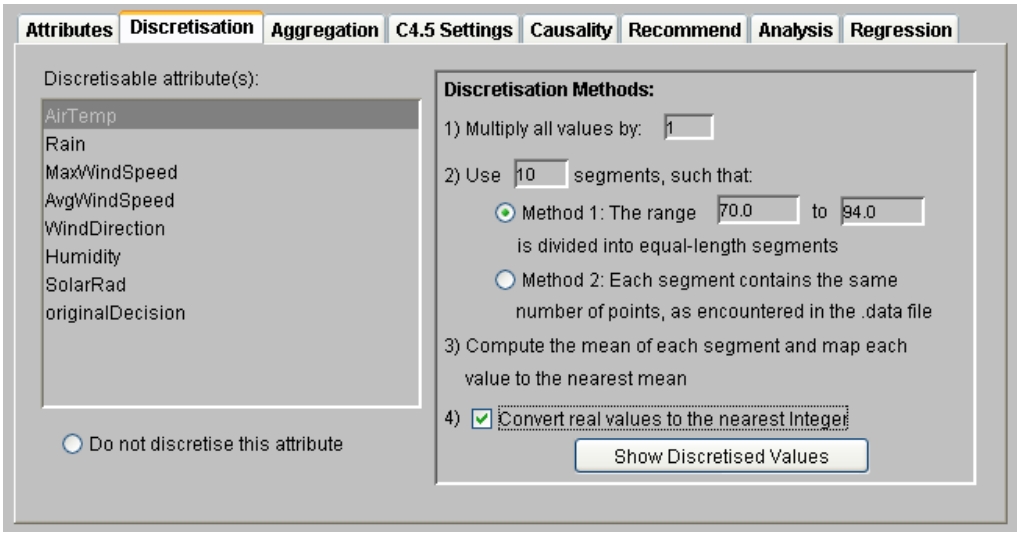


Figure 4.5 The Discretisation panel

The main assumption in TimeSleuth is that the program is provided with the values that are observed at consecutive time steps. However, the user can instruct TimeSleuth to use the aggregate value of an attribute in forming its rules. For example, the user may decide to investigate the effect of the minimum value of the Air Temperature during a certain window size, as shown in Figure 4.6. In such a case, TimeSleuth computes the minimum, and uses that value instead of individual temperature values. Choosing the aggregate function min() on x_1 with a window size of 2 would result in the record: $< x_{12}$, x_{13}, x_{14}, x_{15}, x_{22}, x_{23}, x_{24}, x_{25}, $\min(x_{11}, x_{21})>$. In the actual output file, the last attribute will

be the decision attribute because that is the format expected by C4.5. The effect of having aggregate attributes is explained later, when we introduce the classification panel.

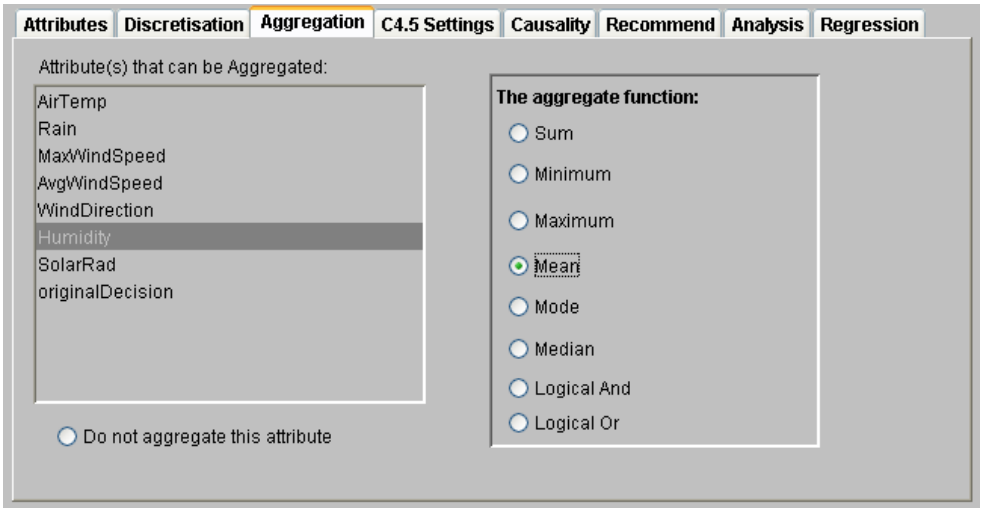


Figure 4.6 The Aggregation panel

The actual running of C4.5 happens in the C4.5Settings panel, as shown in Figure 4.7. It allows the user to specify the location of C4.5's executable files and other related directories. The user can also provide any optional run-time arguments to C4.5, even though the arguments needed for TimeSleuth's functioning will be automatically provided. Here the user can inform TimeSleuth if C4.5 has not been patched. The default values provided in this panel make sure that C4.5 will function in a backward compatible way, in case it has not been modified to work with TimeSleuth.

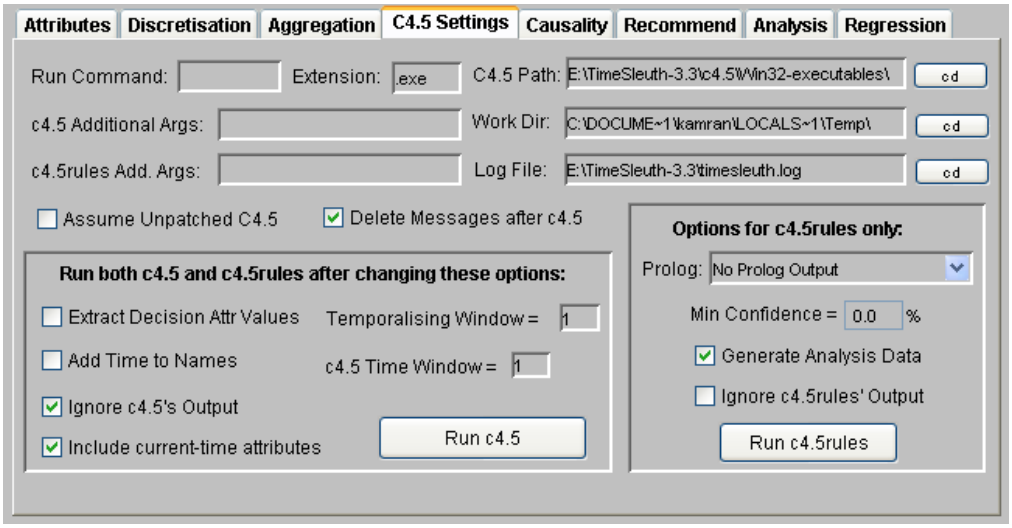


Figure 4.7 The C4.5Settings panel

If C4.5 is not modified, TimeSleuth can still be used to discover causal relations, because it can still perform the temporalisation operation. However, there is a potential problem here: C4.5 does not rely on the names of the attributes to identify them. Rather, it uses their locations in the .data file. If TimeSleuth simply temporalisess the data and copies the names in the .names file multiple times, then in the output the attribute's names from different time steps would be confused. The temporally indexed attribute names as in x_{11}, x_{21}, x_{31}, etc. are not actually differentiated in a .names file, and all are called x_1 by C4.5. In other words, with an unmodified C4.5 we may not be able to distinguish the same attribute at different time steps. If C4.5 has been modified, it actually outputs temporal information in the rules by sorting the attributes according to their time of appearance. So a rule might look like this:

IF {At Time 1: (x_1 = 1) AND At Time 2: (x_4 = 1) AND AT Time 3: (x_1 = 5)} THEN At Time 4: x_5 = true

As seen later, TimeSleuth uses a tabular form to display the same information. If C4.5 is unmodified, then the user can instruct TimeSleuth to add a time index (_t<time>) to the

names it generates. So even though C4.5's output may be temporally out of order because no sorting is done, but it will still be understandable because the attributes from different time steps are distinguishable. An example is the following rule: IF {(x_1_t1 = 1) AND (x_1_t3 = 5) AND (x_4_t2 = 1)} THEN x_5_t4 = true.

The rules generated by c45rules have a confidence level. In order to filter the rules, a user can specify a minimum confidence level, and only rules with higher confidence values will be presented.

If aggregate attributes are present, then the output rules will include the keyword "During Window" to make it clear that the aggregate value of the specified attribute is seen during the whole window. A rule would look like: IF {During Window: x_3 >= 0 AND At Time 1: (x_1 = 1) AND At Time 2: (x_4 = 1) AND AT Time 3: (x_1 = 5)} THEN At Time 4: x_5 = true

C4.5 is supervised, because the user has to indicate the decision attribute, and also has to tell C4.5 exactly which values will be taken on by the decision attribute. TimeSleuth allows the user to specify more than one decision attribute, and can optionally extract the values taken on by the decision attribute from the .data file. It outputs a warning message if the .test file contains a value that is not seen in the .data file. These abilities turn TimeSleuth into an unsupervised tool, as the user simply has to run the program with minimal instructions as to the target attribute and the values it can have.

C4.5 consists of two main programmes. The first one, c4.5.exe, creates a decision tree, and the c4.5rules.exe generates classification rules from this tree. There are two time windows in the classification panel. The first one, called "Temporalising Window" is as the name implies. Its value is used to temporalise the input data for both c4.5 and

c4.5rules. This value is then provided to c4.5rules so that it can sort the output temporally. However, this value is not provided to the tree generation program, c4.5. So the decision tree is generated with no regard to any window size. The value "c4.5 Time Window" is meant to affect the decision tree, as explained in Chapter 3. This value, if different from 1, should be the same as the temporalisation window size.

Finding a suitable window size for the data under investigation can be a challenge. For this reason TimeSleuth allows the user to run it in a batch mode, wherein it tries consecutive values for the window size. Training and testing accuracy can be employed to guide the search. As shown in Figure 4.8, the user can decide to explore all values, or stop after the accuracy has reached a threshold, or after the accuracy has stopped improving. The running time is determined by C4.5's speed, which in our experiments has been good, even for fairly large values of the window size.

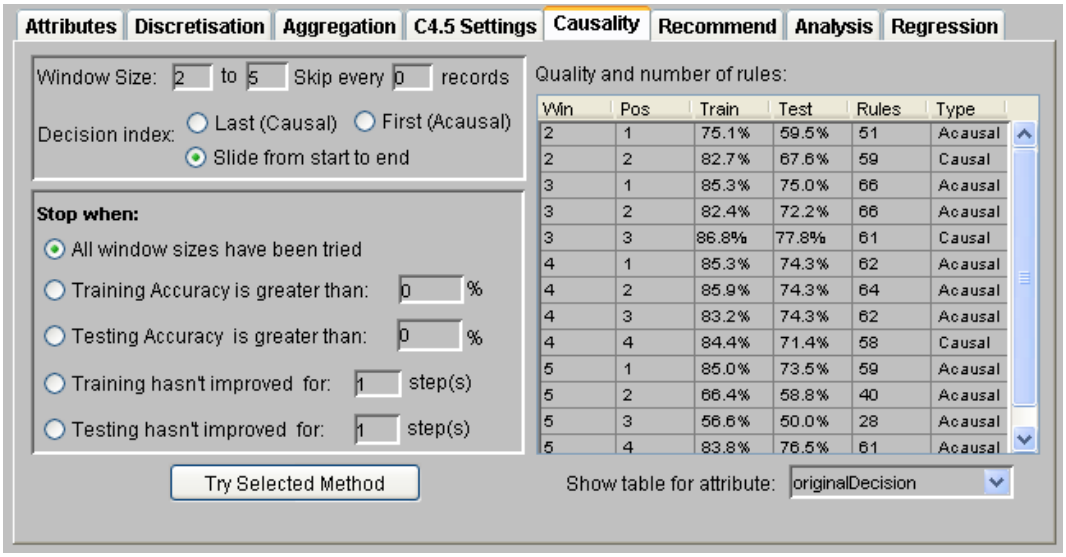


Figure 4.8 Exploring different window sizes

After generating accuracy values using different methods, a decision has to be made as to which method gives the better results. The Recommend panel performs this test.

The best values of each of the instantaneous, causal, and acausal tests are imported from the cuasality panel into the recommend panel. Alternatively the user can fill in the fields with appropriate values. The recommendation of a method can be offered based on both the training and testing results, as shown in Figure 4.9 below.

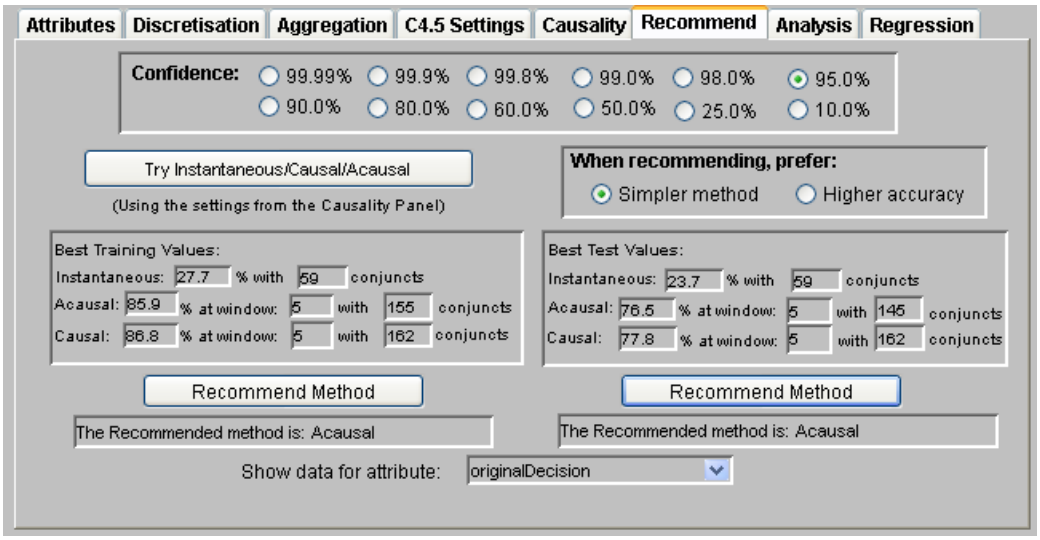


Figure 4.9 The Recommend Panel

Though C4.5 generates rules for only a single decision attribute at a time, TimeSleuth can handle multiple decision attributes by repeatedly calling C4.5. The user selects the decision attributes, and different rule sets are created for each such attribute. TimeSleuth presents the rules in different tables. The user has the choice to see the rules laid out according to the time of each attribute's appearance. It is also possible to summarise the rules and see which attributes appear in what time steps. The user can optionally see which attributes are actually used to classify the training and testing data. These information are presented to the user in the Analysis panel.

In the analysis panel, the user can see how the selected window size has affected the resulting rules. As shown in Figure 4.10, the user can opt to see the rules laid out

according to the time steps in which the attributes appear. This display shows how important each attribute has been in forming the rules.

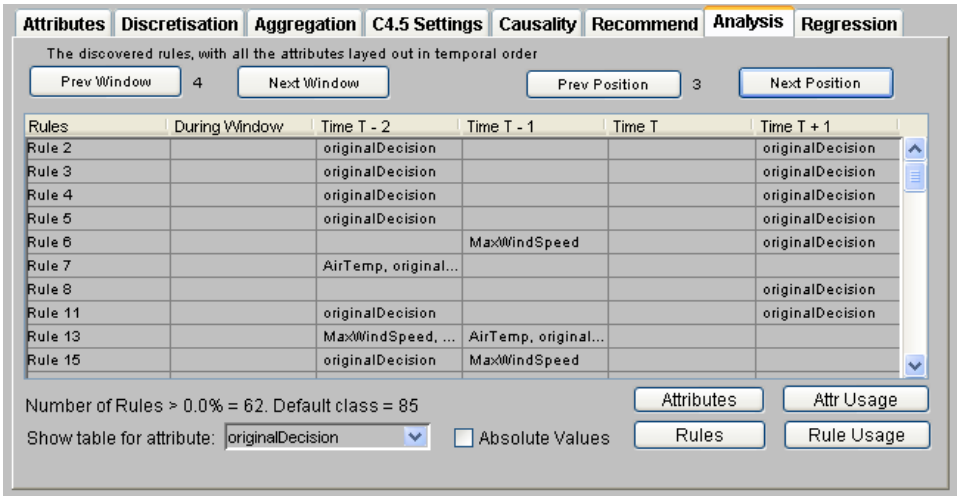


Figure 4.10 Temporal layout of rules

In Figure 4.11, we see that many of the condition attributes used to determine the value of the soil temperature (originalDecision) come from previous time steps. In other words, the current temperature of the soil depends on attributes measured previously. Using standard C4.5 with such data obviously would not be as revealing. As seen in Figure 4.11, the previous value of soil temperature appears in 84.3% of the rules, which supports the common sense guess that the current temperature is related to the corresponding observation an hour ago.

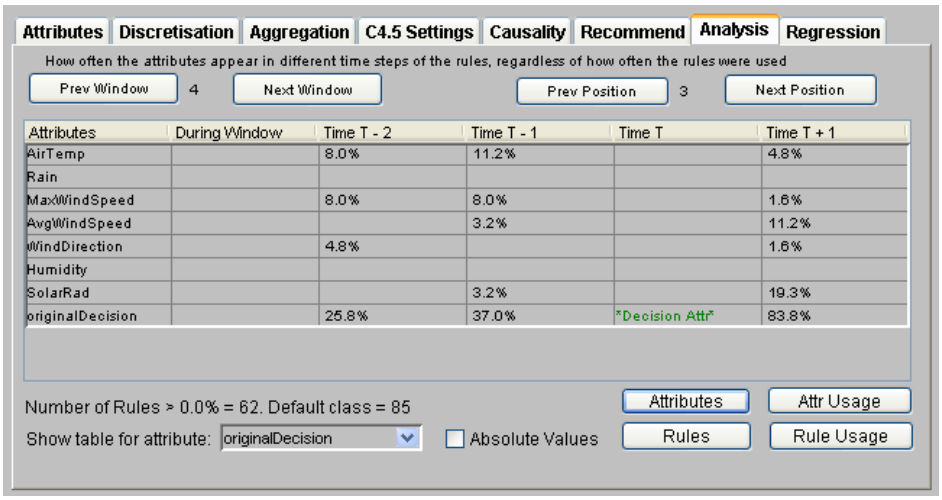


Figure 4.11 Statistics about the attributes

In Figure 4.12, TimeSleuth shows the frequency of attribute usage in rules that were actually fired. In other words, the more a rule has been used (on test data or on training data), the more important those attributes will be. Both training and testing results are displayed. The two values are separated by ":".

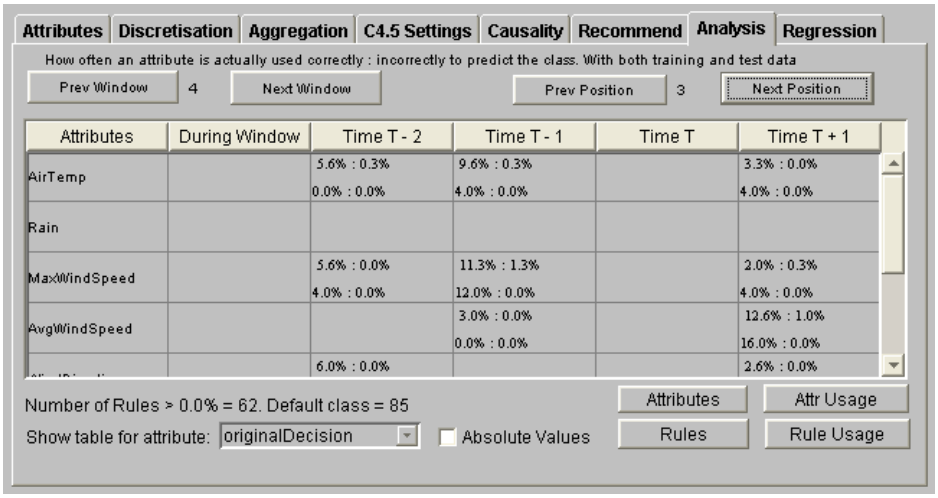


Figure 4.12 Frequency of attribute usage in rules

Figure 4.13 shows some addition information about the rules and how they were used. The column headers are self-explanatory.

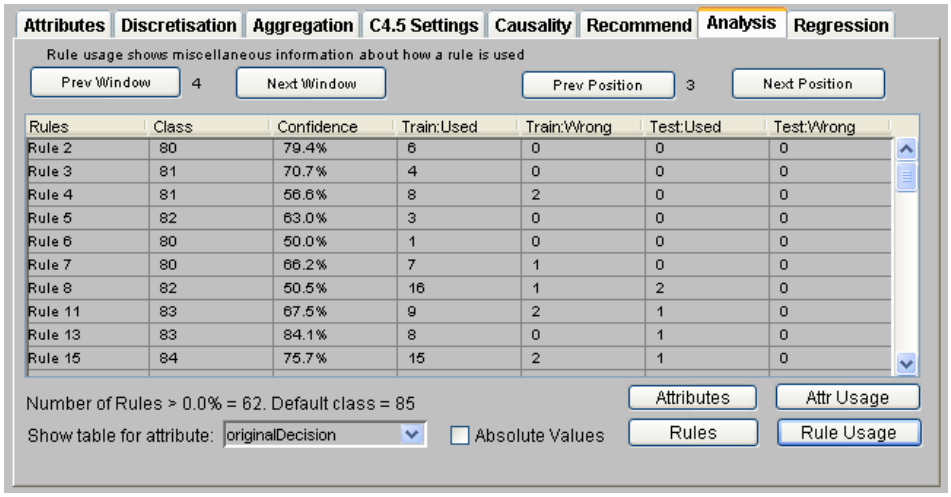


Figure 4.13 Rule usage and other related data

In TimeSleuth the user can opt to have aggregate attributes. An aggregate attribute contains the value of an attribute's values combined with a function over the length of the window. For example, one could have the Logical AND of the value of "Play" over 3 days as a single attribute. This new attribute replaces the "Play" attributes over the 3 days. In this example the result would be a "Yes" if it were possible to play in any of the three days, and "No" if it was not possible to play in at least one of the three days that appear in the temporalised record. An aggregate attribute is not considered to have a specific time of occurrence, and so is ignored in the decision making processes as to the nature of the relationship.

More precisely, an aggregate attribute is considered to have happened at time 0, which is treated specially. The name of the aggregate attribute comes from the name of the function and also the name of the original attribute. In the current example it would be and(Play). All attributes except the decision attribute can be aggregated. TimeSleuth supports the following aggregation functions: Sum(), Minimum(), Maximum(), Mean(), Mode(), Median(), And(), and Or().

4.2 Dependence Diagrams

Here we introduce building diagrams that show the attributes' dependence on each other. In a *dependence diagram*, attributes are connected together based on how much they are actually used in predicting each other's values. A decision attribute depends on another (condition) attribute if it appears in rules that are used to predict the decision attribute. The rules can be causal, acausal, or instantaneous. A dependence diagram does not make any distinctions in this regard. An example dependence diagram for the robot data is shown in Figure 4.14 below. The data comes from an artificial robot doing a random walk in a two-dimensional space, where *x* and *y* denote the position, *a* is the random action taken (the direction of movement) and *f* shows the presence or absence of food.

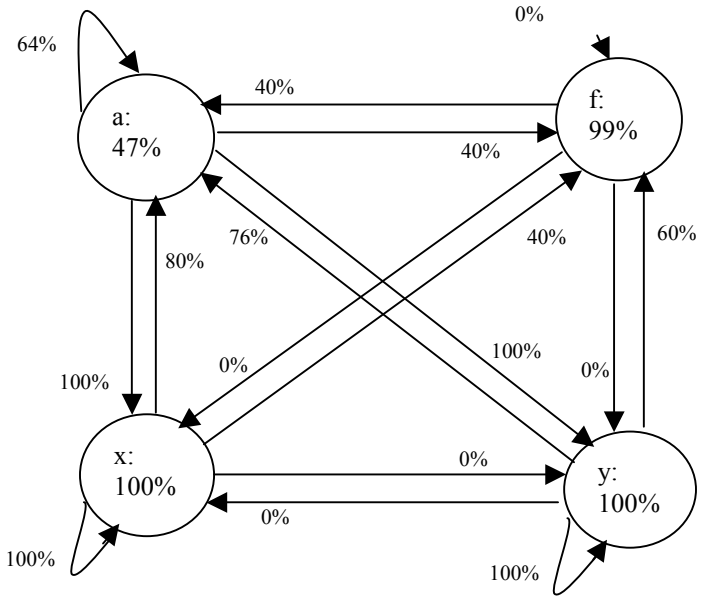


Figure 4.14 An example dependence diagram

4.2.1 Definitions

Definition 4.1: *Static Dependence*

For any set of rules R that predict the value of attribute d,

If condition attribute a appears in at least one rule, then d depends on a.

- If $r \in R$, and $a \in$ CONDITIONS(r), then d = DECISION(r) depends on a.

Regardless of specific time steps, the more rules in which attribute a appears, the more the dependence of d on a.

- d depends on a with strength s% if a appears in s% of the rules, written as $D_d(a)$

In a temporal rule, a may appear at different time steps. The frequency of its appearance in the rules determines the strengths of the dependence.

- In rule r, d depends on a with strength s% at time step t if a appears in s% of the rules at time step t, written as $D_{d,t}(a)$

To compute $D_d(a)$ in a temporal rule, we use the following method:

With a window size $w = 1$ and 2: $D_d(a) = D_{d1}(a)$

With a window size $w > 2$: $D_d(a) = \max(D_{d,1}(a), \ldots D_{d,w\text{-}1}(a))$

Example: Suppose the ruleset R contains 2 rules: {[if at Time 1: <a = 1> and at Time 2:<a = 1> and <b = 2>, then at Time 3: <d = true>], [if at Time1: <a = 3> then at Time 3: <d = false>]}.

Here we have the following: $D_d(a)$ = 100% (a appears in all rules), $D_d(b)$ = 50% (b appears in half of the rules).

From the previous example we have $D_{d1}(a)$ = 100%, $D_{d2}(a)$ = 50%, $D_{d1}(b)$ = 0%, $D_{d2}(b)$ = 50%.

Definiton 4.2: *Dynamic Dependence*

Dynamic dependence is similar to static dependence, except the strength of the dependence is determined by the rules that actually get used for determining the value of the decision attribute. In other words, only the rules that get fired by the test dataset are considered for determining the strength of the edges.

In the previous example, suppose only the first rule was fired. In this case we have: $D_d(a) = 100\%$, $D_d(b) = 100\%$. $D_{d1}(a) = 100\%$, $D_{d2}(a) = 0\%$, $D_{d1}(b) = 0\%$, $D_{d2}(b) = 100\%$.

In what follows, the phrase "appears in a rule" designates static dependence, while "used for prediction" characterises dynamic dependence.

We use a threshold to prune weak and accidental dependencies.

Definition 4.3: *Threshold Dependence and Independence*

d is dependent on *a* if $D_d(a) > \varepsilon$, where ε is a user-specified threshold.

Otherwise, *d* is independent of *a*.

Definition 4.4: *Dependence Diagram*

A dependence diagram is a possibly cyclic, directed, weighted graph $<N, L>$, where N is a set of nodes, each representing an attribute, and L is a set of links, each representing the dependence of a decision attribute on a condition attribute. The direction of the link is from the condition attribute to the decision attribute. Hence a node represents a decision attribute when links are pointing to it, and a condition attribute when links are pointing away from it. There are values (weights) assigned to both the nodes and the links.

To create a dependence diagram, first the rulesets for predicting the values of one or more decision attributes should be generated. The diagram then follows from the rulesets. In both static and dynamic dependencies, the weight of a link is the same as the strength

of the condition attribute. For nodes, in a dynamic dependency, the weight of a node is the training or testing accuracy of the rules created for predicting the decision attribute that corresponds to the node. Nodes have no weight in static dependency because rules are not run, hence no accuracy values are available. Traversing a dependence diagram beyond one link is not possible, as by definition only the immediate links to and from a node are meaningful. This characteristic sets the dependence diagram apart from a standard graph.

4.2.2 Pruning a Dependence Diagram

Pruning a dependence diagram is performed at two levels. At the link level, we prune links that do not have enough strength, or in other words, enough importance, in determining the value of a decision attribute. At the node level, we remove whole relationships, where the evidence for the relationship is not good enough. Node level pruning is only possible with dynamic dependence diagrams.

Link oriented pruning: Each link in the diagram represents the strength of a decision attribute's dependence on a condition attribute, as determined by the number of times the attribute has appeared in the rules predicting the decision attribute. Links that have a strength value below a certain level can be pruned.

Node oriented pruning: Each node in a dependence diagram represents an attribute. Associated with each node is the training or testing accuracy for that attribute when it is set as a decision attribute. This accuracy value is called the node's strength. All the links

that point to nodes with a strength value below a certain level are pruned, regardless of their strength. The node itself is removed if all the links to and from it are pruned, and its strength is below the threshold.

Examples of pruning dependence diagrams are presented in the chapter on experimental results.

4.3 Temporal Rules as Prolog Statements

In this section we describe how TimeSleuth generates Prolog Statements, to be used in any situation where rule execution is needed. Specifically, we show how these statements can be turned into a plan of action for programming a simple creature's movements in an artificial life [48] environment called URAL (University of Regina Artificial Life) [87]. This representation is especially appropriate when the rules have been derived with a window size of two, implying a "before" (the antecedent of the Prolog statement) and an "after" (the consequence of the statement).

4.3.1 Rules Governing an Artificial Robot

Programming robots usually involves writing special programme that determine the robot's behaviour given the sensor values. There have been attempts at automatic programming of robots. Such as Reinforcement Learning or Genetic Programming. But here too a domain expert must explicitly provide the system with high-level information.

For Reinforcement Learning the work includes deciding on the behaviour that should be reinforced, and a payoff function, among others. For Genetic Programming one should determine sets of terminals and primitive functions and come up with a suitable fitness

function. A discussion of this aspect of Genetic Programming and Reinforcement Learning appears in [45]. Implementing a system to actually produce the results should come next. There is no guarantee that performing these pre-processing steps will be easier than writing a program manually. In the rest of the section we show how Prolog rules are created, and investigate automatically generating Prolog programmes in the simple environment of URAL.

URAL is a discrete event simulator where known atemporal and temporal rules govern an artificial environment. Having complete knowledge about the URAL domain allows us to judge the quality of the discovered rules, and this property will be explored in the chapter on experimental results. Other kinds of data would have required interpretations as to the true nature of relations among the attributes, making the process of judging the output more complex and open to debate.

The world in URAL is a rectangular, two-dimensional board with a robot living in it. Food exists at specific locations on the board. The robot can sense its position and also the presence of food at the current position. The food (source of energy) can be used to increase the energy level of the robot. The robot performs a random walk in the domain: at each time-step, it randomly chooses one of the following actions: left (L), right (R), up (U), or down (D). Left and right correspond to moving along the *x*-axis and up and down to moving along the *y*-axis. The robot can sense which action it takes in each situation. If it attempts to move beyond the boundaries of the board, the move is ignored and the location of the robot is left unchanged.

In URAL, we can see the contents of the robot's "brain" in a window as shown in Figure 4.15. Each circle in this figure denotes a (*x*, *y*) position in the world. The empty

locations have not been explored, or they may denote obstacles. The lines show the paths taken by the robot as it has moved from one location to the next (the lines start at the center of the starting location). An obstacle can be recognised if there is a line from the center of a location to itself, which means that a move from that location towards the obstacle resulted in the robot remaining in the same position.

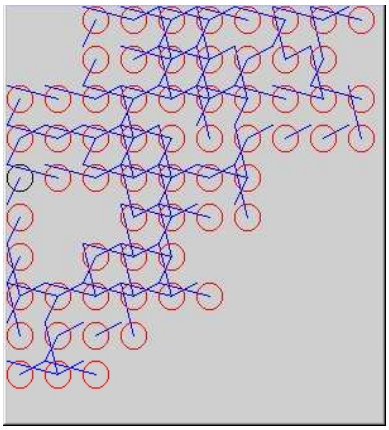


Figure 4.15 A visual representation of the robot's brain

The brain is the learning mechanism of the robot and follows conventional Situation Calculus [54] concepts, where different situation are linked together via movement actions. As time passes, more and more of the board are explored unsupervised, and the brain will register the locations and the appropriate movement to go from one location to the next. There is a one-to-one correspondence between the locations in the brain and the world. This brute-force memorisation of the explored locations allows a planner to find a way from the robot's current position to a location that contained food the last time it was visited. The planner simply finds a route from the current location to the location of the food. The user can activate the planner, or it can be activated automatically whenever the robot's energy reaches a certain threshold. The success of the plan depends on whether the world is dependable, i.e., if what we observed in the past is valid now.

The intuition behind this learning approach is to automate the task of teaching a robot to behave properly in an unknown environment. The robot explores the environment by itself during a learning phase, and makes a map in its brain. The user can then ask the robot to go to a desired position. If the robot has already explored that position, then it can plot a series of actions to get to the requested position. For this scenario to work, the environment should be simple enough for the robot to be able to memorise the interesting positions and the sequence of actions that should be performed to get there. Surprises (ending up in a position that is not on the original path to the desired position) can be handled by re-planning to find a new path from the current, unexpected, position to the desired one.

The main problem with this approach is that it cannot perform any generalisation. The brute-force approach is exponential in nature and thus inefficient in the amount of space it needs. The bigger the map of the environment, the harder it would be for it to be searched for a path.

One way of manually programming a robot is by using Situation calculus, which is a method of describing the effects of actions. Each situation can be considered a snapshot of the values of a set of attributes. One can move from a situation to another by performing actions. This movement can result in a possibly cyclic graph with situations as nodes and actions as links between the nodes. One can interpret the transitions between the situations as the execution of rules. The starting situation forms the left hand side of the rule, and the resulting situation forms the right hand side. Planning is easy here: To go from the current situation to a desired situation, first make sure that they are both in the graph, and then find a path connecting them. Following this path can be

regarded as executing the plan [65]. The familiar problem with this brute-force approach to representing the effects of actions is that the number of possible situations grows exponentially as the number of attributes increases or their domains become larger. This problem can make representing a graph very expensive or even impossible.

At the other end of the spectrum one can represent the situations and the transitions among them using first order logic formulas. [57] suggests that logic is a an appropriate way of representing knowledge. Suppose do(A, S) means performing action A in situation S, with the result being another situation. As an example, a statement like has(O, do(pick(O), S) would then mean that if in any situation the agent picks up an object O, then it has the object O in the next situation.

This method of representation allows us to generalise across attribute values because the statement is true for many values of O and S. It also generalises across attributes themselves, because the statement holds irrespective of what other attributes (other than O and S) hold at the time. This form of representation has been used for programming purposes. For example, in [49] rules extracted in a situation calculus domain are considered as logic programmes. In GOLOG [47] which is a programming language based on Situation Calculus, the programmer writes code to specify the initial state of the environment, the preconditions and the effects of actions. This approach results in efficient representations of the domain.

Our aim is to simplify the process of writing plans. We attempt to do this by deriving the rules of the environment automatically and then using them as parts of a plan generator [40].

4.3.2 Generating Prolog Statements

Each individual rule is concerned with learning the immediate effects of individual actions. After knowing these effects, we can combine the actions in the form of a plan and come up with more complex behaviours. Most other work on planning starts at this point, because they consider the problem of knowing the effects of actions as already solved.

The rules created by TimeSleuth can easily be represented as Prolog statements. The c4.5rules programme in the C4.5 package generates rules from a decision tree. With TimeSleuth this program is modified [34] to optionally generate its rules in Prolog. When the user gives the command line option of '-p 0', or an option in TimeSleuth's user interface is selected, a <file stem>.pl Prolog file will be created in addition to the normal output. The generated Prolog statements are in the Edinburgh dialect and can be fed to most Prolog interpreters without change.

There is a problem in using Prolog to represent temporal rules: there is no concept of time in standard Prolog. But as explained later, using Prolog to represent the rules is especially appropriate in temporal domains, where the decision attribute is actually one of the condition attributes, seen at a later time. A temporal order is implicitly present in Prolog because it follows a rule's conditions from left to right.

So the problem disappears when we set the window size to 2 and perform temporalisation in the forward direction, as the condition attributes will all be coming from the previous time step relative to the decision attribute. The following discussion will concern data generated from the artificial creature in URAl, where X referes to the robot's position along the x axis, and A refers to the action (direction of movement). Table

4.1 shows parts of an example set of statements generated by TimeSleuth when the decision attribute is x_2 (The position of the robot at time step 2).

Table 4.1 Three sample Prolog statements generated by TimeSleuth

class(A1, X1, 0) :- A1 = 2, X1 = 1.
class(A1, X1, 2) :- A1 = 3, X1 = 1.
class(A1, X1, 3) :- A1 = 2, X1 = 4.

In Table 4.1 a value of 2 and 3 for action *A1* could mean going to the left and right, respectively. Following a classification terminology, the results are designated by a predicate called "class." The condition attributes (action *A1* and position *X1* in this case) come first, and the value of the decision attribute (the next value of *x*) comes last. In the head of the rules, the condition attributes are used for the decision making process. In our example temporal data, *A1* and *X1* belong to the current time step, while the classification is done for the value of *x* in the next time step.

To use such rules the user can issue queries like class(2, 4, X2) (where does the creature go from $x = 4$ if it moves Left?). If we are dealing with more than one sensor attribute (*x* and *y* for example) we could rename "class" to something like "classx" to avoid name clashes.

Notice that the automatically generated Prolog statements use the unification operator (=) instead of the comparison operator (=:=). This allows the user to traverse the rules backward and go from the decision attribute to the condition attributes, or from a set of decision and condition attributes, to the remaining condition attributes. Some example queries are class(A1, 1, 2) (which actions take the creature from $x = 1$ to $x = 2$?) or class (A1, X1, 3) (which action/location pairs lead to $x = 3$?). This makes C4.5's discovered rules generally more useful.

TimeSleuth can generate rules that rely on threshold testing and set membership testing. If we use the standard Prolog operators of =< and > for threshold testing, and implement a simple member() function for testing set membership, then we would not be able to traverse the rules backward, as they lack the ability to unify attributes. So if we had a clause like: class(A, B, C) :- A =< B, member(A, C), then we would be unable to use a query like class(A, 3, [1, 2, 3]), because Prolog can not perform the test =< on attributes that are not unified.

Adding the unification ability to =<, > and member() will remove this limitation. For example, $X > 10$ would choose a value above 10 for X if it is not already unified, and member(X, [1, 2, 3]) would unify X with one of 1, 2, or 3 if it is not already unified. Both cases would always succeed if X is not unified, but could fail if it is. We have written some simple code to do just this and the results are shown in Table 4.2. We employed a deterministic method to choose the value of the attribute that is going to be unified, but one could use a random method too. ule (unify less-equal), ug (unify greater) and umember() (unify member) are the unifying counter parts of =<, > and member(), respectively.

Table 4.2 Prolog operators and functions for planning

Name	Unifies?	Implementation
=<	No	Standard
>	No	Standard
member()	No	member(*A*, [*A*|_]). member(*A*, [_|*B*]) :- member(*A*, *B*).
ule	Yes	:- op(800, xfx, ule). *A* ule *B* :- var(*A*), *A* = *B*. *A* ule *B* :- *A* =< *B*.
ug	Yes	:- op(800, xfx, ug). *A* ug *B* :- var(*A*), *A* is *B* + 1. *A* ug *B* :- *A* > *B*.
umember()	Yes	umember(*A*, *B*) :- var(*A*), [*X*|_] = *B*, *A* = *X*. umember(*A*, *B*) :- member(*A*, *B*).

If the argument to the left of ule is not unified, ule sets its value to be the same as its argument to the right, and returns with success. If the left hand side argument is already unified, then it does a =< test. The ug operator does the same with regard to >. If unification is needed, it sets the left hand side argument to the value of the right hand side argument plus 1. In both cases the right hand side argument should already have been unified. The function umember() unifies the first argument with the first member of the list if unification is needed. The second argument to this function should have already been unified. These conditions seem to hold for rules generated by TimeSleuth, so it can generate rules of the form class(A, B, C, 0) :- A = 1 , B ug 10, umember(C, [1, 2, 3]). As explained below, this unification ability has another advantage: it allows us to generate plans by backward chaining in the rules.

To create a Prolog plan generator, we have to make manual modifications in the Prolog statements generated by TimeSleuth. The results will be a set of rules that search backward from a desired situation to the current situation, and if such a path is found, we prints the actions that have to be performed to get to the desired situation. The modifications should be done manually because we have made the changes to c4.5rules in a general manner and compatible with the normal output of C4.5. The Prolog clauses that are generated simply do a normal classification without caring about any "bigger picture" that may exist in a particular application such as planning. We now go over the modifications needed for planning. Suppose we start with the rule: class(*A1*, *X1*, 0) :- *A1* = 2, *X1* = 1.

We have to make sure that Prolog does not get stuck in plans with cycles, where the robot visits the same position unendingly. To prevent this problem we keep track of the classes we have already visited. This is done by adding a list attribute to the class() clause, and adding code for cycle-prevention. So now we have this rule: class(*A1*, *X1*, 0, P_1) :- *A1* = 2, *X1* = 1, not(umember(*X1*, P_1)), P_2 = [*X1*|P_1].

X1 is used to distinguish among the steps in the plan. If we do not find the value of *X1* in our list, then we know we are not in a cycle. If this holds we add it to the list P_1 to get a new list P_2, and continue from there.

In a temporal domain one of the condition attributes may actually be the previous value of our decision attribute. In this example *x* has this property. We now have to make this fact explicit, because we want Prolog to make sure that we are actually in the previous position before advancing to the next one. To do this we introduce the class() clause in the condition part of the Prolog statements, which will allow Prolog to use recursion and try all the paths that lead from one class to the next. So we now have: class(*A1*, *X1*, 0, P_1) :- *A1* = 2, *X1* = 1, not(umember(*X1*, P_1)), P_2 = [*X1*|P_1], class(_, _, 1, P_2).

Notice that the value 1 in class(_, _, 1, P_2) comes from *X1* = 1, as in this example class() is C4.5's name for the *x* position. Now Prolog can search for a way to get the robot from the starting state to a desired state.

We can add a statement to print the plan after it is generated. This is simple to do: class(*A1*, *X1*, 0, P_1) :- *A1*= 2, *X1* = 1, not(umember(*X1*, P_1)), P_2 = [*X1*|P_1], class(_, _, 1, P_2), printOut(*A1*, *X1*).

Finally, we introduce the current position as a class statement. For example, if we are currently at *X1* = 0, we add the clause class(_,_,0,_) to the set of rules. Intuitively this means we are currently at position 0, and we do not care how we got here.

Automating the above steps requires c4.5rules to know the previous value of the decision attribute. Conveying this information to c4.5rules is not very easy, so the transformation is done manually.

Performing the above steps on the rules in Table 4.1 gives us the rules in Table 4.3. The helper functions not() and printOut() are also provided. The function umember() has already appeared in Table 4.2.

Table 4.3 Prolog rules modified for planning

class(_, _, 0, _).
class(*A1*, *X1*, 0, P_1) :- *A1* = 2, *X1* = 1, not(umember(*X1*, P_1)), P_2 = [*X1*|P_1], class(_, _, 1, P_2), printOut(*A1*, *X1*).
class(*A1*, *X1*, 2, P_1) :- *A1* = 3, *X1* = 1, not(umember(*X1*, P_1)), P_2 = [*X1*|P_1], class(_, _, 1, P_2), printOut(*A1*, *X1*).
class(*A1*, *X1*, 3, P_1) :- *A1* = 2, *X1* = 4, not(umember(*X1*, P_1)), P_2 = [*X1*|P_1], class(_, _, 4, P_2), printOut(*A1*, *X1*).
not(*G*) :- *G*, !, fail. not(*G*).
printOut(*A*, *X*) :- write('Robot is at: '), write(*X*), write(', it does action: '), write(*A*), nl.

Using the above statements, the user can perform Prolog queries of the form class(_, _, 7, []) to find a plan for going to position $x = 7$. He can include a position in the last argument of his query to prevent that position from showing up in the plan, as anything in that list will be avoided. Prolog will then print out the actions that should be performed to get from the current position to the desired position.

C4.5 assigns a certainty value to each rule it generates, which shows how reliable that rule is. TimeSleuth can optionally include certainty information in the generated Prolog rules. Standard Prolog does not support the notion of reliability of a statement. To add

this information to the Prolog statements in a way that would be understandable to most Prolog systems, we use a pseudo random number generator to cause a rule to fail in proportion to its certainty value. A random integer is generated and tested against the certainty value. A statement can fail if this test fails, no matter what the value of the condition attributes. C4.5 computes the certainty values as a number less than 1 and outputs them with a precision of 0.001. TimeSleuth multiplies this number by 1000 to convert it to an integer. It outputs the necessary code to handle the certainty value if the user invokes it with a '-p 1' command line argument, or the appropriate option from TimeSleuth's user interface.

We used example Prolog statements generated from the Letter Recognition Database from University of California at Irvine's Machine Learning Repository [8] to illustrate the certainty values (URAL data would result in very high certainty values). This database consists of 20,000 records that use 16 condition attributes to classify the 26 letters of the English alphabet. The 16 attributes are named A1 to A16. The decision attribute encodes the index of the letters. The results, showing some statements for the letter "I," are shown in Table 4.4.

Table 4.4 Prolog statements with certainty values

class(A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16, 8) :- random(1000, N__), N__ < 793, A10 = 8, A12 = 5, A13 = 3, A14 = 8,
class(A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16, 8) :- random(1000, N__), N__ < 793, A7 = 9, A13 = 0, A14 = 9, A16 = 7.
class(A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16, 8) :- random(1000, N__), N__ < 707, A6 = 9, A10 = 7, A11 = 6, A13 = 0.

All condition attributes are represented in the left-hand side of the Prolog statements, which allows Prolog to distinguish among the condition attributes by using their position, so the user can specify the attributes unambiguously. In Table 4.4, the first rule has a

certainty value of 79.3%. The random(1000, N__) function assigns a number between 0 and 999 to N__. This value is then compared to the certainty value of the rule, which is 793. The statement could fail based on the results of the comparison. The random number is named N__ to lessen the chances of an accidental clash with the name of a condition attribute. One could implement the random number generator as follows [15]:

```
seed(13).
random(R, N) :- seed(S), N is (S mod R), retract(seed(S)),
        NewSeed is (125 * S + 1) mod 4096, asserta(seed(NewSeed)), !.
```

We have taken an active approach to representing the certainty values, because they can actually cause the statements to fail if the random number is bigger than the certainty value. In an alternate implementation, we could choose to simply output these values as part of the statements and leave any specific usage to the user. An example, taken from the last statement in Table 4.4, would be class(A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16, N__, 8) :- A6 = 9, A10 = 7, A11 = 6, A13 = 0, N__ = 707.

Chapter 5

Experimental Results

In this chapter, we present the results of comparisons and experiments done with TIMERS. First we will perform a comparison of TIMERS with other causality discoverers and see that they yield some incorrect results. Next we perform a series of experiments on TIMERS to show its behaviour on synthetic and real data.

In Section 5.1 we compare TIMERS to CaMML and TETRAD. Causality has often been applied to complex domain such as sociology, where the subject of what are the causes and what are the effects is open to debate, so claims of discovering causality are hard to verify. In this section we use the well-defined domain of the artificial robot as a sanity check for these methods, because we know all the rules governing it, and as a consequence we can analyse the results will little ambiguously.

In Section 5.2 we provide the results of further experiments with TIMERS on the robot data, as well as a real world database containing weather observations and a moving robot database. We also provide the results of experiments on a spatial database that contains samples taken every half a metre from an oil well.

5.1 Comparison with Other Approaches to Causal Discovery

The purpose of the experiments in this section is to measure the effectiveness of TimeSleuth, TETRAD, and CaMML in discovering rules when provided with sequential data. First we apply the methods to non-temporalised data, which has information for discovering atemporal rules. Then we apply the methods to data that is temporalised with various window sizes, which should allow them to discover temporal rules. The results show how temporalisation affects the output of the methods.

5.1.1 The Problem and the Desired Output

We apply the three methods to example dataset from the URAL programme. This dataset is created by a robot that is performing a random walk in a two-dimensional space. Each record in this dataset contains x and y position values at any given time, the direction of movement at that time, and also a binary attribute indicating the presence or absence of food. TETRAD restricts attributes to at most 8 values, so the size of the world was set to be 8×8, with x and y values ranging from 0 to 7. To ensure fairness to all methods, all results presented in this section are derived from a single run of 10,000 time steps in URAL where food existed at locations (0,6), (1,2), (3,5), (5,5), and (6,3). The location of food would not change during the run. These results are representative of many experiments we have performed with varying number of time steps and a variety of locations for food.

To be able to better pass judgment on the results, Table 5.1 shows the desired output of TimeSleuth, TETRAD, and CaMML in each tool's notation. Given our knowledge of

the domain, our expected result is provided under the column Domain Expert and the following columns give the desired output in the notation of the tools used. TimeSleuth's output is represented by rules because rules are C4.5's output. *FoodXY*() is an atemporal relation saying whether or not food is present at a given (*x*,*y*) location, while *MoveX*() and *MoveY*() are temporal relations describing the effects of moving along the *x*-axis and *y*-axis, respectively.

Table 5.1 The desired output for TimeSleuth, TETRAD, and CaMML

Window Size	Domain Expert	TimeSleuth	TETRAD	CaMML
1	$f = FoodXY(x, y)$	if $\{(x = \alpha)$ & $(y = \beta)\}$ then $(f = \delta)$	$x \rightarrow f$, $y \rightarrow f$	$(x, y \rightarrow f)$
	No relation for *a, x, y*	No rules (low accuracy)	*a, x, y*	$(\rightarrow a)$, $(\rightarrow x)$, $(\rightarrow y)$
$w \geq 2$	$f_w = FoodXY(x_w, y_w)$	if $\{(x_w = \alpha)$ and $(y_w = \beta)\}$ then $(f_t = \delta)$	$x \rightarrow f_w$,, $y \rightarrow f_w$	$(x_w, y_w \rightarrow f_w)$
	$x_w = MoveX(x_{w-1}, a_{tw1})$	if $\{(x_{w-1} = \alpha)$ and $(a_{w-1} = \gamma)\}$ then $(x_w = \delta)$	$x_{t-1} \rightarrow x_w$, $a_{t-1} \rightarrow x_w$	$(x_{w-1}, a_{w-1} \rightarrow x_w)$
	$y_w = MoveY(y_{w-1}, a_{w-1})$	if $\{(y_{w-1} = \alpha)$ and $(a_{w-1} = \gamma)\}$ then $(y_w = \delta)$	$y_{t-1} \rightarrow y_w$, $a_{t-1} \rightarrow y_w$	$(y_{w-1}, a_{w-1} \rightarrow y_w)$
	No cause for a_t	No rules (low accuracy)	a_t	$(\rightarrow a_w)$

For any window size $w \geq 2$, the result of a move depends only on the position and the move direction in the w-1 time step (time starts at 1 and ends at w in each record that is temporalised in the forward direction). We expect the atemporal relations that hold for $w = 1$ to also hold for $w > 1$ because atemporal relations do not depend on time.

In Table 5.1, TETRAD's and CaMML's desired output are represented directly in their respective notations. TimeSleuth's desired output is given in template form, as generalised rules. The actual output has multiple rules, with one rule for each combination of the values for the attributes representing conditional attributes. Also, the actual output does not contain keywords such as *if*, *then*, or *and*, and has specific values instead of the α, β, and δ parameters.

5.1.2 Results with a Window Size of 1

To test their ability to find atemporal relations, we ran TETRAD 4.3.3 [93], CaMML and TimeSleuth on non-temporalised data. We consider the rules $x \text{ o}\!\!\rightarrow f$, $y \text{ o}\!\!\rightarrow f$ to be incorrect because they imply the wrong direction. We note that by design TETRAD cannot find rules of the form $(x, y \rightarrow f)$, because it only searches for relationships between two attributes at a time.

TETRAD can be run with several options. If the existence of latent common causes is assumed, TETRAD uses the FCI algorithm, and otherwise if causal sufficiency is assumed, TETRAD uses the PC algorithm [71]. The PC algorithm has been proved to be correct if the Markov and Faithfulness assumptions hold in the data [71]. The results for TETRAD with the FCI algorithm are shown in Table 5.2.

Table 5.2 Rules discovered by TETRAD's FCI algorithm (w = 1)

Significance Levels	Correct Rules	Incorrect Rules
0.0	a	$x \text{ o}\!\!\rightarrow f$, $y \text{ o}\!\!\rightarrow f$
0.001, 0.01, 0.05, 0.1, 0.2	a	$y \text{ o}\!\!-\!\!\text{o } f$, $y \text{ o}\!\!-\!\!\text{o } x$, $a \text{ o}\!\!-\!\!\text{o } x$

In Table 5.3 we provide TETRAD's results with the PC algorithm. In a some cases PC's results are less conclusive than FCI's results. For example, $y - f$ is undirected and could mean any of $y \rightarrow f$, $y \text{ o}\!\!-\!\!\text{o } f$, $y \text{ o}\!\!\rightarrow f$, $f \rightarrow y$, etc. TETRAD's best results are obtained with the PC algorithm and a significance level of 0.0.

Table 5.3 TETRAD's rules with the PC algorithm (w =1)

Significance Levels	Correct Rules	Incorrect Rules
0.0	$a, y \rightarrow f, x \rightarrow f$	
0.001, 0.01, 0.05, 0.1, 0.2	a	$x - f$, $y - f$, $x - y$

For the same data, CaMML gave the results shown in Table 5.4. CaMML found all the correct atemporal rules. In reality, x and y do not cause f in URAL, but without any domain knowledge, it is reasonable to interpret this relationship as a causal one, so we accept the rule $(x, y \rightarrow f)$ as correct.

Table 5.4 CaMML's results with non-temporalised records ($w = 1$)

Correct Rules	Incorrect Rules
$(\rightarrow a)$, $(\rightarrow x)$, $(\rightarrow y)$, $(x, y \rightarrow f)$	

We ran TimeSleuth with the same data, with options set to try all attributes as the decision attribute. The results are shown in Table 5.5. The first two entries for predicting the value of f show that c4.5rules eliminated unneeded condition attributes when creating rules. In general, if the value of x is sufficient to predict the outcome regardless of the value y, the generated rule will not include y and vice versa. We accept all the rules generated for predicting f as correct, because together they correctly predict the presence or absence of food.

As a classifier, C4.5 creates rules regardless of whether or not they make semantic sense, but none the less the results were interesting. There is a strong association between f, x, and y, and this is exploited by TimeSleuth for predicting the value of these attributes. The value of a, however, is unrelated to any other observable attribute, and this is reflected in low accuracy value for the rules. The low accuracy rules are discarded because they fall below any threshold value of 35% or more, including 80%, 90%, 95%, 99.5%, and 99.99% that correspond to TETRAD's significance levels.

Table 5.5 TimeSleuth's results with non-temporalised records ($w = 1$)

Decision Attribute	Condition Attribute(s)	Number of Rules	Correctness	Training Accuracy
f	x	3	Correct	100%
	y	4		
	x, y	20		
x	y, f	4	Incorrect	19.9%
y	x, f	5	Incorrect	21.2%
a	x, y	7	Incorrect	26.4%

In TIMERS, rulesets with low accuracy values are considered to imply the absence of a reliable relationship between the involved attributes.

5.1.3 Results with Larger Window Sizes

Window size 2: Temporalising using a window size of $w = 2$, produced records with 8 attributes $\{x_1, y_1, f_1, a_1, x_2, y_2, f_2, a_2\}$. To test the discriminating powers of these methods, we included all eight attributes in the tests, including the ones that appear at the same time as the decision attribute. In the experiments reported in Section 5.2, we will only use the decision attribute at the current time.

TETRAD, CaMML, and TimeSleuth were applied to determine whether they could discover the *FoodXY*() atemporal function and the *MoveX*() and *MoveY*() temporal functions.

TETRAD's output with the FCI algorithm (assuming the existence of latent causes) is summarised in Table 5.6. TETRAD allows the user to specify any temporal order among the input attributes, and we provided the information that attributes from one time step have a temporal precedence from the attributes of the following time step. With FCI, temporalising the records resulted in many more rules being discovered than with window size 1. Most of these rules are incorrect.

Table 5.6 TETRAD's FCI algorithm with temporalised records ($w = 2$)

Significance Levels	Correct Rules	Incorrect Rules
0.0	a_2	$y_1 \circ\!\!\rightarrow y_2, f_1, a_1 \rightarrow y_2, a_1 \circ\!\!\rightarrow x_2, x_1 \circ\!\!\rightarrow x_2, f_2$
0.001, 0.01, 0.05, 0.1, 0.2	a_2	$y_1 \circ\!\!\rightarrow f_1, y_1 \circ\!\!\rightarrow y_2, a_1 \circ\!\!\rightarrow x_2, a_1 \circ\!\!\rightarrow y_2, x_1 \circ\!\!\rightarrow f_1, x_1 \circ\!\!\rightarrow x_2, f_2$

TETRAD's output assuming no latent common causes (the PC algorithm) appears in Table 5.7. As can be seen, PC also discovered more incorrect rules after temporalisation.

Table 5.7 TETRAD's PC algorithm with temporalised records ($w = 2$)

Significance Levels	Correct Rules	Incorrect Rules
0.0	$y_1 \rightarrow f_1, y_1 \rightarrow y_2, a_1 \rightarrow x_2, a_1 \rightarrow y_2, a_2, x_2 \rightarrow f_2$	$f_1 \rightarrow x_2, f_1 \leftrightarrow x_2, x_2 \rightarrow x_1$
0.001, 0.01, 0.05, 0.1	$y_1 \rightarrow y_2, a_1 \rightarrow x_2, a_1 \rightarrow y_2, a_2, x_2 \rightarrow f_2$	$y_1 - f_1, y_1 \rightarrow x_2, f_1 \rightarrow x_2, f_1 \rightarrow x_2, x_2 \rightarrow x_1$
0.2	$y_1 \rightarrow y_2, a_1 \rightarrow x2, a_1 \rightarrow y_2, x_2 \rightarrow f_2, a_2$	$f_1 \rightarrow y_1, f_1 \rightarrow x_2, f_1 \rightarrow x_1, x_2 \rightarrow x_1$

The results of applying CaMML with a window size of 2 appear in Table 5.8. For CaMML, we were unable to discover any means of specifying a temporal order among the input attributes. It continued to discover the same relations as it had found with non-temporalised records. These relations are correct, because the relations that existed in the previous case continue to exist in the temporalised data. However, CaMML failed to discover the relationships between the previous location and action, and the current location. The correct relationships might have been expressed as (x_1, $a_1 \rightarrow x_2$) and (y_1, $a_1 \rightarrow y_2$), but they are absent in the output. CaMML discovered the single rule (x_1, y_1, x_2, $y_2 \rightarrow a_1$), which represents the same information in an alternative format. However, this might be considered to be temporally invalid, because it refers to the values of variables in the future to predict the past. In fairness to CaMML, the temporal information was not available to it.

Table 5.8 CaMML's rules with temporalised records ($w = 2$)

Correct Rules	Incorrect rules
$(\rightarrow x_1)$, $(\rightarrow y_1)$, $(x_1, y_1 \rightarrow f_1)$, $(x_2, y_2 \rightarrow f_2)$, $(\rightarrow a_2)$	$(x_1, y_1, x_2, y_2 \rightarrow a_1)$, $(\rightarrow x_2)$, $(\rightarrow y_2)$

The results of applying TimeSleuth to the same data are given in Table 5.9. In our 8×8 board, 4 possible actions and 8 distinct values exist for each of x_1 and y_1. In this example, the agent has explored the entire world, and TimeSleuth created 32 (8×4) rules for predicting the next value of each of x_2 or y_2. It correctly pruned y_1 from the rules for x_2, because the rules for moving along the x-axis are independent of the value of y. Similarly, it pruned x_1 from the rules for y_2.

The rules for predicting f_2 are the same as the rules given in Table 5.1 for predicting f in the atemporal case, which is correct, because the rules for food are not dependent on time. TimeSleuth was also tried on the a_2 attribute, which is not caused by any of the observable attributes. We have not shown the results with x_1, y_1, f_1, and a_1 as decision attributes. Consistent with the results with non-temporalised data, those rules have low accuracy values.

Table 5.9 TimeSleuth's results with temporalised records ($w = 2$)

Decision Attribute	Condition Attribute(s)	Number of Rules	Correctness	Training Accuracy
f_2	x_2	3	Correct	100%
	y_2	4		
	x_2, y_2	20		
x_2	x_1, a_1	32	Correct	100%
y_2	y_1, a_1	32	Correct	100%
a_2	x_1, a_1, y_2	16	Incorrect	29.8%
	$x_1, y_1, a_1,$	18		
	x_1, y_1, y_2	6		
	x_1, x_2, y_2	7		
	x_1, y_1, x_2	4		
	x_1, y_2	3		
	x_1, y_1	2		
	x_1, a_1	1		
	a_1, y_2	1		

Window Sizes Larger Than 2: TETRAD's FCI and PC algorithms were tried with $w = 3$. The results are shown in Table 5.10 for FCI and in Table 5.11 for PC.

Table 5.10 TETRAD's FCI algorithm with temporalised records ($w = 3$)

Significance Levels	Correct Rules	Incorrect Rules
0.0	a_3	$y_1 \circ\!\!\rightarrow y2, f_1, a_1 \circ\!\!\rightarrow y_2, a_1 \circ\!\!\rightarrow x_2, x_1 \circ\!\!\rightarrow x_2, f_2,$ $a_2 \circ\!\!\rightarrow y_3, a_2 \circ\!\!\rightarrow x_3, f_3 \circ\!\!\rightarrow y_3, x_3 \circ\!\!\rightarrow a_2$
0.001, 0.01, 0.05, 0.1, 0.2	a_3	$y_1 \circ\!\!\rightarrow f_1, y_1 \circ\!\!\rightarrow y_2, a_1 \circ\!\!\rightarrow y_2, a_1 \circ\!\!\rightarrow x_2, x_1 \circ\!\!\rightarrow f_1,$ $x_1 \circ\!\!\rightarrow x_2, f_2, a_2 \circ\!\!-\!\!\circ x_3, a_2 \circ\!\!\rightarrow y_3, f_3 \circ\!\!\rightarrow y_3$

Table 5.11 TETRAD's PC algorithm with temporalised records ($w = 3$)

Significance Levels	Correct Rules	Incorrect Rules
0.0	$y_1 \rightarrow y_2, x_1 \rightarrow x_2, y_1 \rightarrow f_1, a_1 \rightarrow y_2,$ $a_1 \rightarrow x_2, x_1 \rightarrow f_1, a_2 \rightarrow x_3, a_2 \rightarrow y_3, a_3$	$f_2,\ y_3 \leftrightarrow f_3, f_3 \leftrightarrow x_3$
0.001	$y_1 \rightarrow y_2, a_1 \rightarrow y_2, a_1 \rightarrow x_2,$ $x_1 \rightarrow x_2, a_2 \rightarrow x_3, a_2 \rightarrow y_2, a_3$	$y_1 - f_1, y_1 - x_1, f_1 - x_1, f_2,$ $y_3 \leftrightarrow f_3, x_3 \leftrightarrow f_3$
0.01, 0.5, 0.1, 0.2	$y_1 \rightarrow y_2, a_1 \rightarrow y_2, a_1 \rightarrow x_2,$ $x_1 \rightarrow x_2, a_2 \rightarrow x_3, a_2 \rightarrow y_3, a_3, x_3 \rightarrow f_3$	$y_1 - f_1, y_1 - x_1, f_1 - x_1, f_2,$ $y_3 \leftrightarrow f_3,$

In all experiments performed with TETRAD, the PC algorithm clearly gave better results than the FCI algorithm, implying that in the test data the assumption of causal

sufficiency holds. PC's results were generally better when the significance level was set to 0.0.

TimeSleuth obtained good results with larger window sizes. The causal rules concerning x_w and y_w all have an accuracy value of 100%. As demonstrated in [33], C4.5 effectively handles the bigger input records that are created by larger window sizes by pruning irrelevant attributes. More experiments with TimeSleuth on the robot database are given in Section 5.2. Table 5.12 summarises the results obtained with window sizes from 3 to 10.

Table 5.12 Test results for larger window sizes

Window Size	Domain Expert	TimeSleuth	TETRAD	CaMML
$3 \le w \le 10$	$FoodXY(x_w, y_w)$	if $\{(x_w = \alpha)$ and $(y_w = \beta)\}$ then $(f_w = \delta)$.	Better results with the PC algorithm.	Not tested because it cannot handle the input size
	$MoveX(x_{w-1}, a_{w-1})$	if $\{(x_{w-1} = \alpha)$ and $(a_{w-1} = \gamma)\}$ then $(x_w = \delta)$.		
	$MoveY(y_{w-1}, a_{w-1})$	if $\{(y_{w-1} = \alpha)$ and $(a_{w-1} = \gamma)\}$ then $(y_w = \delta)$.		

5.1.4 Summary of Comparisons

The results of our experiments are summarised in Table 5.13, where the number of correct and incorrect relations/rules discovered by each system, for a variety of window sizes, is given. For TETRAD, we consider the PC results with a 0.0 significance level and for TimeSleuth we assumed a confidence threshold of 95%.

Table 5.13 Summary of experimental results

Window Size	TETRAD		CaMML		TimeSleuth	
	Correct	Incorrect	Correct	Incorrect	Correct	Incorrect
$w = 1$	3	0	4	0	27	0
$w = 2$	6	3	5	3	91	0
$w = 3$	9	3	N/A		91	0

TIMERS performs well on this problem. CaMML was not able to use any temporal information. Both TETRAD and CaMML found some wrong relations as the window size was increased.

5.2. Evaluation of TIMERS on Synthetic and Real Data

In the previous section we used temporalisation strictly in a forward direction, so we could perform comparisons with other software. In this section, we employ TIMERS' more flexible treatment of time. We also assume that the domain expert can choose a decision attribute, so we concentrate on specific attributes instead of trying all of them..

We first use two temporal datasets to test TIMERS' ability in discovering casual and acausal rules. The first dataset is from URAL and the second dataset is from a weather station in Louisiana AgriClimatic Information System [88]. To show that TIMERS can be employed independently of the underlying rule discovery method, we employ both classification and regression to generate rules.

Later in this section we discover rules from another real-world temporal dataset that contains observations regarding the detection of failure in a robot that grabs and moves objects. In the last set of experiments we show how TIMERS can be applied to a spatial dataset.

5.2.1 Using Classification to Generate Rules

The Artificial Robot

We used 2500 records for training, and 500 for testing the rules (predictive accuracy). The decision attribute is set to be the current value of x, and the other three attributes (y, f,

and *a*) are set as the condition attributes. There is no relationship between the current value of *x* on one hand, and the current values of *y*, direction of the movement, or the presence of food on the other hand. So we predict that an instantaneous test (window size of 1) will give poor results. From our understanding of the domain we know that the current value of *x* depends on the previous value of *x*, and the previous direction of movement (the same holds for *y*). We expect the method to classify the relationship as a causal one. The acausal hypothesis says that you can tell where you were before if you know where you are now. This hypothesis is clearly wrong, as we could have ended at the current position from a different number of previous positions. Hence we do not expect to get good results with our acausality test. The results are shown in Table 5.14, where "T Accuracy" stands for training accuracy, and "P Accuracy" stands for predictive accuracy. The "Position" column shows the index of the decision attribute within the temporalised record.

Table 5.14 TIMERS' results with the robot data. Verdict is causal

Window	Position	T Accuracy	P Accuracy	Type of test	Actual rules
1	1	19.7%	20.4%	Instantaneous	Instantaneous
2	1	56.2	55.7%	Acausal	Acausal
2	2	100%	100%	Causal	Causal
3	1	57.6%	55.6%	Acausal	Acausal
3	2	100%	100%	Acausal	Causal
3	3	100%	100%	Causal	Causal
4	1	58.4%	58.1%	Acausal	Acausal
4	2	100%	100%	Acausal	Causal
4	3	100%	100%	Acausal	Causal
4	4	100%	100%	Causal	Causal
5	1	58.4%	57.1%	Acausal	Acausal
5	2	100%	100%	Acausal	Causal
5	3	100%	100%	Acausal	Causal
5	4	100%	100%	Acausal	Causal
5	5	100%	100%	Causal	Causal

Considering the 100% accuracy values with a window size of 2 or bigger in the causal tests, TIMERS declares the relation to be causal. With any position bigger than 1, the previous record which contains the relevant information for an accurate prediction of current *x* value, is included in the temporalised data, and TIMERS discovers the correct temporal relation between the current value of *x* and the previous *x* and movement direction. In other words, even with an acausality test, the rules are all causal because they only contain attributes from the previous time step.

An example rule would be: if {(x_t = 1) AND (a_t = Right)} then (x_{t+1} = 2). The equivalent Prolog rule generated by TimeSleuth is: class(Y_t1, F_t1, originalDecision_t1, X_t1, 2) :- OriginalDecision_t1 = 3, X_t1 = 1. originalDecision is the default name given to the decision attribute in the input file (in this case it is the direction of movement) as opposed to the decision attribute chosen by the user (in this case we had chosen *x*). 3 is the code for moving to the Right. *Y* and *F* do not appear among the condition attributes, which means that their values are not important in this rule. The last argument in the prolog rule (2 in this case) is the decision attribute's next value. This rule indicates that no matter what the previous *y* location and the presence or absence of food, to go to the *x* location 2 in the next time step, you can be in *x* location 1 and move to the Right. As explained in Chapter 4, a series of such rules can be combined to automatically come up with a plan for moving from a starting location to a destination. The whole Prolog file is presented in Appendix 1.

We now create a dependence diagram for the robot data.

Step 1: building the diagram. The diagram is as shown in Figure 4.14.

Step 2: Pruning. We remove all links with strength less than or equal to 80%, and remove all nodes with strength less than 50%. As it turns out, no nodes are removed. Results are shown in Figure 5.1 below.

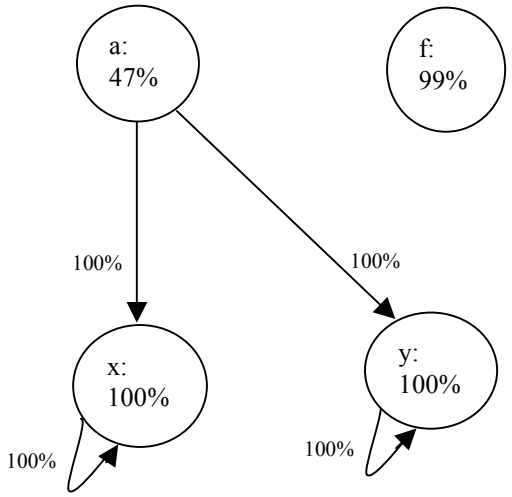


Figure 5.1 The pruned dependence diagram for the robot data

We can observe the influence of the attributes on each other in this dependence diagram, and thus have a pictorial summary of the rules. If there is a link from a node to itself, then we know that the reference means that the value of the corresponding attribute at a different time (past of future) is used for predicting the present value. This implies that we see a node pointing to itself only when the window size is bigger than 1.

Compared to the experiments of the previous chapter, it seems that the rules for predicting the value of *f* have changed (*x* and *y* are not pointing to *f* as one would expect from the results of Section 5.1). The reason is that in the experiments of this section TIMERS is not including the *x* and *y* values that are observed at the same time as *f*. To re-create the atemporal rules for *f* in Section 5.1, one should perform the test with a widow size of 1.

We do not claim that a dependence diagram necessarily implies the existence of causality, because the rules that are used as the basis for the diagram may have come from an instantaneous or acausal investigation. Any interpretation of causal relationships derived from a dependence diagram is left to the domain expert.

The weather data

The subject of experiments in this subsection is a real-world dataset from weather observations in Louisiana, and hence interpreting the dependencies and relationships is harder than the robot dataset. It contains observations of 8 environmental attributes gathered hourly from 22/7/2001 to 6/8/2001. There are 343 training records, each with the air temperature, the soil temperature, humidity, wind speed and direction and solar radiation, gathered hourly. 38 other records were used for testing the rules and generating predictive accuracy values. We have set the soil temperature to be the decision attribute. The results obtained are shown in Table 5.15.

Table 5.15 TIMERS' results the weather data. Verdict is acausal

Window	Position	T Accuracy	P Accuracy	Type of test	Actual rules
1	1	27.7%	23.7%	Instantaneous	Instantaneous
2	1	75.1%	59.5%	Acausal	Acausal
2	2	82.7%	67.6%	Causal	Causal
3	1	85.3%	75.0%	Acausal	Acausal
3	2	82.4%	72.7%	Acausal	Acausal
3	3	86.8%	77.8%	Causal	Causal
4	1	85.3%	74.3%	Acausal	Acausal
4	2	85.9%	74.3%	Acausal	Acausal
4	3	83.2%	74.3%	Acausal	Acausal
4	4	84.4%	71.4%	Causal	Causal
5	1	85.0%	73.5%	Acausal	Acausal
5	2	87.0%	76.5%	Acausal	Acausal
5	3	85.0%	76.5%	Acausal	Acausal
5	4	83.8%	76.5%	Acausal	Acausal
5	5	86.7%	73.5%	Causal	Causal

Because the accuracy values in the two directions of time are close, TIMERS declares the relationship between the soil temperature and other attributes to be acausal. The relationship is not instantaneous, as observed by relatively poor results with a window size of 1 (instantaneous test). The accuracy goes up after temporalisation, implying that there is a temporal relationship at work.

We now create a dependence diagram for the weather data created from a causal investigation with window size 2. We use a minimum strength value of 40% for pruning the edges, and a minimum strength value of 50% for pruning the nodes.

Step 1. Building the diagram. To save space and avoid displaying many links, we show the diagram after pruning the links, so only links with strength values more than 40% are shown. Figure 5.2 shows the results.

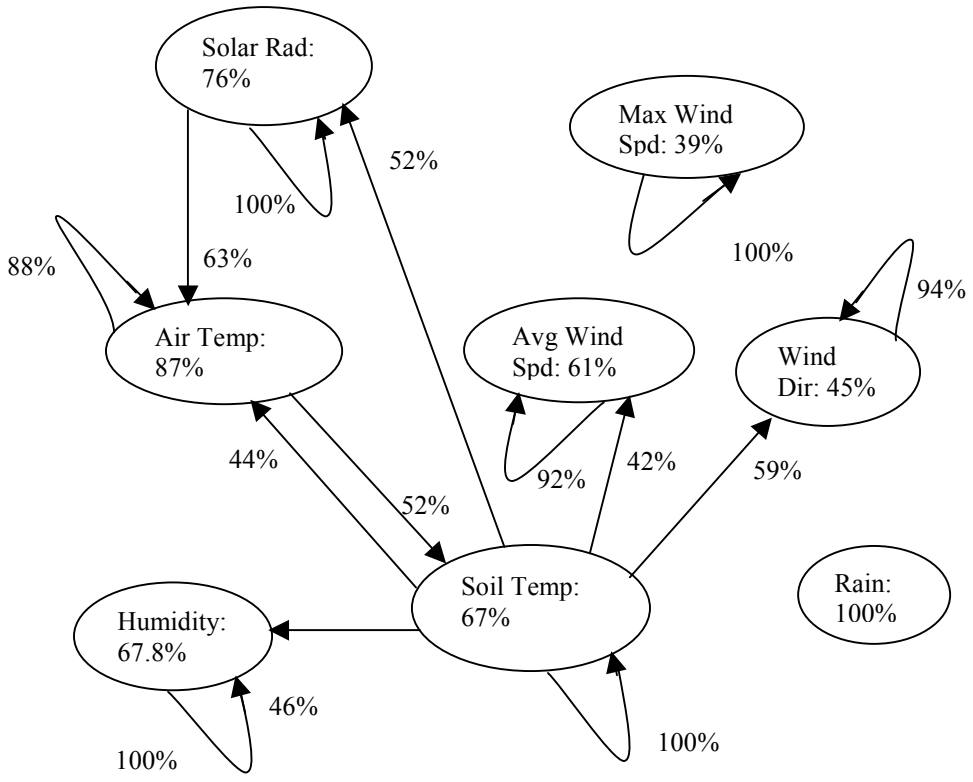


Figure 5.2 Dependence diagram for the weather data

Step 2. Remove nodes with strength values less than 50%. Results are in Figure 5.3.

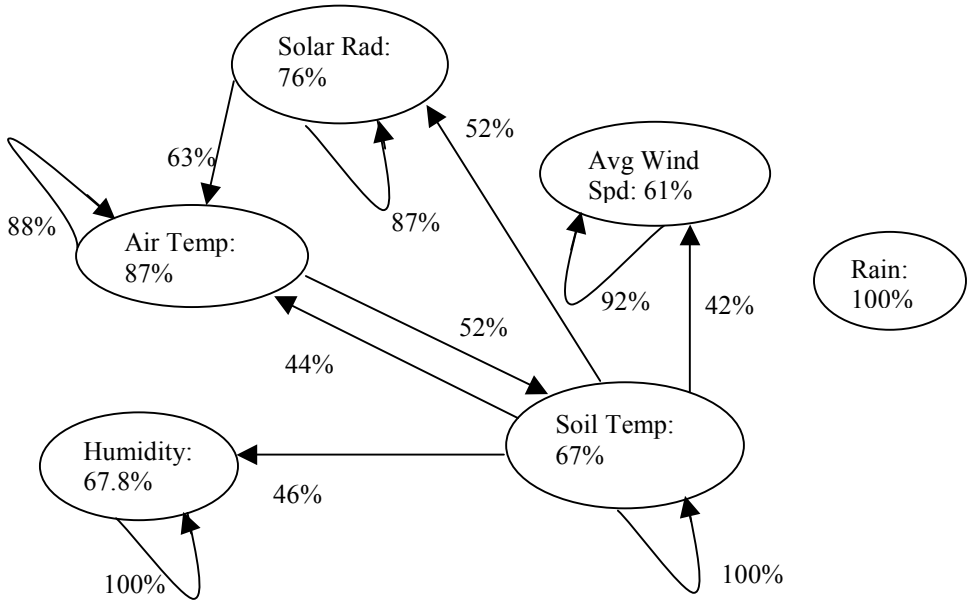


Figure 5.3 The pruned dependence diagram for the weather data

In a dependence diagram, the influence of a node is limited to the nodes to which it is pointing. Nodes that are connected together indirectly, such as Solar Radiation and Soil Temperature in the path Solar-Radiation → Air-Temperature → Soil-Temperature, do not necessary guaranty the presence of a direct link such as Solar-Radiation → Soil-Temperature (no transitivity). In our tests, however, when we removed the intermediate nodes, we would usually see the emergence of a link between the nodes, as illustrated in Figure 5.4, where the node Air Temperature is removed from the input data. A new link Solar-Radiation → Soil-Temperature emerges. However this result is not to be generalised.

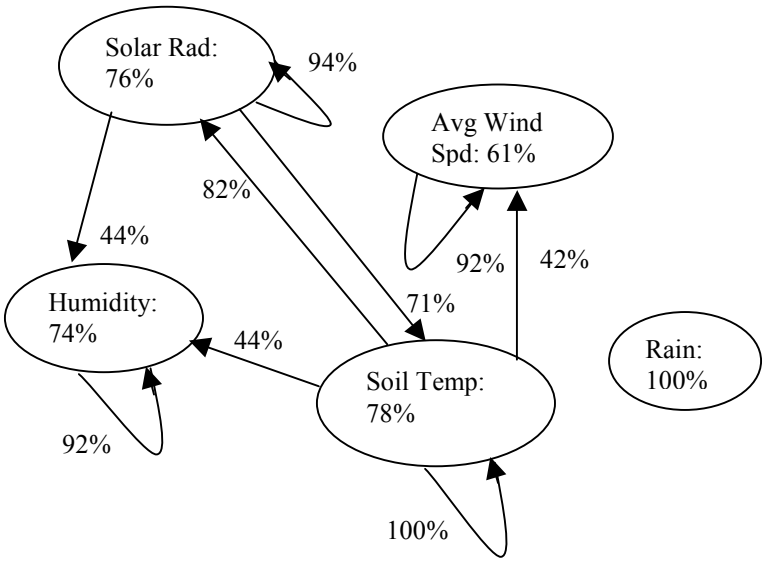


Figure 5.4 Dependence diagram with Air Temperature removed from the data

5.2.2 Using Regression to Generate Rules

In this subsection we compare the effectiveness of our TIMERS algorithm when using a regression programme called CART. When applicable, we have used both CART's regression, as well as its classification abilities. We show that the results are consistent with those obtained from classification with C4.5.

In the following tables, the values under "classification" represent the percentage of correct classifications done on training data (Training accuracy) and testing data (Predictive accuracy), while the values for "regression" represent the error values (mean square error). So higher values for classification are better, while lower values for regression are desired.

Unlike C4.5, CART has not been integrated into TimeSleuth, so to perform the following tests we prepared CART's input files manually by performing temporalisation and then removing the current-time condition attributes. To use CART as the classifier

with TIMERS, a few points have to be addressed. For one, CART's main output form is a decision tree, which does not cause any problem when evaluating the output, because the accuracy of the tree can be used instead of accuracy of rule sets. As to the complexity of the output, one could use the number of nodes in the tree as the measure of complexity.

The other point to consider when using CART is how to measure the quality intervals. As we have seen this was easy to do with accuracy values because they lie between 0% and 100%, while when performing regression, CART's error values start from 0.0 and increase without bounds. Hence a notion of overlap is harder to define in this case. For this reason we can refrain from constructing an interval, and simply compare the error values with a tolerance value as determined by the domain expert.

The Artificial Robot

Results of running regression on the artificial robot dataset are shown in Table 5.16.

Table 5.16 TIMERS' results with CART on the robot data. Verdict is Causal

Win	Pos	Regression		Classification		Type of test	Actual rules
		T Error	P Error	T Accuracy	P Accuracy		
1	1	1.822	1.402	23.1%	26.5%	Instantaneous	Instantaneous
2	1	0.657	0.672	55.3%	53.3%	Acausal	Acausal
2	2	0.0	0.0	100%	100%	Causal	Causal
3	1	0.657	0.673	55.3%	53.2%	Acausal	Acausal
3	2	0.0	0.0	100%	100%	Acausal	Acausal
3	3	0.0	0.0	100%	100%	Causal	Causal
4	1	0.657	0.675	55.3%	52.9%	Acausal	Acausal
4	2	0.0	0.0	100%	100%	Acausal	Acausal
4	3	0.0	0.0	100%	100%	Acausal	Acausal
4	4	0.0	0.0	100%	100%	Causal	Causal
5	1	0.657	0.675	55.7%	52.7%	Acausal	Acausal
5	2	0.0	0.0	100%	100%	Acausal	Acausal
5	3	0.0	0.0	100%	100%	Acausal	Acausal
5	4	0.0	0.0	100%	100%	Acausal	Acausal
5	5	0.0	0.0	100%	100%	Causal	Causal

As shown in Table 5.16, CART and C4.5 behave similarly when provided with the same data. The conclusion is the same in both cases: value of x is in a causal relation with

other attributes, because a causality test provides better results than either the instantaneous or acausality tests. The trend of having 100% accuracy for the tests continued with window sizes higher than 5. For the decision trees, the conclusion is that the system is causal, because the causal trees were smaller than the acausal ones, hence here the complexity measure works as a tie-breaker.

The weather data

As with the previous dataset the main aim is to compare CART's and C4.5's results so as to evaluate TIMERS' consistency in giving a verdict when using different rule/tree discovering programmes. We have set the soil temperature to be the decision attribute. The results are shown in Table 5.17.

Table 5.17 TIMERS' results with CART on Louisiana weather data

Window	Position	T Error	P Error
1	1	4.111	3.359
2	1	385.883	1173.173
2	2	0.562	0.716
3	1	2.055	2.193
3	2	0.463	0.648
3	3	0.653	0.669
4	1	4.127	5.104
4	2	0.456	0.656
4	3	0.463	0.615
4	4	0.472	0.484
5	1	2.051	2.226
5	2	0.463	0.680
5	3	0.487	0.654
5	4	0.454	0.675
5	5	0.626	1.442

Because Soil Temperature is a continuous attribute and CART does not have an integrated mechanism for discretisation, we only performed regression. With different window size values, CART displayed more variation than C4.5, but the user is still able

to make a decision as to the acausal nature of the relationship, because the results of the causal and acausal tests are close.

5.2.3 Short-interval Temporal Data

Previous temporal data concerned observations that were made one after the other, from the same system. It can happen that we deal with "bursts" of temporal data, where a certain number of observations are made at each time, and there is no relationship between the bursts.

In this section we attend to the problem of failure detection in a robot that grabs, moves, and puts down objects. Upon encountering a failure, the force and torque values along the *x*, *y*, and *z* axis (a total of 6 values) are recorded 15 times at regular intervals. The whole process takes 315 ms. The results are then used to classify the type of error that occurred. We are interested only in the type of failure, not in any given event or value in the data. The event (failure) has happened before the 15 records are collected. For this reason we should temporalise the data using a window size of 15 and a position for the decision attribute that is either before, or after the observations. This temporalisation is possible because we assume that at the time index of the decision attribute, no other (condition) attribute is present. We add a dummy attribute to all time steps to represent the decision attribute.

In [73], five strategies were used to create decision rules for solving the problem. The first one uses the 6 sensor values as they are, while in others these values are pre-processed, and then used in the decision making process. The 5^{th} strategy combines all the data available to other strategies. The observations have been divided into 5 learning

problem datasets. LP1 (failure in approach to grasp), LP2 (failure in transfer), LP3 (failure in positioning a part after a transfer), LP4 (failures in approach to ungrasp), and LP5 (failure in motion with part). This kind of data cannot be processed by the standard sliding position TIMERS method, as it does not make sense to place the decision attribute (occurrence of failure) within the observed records.

To obtain the results in Table 5.18, we used TIMERS to merge every 15 consecutive records into a single one, and used C4.5 to create decision trees. C4.5 was invoked with default parameters, with the exception of the values marked with a *, where the -g (use gain) option was used to generate the decision tree. In these cases the values obtained by default arguments appear in parenthesis. The five strategies covered in [73] are presented as S1 to S5.

Table 5.18 Accuracy values for the robot learning problem

Problem	S1	S2	S3	S4	S5	TIMERS	
						Position = 15	Position = 1
LP1	78%	80%	96%	85%	89%	97.7%	97.7%
LP2	45%	57%	51%	68%	64%	95.7%	95.7%
LP3	49%	75%	87%	85%	83%	85.1%* (48.0)	97.9%
LP4	65%	60%	95%	77%	83%	100%* (94.9)	100%* (99.1)
LP5	69%	63%	72%	49%	77%	90.9%* (89.0)	90.9%* (82.3)

There is a very close correspondence between the values obtained using different directions of time (positions of 1 or 15), which is to be expected, as the decision attribute is not a an attribute observed at any other time. We see that TIMERS gives either better or nearly the same accuracy values as the best of the 5 strategies in [73]. It is also more consistent compared to the other 5 strategies in terms of the quality of results. While TIMERS and S1 both use the original values of force and torque, TIMERS performs considerably better without requiring the user to come up with ways to process data,

which is a desirable quality because it frees the user from having to guess which pre-processing method should be used in any particular case.

5.2.4 Spatial Data

We have mentioned that there are similarities between sequential data that come from temporal observations of a system, and sequential data from a spatial domain. For the experiments described in this section, we used data generated while drilling an oil well [86]. It includes observations about the characteristics of the soil being pierced, including the porosity of the soil (the capacity to hold oil) and different resistance values. The records were registered every 0.5 metre, between the depths of 7400 and 8907.5 metres. The decision attribute was set to be the porosity.

The attributes are real-valued, so to produce results with a classifier such as C4.5, we discretised porosity to 20 different values. Regression is a more natural approach to this kind of data, and as we will see, regression works better than classification for this problem. TIMERS' results with classification are given in Table 5.19.

Table 5.19 TIMERS' classification results with C4.5 on the drilling data

Window	Position	T Accuracy	P Accuracy
1	1	43.2%	50.5%
2	1	42.6%	45.7%
2	2	43.1%	47.8%
3	1	38.1%	43.2%
3	2	45.4%	46.6%
3	3	40.7%	45.6%
4	1	39.9%	45.0%
4	2	45.7%	51.2%
4	3	43.6%	47.6%
4	4	40.5%	48.0%
5	1	38.4%	39.2%
5	2	42.6%	46.6%
5	3	44.5%	48.4%
5	4	45.6%	52.3%
5	5	38.4%	45.8%

We do not consider determining causality or acausality with this data. According to the results in Table 5.19, to predict the porosity at a given depth, the neighbouring values should be used. Throughout this Table the values are relatively close, and the results obtained with a window size of 1 are fairly close to those obtained with bigger window sizes, though over all the accuracy values are low because the attributes are numerical, and not suitable for a classification approach. We consider these results inconclusive, and suggest using regression instead.

On this data CART was much more effective with regression than classification. Table 5.20 shows CART's regression results.

Table 5.20 TIMERS' results with CART's regression on drilling-sample data

Window	Position	T Error	P Error
1	1	0.017	0.030
2	1	0.017	0.015
2	2	0.024	0.017
3	1	0.020	0.016
3	2	0.012	0.010
3	3	0.018	0.014
4	1	0.018	0.015
4	2	0.011	0.009
4	3	0.012	0.010
4	4	0.020	0.015
5	1	0.021	0.016
5	2	0.013	0.010
5	3	0.012	0.009
5	4	0.013	0.010
5	5	0.019	0.014

The error values are low and close to each other. Here we get better results when the decision attribute is in between some previous and next observations, implying that the value of porosity changes gradually. At the extreme ends, where the position is either 1 or equal to the window size, the error rate increase. The error rate is also higher for the case where window size is 1. This confirms that at any point, the current value of porosity is best predicted by observations made at both sides of that point.

Chapter 6

Concluding Remarks

In this thesis, we have presented the main characteristics and abilities of TIMERS and its implementation, TimeSleuth. In this chapter we discuss and summarise the main points of the approach. Section 6.1 discusses when this method can be applied for problem solving. Section 6.2 mentions the advantages and the disadvantages of this method, as they usually come together. A summary of the thesis is presented in Section 6.3.

6.1 Applicability of the Approach

For the examples of this section we assume that *A* and *B* denote the events of two attributes taking on certain values. The test for judging the causality of a set of rules depends on the quality of the rules in two temporal directions. For causality to manifest itself, the source of the observations should represent an irreversible relationship that exists in only the forward temporal direction. For example, *B* should often follow *A*, but *A* should only sometimes follow *B*. In such a case *A* would be considered a cause of *B*

because it can be reliably used to predict *B* using forward temporalisation, while *B* cannot be used to predict the presence of *A* with the same accuracy. The implication of this condition is that *B* should have more than one cause, so *B* would follow *A* more often than *A* following *B*.

This restriction on the temporal direction of a relationship matches the common sense notion of causality. If *A* and *B* always follow each other, then one may tend to conclude that they are causing each other. Many researchers are not interested in circular causality, and assume that a hidden common cause is making both *A* and *B* happen. TIMERS also considers such a relationship to be acausal.

TIMERS may make mistakes when *A* is the only cause of *B*, and *A* reliably causes *B*. In such a case we always observe *A* and then *B* in the normal direction of time (because *A* always causes *B*), and we always observe *B* and then *A* in the reverse direction of time (because *B* is only caused by *A*). In this case, TIMERS will declare the relationship as acausal. In the absence of any other knowledge, TIMERS' verdict in this case is intuitive, because one can argue that it is possible to have a hidden common cause that is at work and is causing both *A* and *B* to appear, but at different times.

From a more practical point of view, TIMERS works reliably when the effect always follows the cause within a specific amount of time (a time window), but the causes are more widely separated. In short, TIMERS declares a relationship between the decision attribute and the condition attributes to be causal when the relationship is not temporally circular and is not reversible in time. In practice, and in absence of domain knowledge that is not reflected in the input data, most real relations are better described as acausal.

Another aspect of when this method should be applied becomes evident when one needs to execute acausal rules. The question is how one can reference future values in the present? We must emphasise that using future values is just our test for distinguishing among causality and acausality of the relationship between the decision attribute and the condition attributes. As in the experiments in this thesis, when dealing with saved data, the future values are readily available in the next records. If we need to predict the decision attribute in real-time, then one could use rules generated by the causal method, even if they are less accurate than the acausal rules. In such cases the relationship would not be called causal.

Another important point is that we have been using the quality of decision rules (or trees), versus other representations to choose the type of the relationship at work. That choice is because in a decision rule there is a distinguished decision attribute, whose time of observation is assumed to be the current time. This assumption would not be valid in an association rule, where one or more attributes' values are associated with the values of other attributes. For example, $(A = a)$ and $(B = b) \rightarrow (C = c)$ and $(D = d)$ is an association rule, stating that if one observes certain values for attributes A and B, then one can also expect to see certain values for C and D. There is no single distinguished attribute that could serve as denoting the current time, so association rules and other knowledge representations with a similar feature cannot be used in TIMERS.

6.2 The Main Limitations of TIMERS

Our approach considers a set of input, namely sequential data, that is more restrictive than what is acceptable to software such as TETRAD and CaMML, so we do not attempt

to solve the causality problem in a broad sense. While we consider the inclusion of a temporal order between the possible causes and effects as intuitive, using a rule discoverer to select the relevant attributes for predicting the condition attribute is not always reliable. We have encountered cases where obviously wrong choices in condition attributes were made because with those particular data, better rules could be generated by using those attributes. Fortunately in most cases the quality of the rules that used irrelevant attributes were not very high, and we were able to discard the results.

Another point is that while TIMERS' running time is usually good (classification rule discoverers are usually fast), if the user is interested in analysing a number of attribute's influence on each other, then the algorithm has to be run many times, and the results integrated in a dependence diagram. Bayesian methods usually do that by default. More generally, the user has to specify the values of six parameters for the TIEMRS method, and this requirement implies that the user must have a good understanding of characteristics of the input data.

TIMERS exploits the sequential nature of its input data to discover causality and acasuality, which creates a restriction on the types of input data that are appropriate for analysis with this approach. While causal Bayesian methods can be used on census type data, for example, where each record comes from a different source and may be collected at different times, the data suitable for TIMERS must obey a strict temporal order.

Another characteristic of this method is that if the domain of the attributes can take on many values (the attributes are real-valued, for example), and there are not enough observations to sample all the values sufficiently, then the decision rules may be of poor quality, and no causal relationship may be detected. To remedy this problem, we suggest

that if an attribute takes on many different values compared with the number of input records, then the user should discretise that attribute to help in the process of rule discovery.

A potential pitfall should be mentioned. Suppose a person chooses a reversible operator, and applies it to the values of a number of attributes. Would this system be labelled as acausal? If the values produced after each application of the function are never repeated, then our method cannot discover reliable classification rules, and the method refrains from giving a verdict. If, however, the results are cyclic, then the method may well find relationships in both directions, and the verdict will be acausal. However, cyclic observations can imply cyclic causes and effects, which we consider to be of no interest in this work.

To summarise: (1) TIMERS is scalable in the number of attributes, but may need several runs in case the user is interested in predicting many attributes. (2) It is intuitive in that it strictly requires time to pass between causes and effects. (3) The rule discoverer may not choose the relevant attributes, and frequently, choosing irrelevant attributes results in rules with poor quality. (4) TIMERS usually performs much better than other methods when given the suitable sequential data, but the restrictions on the suitable type of data limit its applicability. (5) The domain expert may need to observe the data and perform operations such as discretisation to obtain more reliable results.

6.3 Summary

We have presented a method to discover and distinguish between instantaneous, causal, and acausal relationships between a decision attribute and a set of condition

attributes. The proposed method is based on the assumption that the passage of time and causality are closely related.

TIMERS tests to see whether referring to condition attribute values that appear at different time steps increases the accuracy of the prediction of a decision attribute's value. If not, then the relationship is instantaneous. If including a time difference between the attribute observations results in better prediction, then a distinction is made as to whether the relationship is causal (previous values of the condition attributes determines the present value of the decision attribute) or acausal (succeeding values determines the present value of the decision attribute). Each test is performed after an appropriate type of temporalisation. We used the accuracy values of the rules as an indication of the appropriateness of the temporalisation method, and hence the type of the relationship. In general any other measurement can be used. This method works with different underlying rule-discovery programs, as evidenced by employing two very different programmes, C4.5 and CART.

The resulting rules show us which attributes are important in predicting the value of the decision attribute. They also show how the relationship is formed. For example, in the Louisiana weather data, the soil temperature an hour before the current time had the most importance in determining the soil temperature [38].

The rule sets are useful by themselves, but to help the user better understand them, dependence diagrams graphically show which attributes take part in the process of classification. The strength of the relations among the condition attributes and the decision attribute is also displayed. The user thus sees how the attributes influence each other.

The TIMERES method can be applied to any one-dimensional data. The similarities between a spatial line and the arrow of time make this generalisation intuitive. When there is no significance between moving backwards or forwards in the data, as non-temporal, on-dimensional data, there is no need to distinguish the resulting rules as causal or acausal.

The TimeSleuth package includes executables and source code in Java, as well as help and example files. It can be downloaded freely from http://www.cs.uregina.ca/~karimi/doanloads.html.

References

[1] Agrawal R. and Srikant R., Mining Sequential Patterns*, Proceedings of the International Conference on Data Engineering (ICDE'1995)*, Taipei, Taiwan, March 1995.

[2] Aldous, D., and Fill J.A., *Reversible Markov Chains and Random Walks on Graphs*, Book in preparation, http://stat-www.berkeley.edu/~aldous/RWG/book.html.

[3] Antunes, C. and Oliveira, A., Using Context-Free Grammars to Constrain Apriori-based Algorithms for Mining Temporal Association Rules, *Workshop on Temporal Data Mining (KDD2002)*, Edmonton, Canada, July, 2002.

[4] Baggott, J., *The Meaning of Quantum Theory: A Guide for Students of Chemistry and Physics*, Oxford Science Publications, 1992.

[5] Bennett, C. H., Logical Reversibility of Computation, *IBM Journal of Research and Development* 17:525-532, November, 1973.

[6] Berndt, D. J. and Clifford, J., Finding Patterns in Time Series: A Dynamic Programming Approach, *Advances in Knowledge Discovery and Data Mining*. U.M.

Fayyad, G. Piatetsky-Shapiro, P. Smyth, et al. (eds.), AAAI Press/ MIT Press, 1996, pp. 229-248,

[7] Bertsekas, D.P., and Tsitsiklis, J.N., *Introduction to Probability*, Athena Scientific, 2002.

[8] Blake, C.L and Merz, C.J., UCI Repository of machine learning databases http://www.ics.uci.edu/~mlearn/MLRepository.html. Irvine, CA: University of California, Department of Information and Computer Science, 1998.

[9] Bowes, J., Neufeld, E., Greer, J. E. and Cooke, J., A Comparison of Association Rule Discovery and Bayesian Network Causal Inference Algorithms to Discover Relationships in Discrete Data, *Proceedings of the Thirteenth Canadian Artificial Intelligence Conference (AI'2000)*, Montreal, Canada, 2000.

[10] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. *Classification and Regression Trees*. Wadsworth Inc., 1984.

[11] Brin, S., Motwani, R., Tsur, D. and Ullman, J., Dynamic Itemset Counting and Implication Rules for Market Basket Data, *Proceedings of 1997 ACM SIGMOD*, Montreal, Canada, June 1997.

[12] Chatfield, C., *The Analysis of Time Series: An Introduction*, Chapman and Hall, 1989.

[13] Chickering, D., Geiger, D., and Heckerman, D., Learning Bayesian Networks is NP-Hard*, Technical Report MSR-TR-94-17*, Microsoft Research, 1994.

[14] Chow, T.L., *Mathematical Methods for Physicists: A Concise Introduction*, Cambridge University Press, 2000.

[15] Clocksin, W.F., Melish, C.S, *Programming in Prolog*, Springer Verlag, 1984.

[16] Durand-Lose, J. O., Reversible Space-Time Simulation of Cellular Automata, *Rapport de Recherche Numéro 1177-97*, Université Bordeaux I, 1997.

[17] Feinberg, G., Possibility of Faster-Than-Light Particles, *Physical Review 159 N5*, 1967, pp.1089-1105.

[18] Freedman, D. and Humphreys, P., *Are There Algorithms that Discover Causal Structure?*, Technical Report 514, Department of Statistics, University of California at Berkeley, 1998.

[19] Granger, C., Investigating Causal Relations by Econometrics Models and Cross-Spectral Methods, *Econometrica*, volume 37, 1969, pp. 424-438.

[20] Grefenstette, J.J., Ramsey, C.L., and Schultz, A.C, Learning Sequential Decision Rules Using Simulation Models and Competition, *Machine Learning 5(4)*, 1990, pp. 355-381.

[21] Guralnik, V., Wijesekera, D. and Srivastava, J., Pattern Directed Mining of Sequence Data, *Proceedings of the Fourth International Conference on Knowledge Discovery & Data Mining (KDD'1998)*, 1998.

[22] Hawking, S.W., Thorne, K.S., Novikov, I., Ferris, T., and Lightman, A., *The Future of Spacetime*, Norton, 2002.

[23] Hawking, S.W., *The Universe in a Nutshell*, Bantam Books, 2001.

[24] Heckerman, D., *A Bayesian Approach to Learning Causal Networks*, Microsoft Technical Report MSR-TR-95-04, Microsoft Corporation, May 1995.

[25] Heckerman, D., Geiger, D. and Chickering, D.M., Learning Bayesian Networks: The Combination of Knowledge and Statistical Data, *Machine Learning*, 20(3), pp. 197-243. 1995.

[26] Höppner, F., *Knowledge Discovery from Sequential Data*, PhD dissertation, Fachbereich für Mathematik und Informatik der Technischen Universität Braunschweig, 2003.

[27] Höppner, F., Discovery of Temporal Patterns: Learning Rules about the Qualitative Behaviour of Time Series, *Principles of Data Mining and Knowledge Discovery* (*PKDD'2001*), 2001.

[28] Humphreys, P. and Freedman, D., The Grand Leap, *British Journal of the Philosophy of Science 47*, pp. 113-123, 1996.

[29] Johnson, G., *A Shortcut Through Time: The Path to the Quantum Computer*, New Alfred A. Knopf, New York, 2003.

[30] Kadous, M. W., Learning comprehensible descriptions of multivariate time series, *The 16th International Conference on Machine Learning*, pp. 454-463, 1999.

[31] Kari, J., Reversibility of 2D cellular Automata is Undecidable, *Physica D*, 45:379-385, 1990.

[32] Karimi, K., and Hamilton, H.J., Temporal Rules and Temporal Decision trees: A C4.5 Approach, *Technical Report CS-2001-02*, Department of Computer Science, University of Regina, December 2001.

[33] Karimi, K. and Hamilton, H.J., Finding Temporal Relations: Causal Bayesian Networks vs. C4.5, *The Twelfth International Symposium on Methodologies for Intelligent Systems (ISMIS'2000)*, Charlotte, NC, USA, October 2000, pp. 266-273.

[34] Karimi, K. and Hamilton, H.J., Learning With C4.5 in a Situation Calculus Domain, *The Twentieth SGES International Conference on Knowledge Based Systems and Applied Artificial Intelligence (ES2000)*, Cambridge, UK, December 2000, pp. 73-85.

[35] Karimi, K. and Hamilton, H.J., Logical Decision Rules: Teaching C4.5 to Speak Prolog, *The Second International Conference on Intelligent Data Engineering and Automated Learning (IDEAL'2000)*, Hong Kong, December 2000, pp. 85-90.

[36] Karimi, K., and Hamilton, H.J. TimeSleuth: A Tool for Discovering Causal and Temporal Rules, *The 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'2002)*, Washington DC, November, 2002, pp. 375-380.

[37] Karimi, K., and Hamilton, H.J., Distinguishing Causal and Acausal Temporal Relations, *The Seventh Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'2003)*, Seoul, South Korea, April/May 2003, pp 234-240.

[38] Karimi, K. and Hamilton H.J., Using TimeSleuth for Discovering Temporal/Causal Rules: A Comparison, *The Sixteenth Canadian Artificial Intelligence Conference (AI'2003)*, Halifax, Nova Scotia, Canada, June 2003, pp. 175-189.

[39] Karimi, K., and Hamilton, H.J., From Temporal Rules to One Dimensional Rules, *Proceedings of the Workshop on Causality and Causal Discovery, Technical Report CS-2004-02*, Kamran Karimi (Ed.). Department of Computer Science, University of Regina, May 2004. pp 30-44.

[40] Karimi, K., Mehrandezh, M., and Hamilton, H.J., A Proposal for Self-Recognition in Robot Programming, *The 18th Annual Canadian Conference on Electrical and Computer Engineering (CCECE'2005)*, Saskatoon, Canada, May 2005.

[41] Kemeny, J.G., and Snell, J.L., *Finite Markov Chains*, Van Norstrand, New York, 1960.

[42] Kennett, R.J., Korb, K.B., and Nicholson, A.E., Seabreeze Prediction Using Bayesian Networks: A Case Study, *Proc. Fifth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'2001).* Hong Kong, April 2001.

[43] Keogh, E. J. and Pazzani, M. J., Scaling up Dynamic Time Warping for Data Mining Applications, *The Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'2000)*, August 2000.

[44] Korb, K. B. and Wallace, C. S., In Search of Philosopher's Stone: Remarks on Humphreys and Freedman's Critique of Causal Discovery, *British Journal of the Philosophy of Science 48,* 1997, pp. 543-553.

[45] Koza, J. R. and Rice, J. P. Automatic Programming of Robots using Genetic Programming. *The Tenth National Conference on Artificial Intelligence*, Menlo Park, CA, USA, 1992.

[46] Krener, A. J., Acausal Realization Theory, Part I; Linear Deterministic Systems. *SIAM Journal on Control and Optimization*, Vol 25, No 3, 1987, pp. 499-525.

[47] Levesque, H. J., Reiter, R., Lespérance, Y., Lin, F. and Scherl. R., GOLOG: A Logic Programming Language for Dynamic Domains, *Journal of Logic Programming, 31*, 1997, pp. 59-84.

[48] Levy, S., *Artificial Life: A Quest for a New Creation*, Pantheon Books, 1992.

[49] Lin F. and Reiter, R., Rules as actions: A Situation Calculus Semantics for Logic Programs. *Journal of Logic Programming Special Issue on Reasoning about Action and Change*, 31(1-3), 1997, pp.299-330.

[50] Li, Y., Ning, P., Wang X.S., and Jajodia, S., Discovering Calendar-based Temporal Association Rules, *Proceedings of the 8th International Symposium on Temporal Representation and Reasoning (TIME'2001)*, Italy, June 2001, pp. 111-118.

[51] Lin, J, Keogh, E. and Truppel, W., Clustering of streaming time series is meaningless, The eighth ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, San Diego, California, USA, 2003, pp. 56-65.

[52] Lindley, D., *Where does the Weirdness Go?*, Vintage, 1997.

[53] Mannila, H., Toivonen, H. and Verkamo, A. I., Discovering Frequent Episodes in Sequences, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, 1995, pp. 210-215.

[54] McCarthy, J. and Hayes, P. C, Some Philosophical Problems from the Standpoint of Artificial Intelligence. *Machine Intelligence 4*, 1969.

[55] Metropolis, N., Rosenbluth, A. W., Rosenbluth, N., Teller, A. H., and Teller, E. Equation of state calculation by fast computing machines. Journal of Chemical Physics 21: 1087-1092, 1953.

[56] Mitchell, T., *Machine Learning*, McGraw Hill, 1997.

[57] Moore, R. C., The Role of Logic in Knowledge Representation and Commonsense Reasoning, *Readings in Knowledge Representation*, Morgan Kaufmann, 1985, pp. 335-341.

[58] Morita, K., Computation Universality of One-Dimensional Reversible Cellular Automata. *Information Processing Letters*, 42:325-329, 1992

[59] Morrison, M.A., *Understanding Quantum Physics: A User's Manual*, Prentice Hall, 1990.

[60] Nadel, B.A., Constraint Satisfaction Algorithms, *Computational Intelligence*, No 5, 1989.

[61] Oates, T. and Cohen, P. R., Searching for Structure in Multiple Streams of Data, *Proceedings of the Thirteenth International Conference on Machine Learning*, 1996, pp. 346 – 354.

[62] Pearl, J., *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann Publishers, 1988.

[63] Pearl, J., *Causality: Models, Reasoning, and Inference*, Cambridge University Press, 2000.

[64] Penrose, R., *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics*, Oxford University Press, 1989.

[65] Poole, D., Decision Theory, the Situation Calculus, and Conditional Plans, Linköping Electronic Articles in Computer and Information Science, Vol. 3: nr 3, 1998, http://www.ep.liu.se/ea/cis/1998/008.

[66] Quinlan, J. R., *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.

[67] Roddick, J.F. and Spiliopoulou, M., *Temporal Data Mining: Survey and Issues*, Research Report ACRC-99-007. School of Computer and Information Science, University of South Australia, 1999.

[68] Russel, S. and Norvig, P., *Artificial Intelligence: A Modern Approach*, Prentice Hall International, 1995.

[69] Sætrom, P. and Hetland, M.L., Unsupervised Temporal Rule Mining with Genetic Programming and Specialized Hardware, *International Conference on Machine Learning and Applications (ICMLA'2003)*, 2003.

[70] Scheines, R., Spirtes, P., Glymour, C. and Meek, C., *Tetrad II: Tools for Causal Modeling*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1994.

[71] Scheines R, Spirtes P, Glymour C, Meek C, Richardson T., The TETRAD Project: Constraint-Based Aids to Causal Model Specification, *Multivariate Behavioral Research*, 33, 1998, pp. 65-118.

[72] Schwarz, R. J. and Friedland B., *Linear Systems*. McGraw-Hill, New York. 1965.

[73] Seabra Lopes, L. and Camarinha-Matos, L.M. Feature Transformation Strategies for a Robot Learning Problem, *Feature Extraction, Construction and Selection. A Data Mining Perspective*, H. Liu and H. Motoda, Editors, Kluwer Academic Publishers, 1998.

[74] Shafer, G., Causal Relevance. *Reasoning with Uncertainty in Robotics*, Lecture Notes in Artificial Intelligence 1093, 1996, pp. 187-208.

[75] Shafer, G., *The Art of Causal Conjecture*, The MIT Press, 1996.

[76] Silverstein, C., Brin, S., Motwani, R. and Ullman J., Scalable Techniques for Mining Causal Structures, *Proceedings of the 24th VLDB Conference*, New York, USA, 1998, pp. 594-605.

[77] Sloane, N.J.A., and Wyner A.D., Editors, *Claude Elwood Shannon: Collected Papers,* New York: IEEE Press, 1993.

[78] Spirtes, P. and Scheines, R., Reply to Freedman, In McKim, V. and Turner, S. (editors), *Causality in Crisis*, University of Notre Dame Press, 1997, pp. 163-176.

[79] Tooley, M. (Ed.), *Analytical Metaphysics: A Collection of Essays*, Garland Publishing, Inc., 1999.

[80] Van Le, T. *Techniques of Prolog Programming*. John Wiley & Sons, 1993.

[81] Wallace, C. and Boulton, D., An Information Measure for Classification, *Computer Journal* 11:185-194, 1968.

[82] Wallace, C., Korb, K., and Dai, H., Causal Discovery via MML, *13th International Conference on Machine Learning (ICML'1996)*, 1996, pp. 516-524.

[83] Witten, I.A., and Frank, E., *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann, 2000.

[84] Wong, S.K.M., Butz, C.J., and Wu, D., On the Implication Problem for Probabilistic Conditional Independency, *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, Vol. 30, No. 6, November 2000, pp. 785-805.

[85] Zadeh, L., Causality is undefinable, Abstract of a Lecture Presented at the BISC Seminar, *http://www.cs.berkeley.edu/~nikraves/zadeh/Zadeh2.doc*, University of California, Berkeley, 2001.

[86] http://explorer.ndic.state.nd.us/. Our data came from a sample CD.

[87] http://www.cs.uregina.ca/~karimi/downloads.html/URAL.java

[88] http://typhoon.bae.lsu.edu/datatabl/current/sugcurrh.html. Contents change.

[89] http://poli.haifa.ac.il/~levi/inference.html

[90] http://mathworld.wolfram.com/Likelihood.html

[91] http://www.phil.cmu.edu/projects/tetrad/tet3/chp2.htm

[92] TETRAD Project Homepage. http://www.phil.cmu.edu/projects/tetrad/tet3/chp2.htm

[93] TETRAD Download Site. http://www.phil.cmu.edu/projects/tetrad_download/

Appendix A

TimeSleuth's Output in Prolog for the Robot's *x* Movements

```
:- op(800, xfx, ule).
:- op(800, xfx, ug).

class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 0) :- OriginalDecision_t1 = 2, X_t1 = 0.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 0) :- OriginalDecision_t1 = 0, X_t1 = 0.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 0) :- OriginalDecision_t1 = 2, X_t1 = 1.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 0) :- OriginalDecision_t1 = 1, X_t1 = 0.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 1) :- OriginalDecision_t1 = 0, X_t1 = 1.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 1) :- OriginalDecision_t1 = 3, X_t1 = 0.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 1) :- OriginalDecision_t1 = 1, X_t1 = 1.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 1) :- OriginalDecision_t1 = 2, X_t1 = 2.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 2) :- OriginalDecision_t1 = 0, X_t1 = 2.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 2) :- OriginalDecision_t1 = 3, X_t1 = 1.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 2) :- OriginalDecision_t1 = 2, X_t1 = 3.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 2) :- OriginalDecision_t1 = 1, X_t1 = 2.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 3) :- OriginalDecision_t1 = 0, X_t1 = 3.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 3) :- OriginalDecision_t1 = 3, X_t1 = 2.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 3) :- OriginalDecision_t1 = 2, X_t1 = 4.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 3) :- OriginalDecision_t1 = 1, X_t1 = 3.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 4) :- OriginalDecision_t1 = 3, X_t1 = 3.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 4) :- OriginalDecision_t1 = 0, X_t1 = 4.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 4) :- OriginalDecision_t1 = 1, X_t1 = 4.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 4) :- OriginalDecision_t1 = 2, X_t1 = 5.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 5) :- OriginalDecision_t1 = 1, X_t1 = 5.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 5) :- OriginalDecision_t1 = 0, X_t1 = 5.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 5) :- OriginalDecision_t1 = 2, X_t1 = 6.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 5) :- OriginalDecision_t1 = 3, X_t1 = 4.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 6) :- OriginalDecision_t1 = 1, X_t1 = 6.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 6) :- OriginalDecision_t1 = 0, X_t1 = 6.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 6) :- OriginalDecision_t1 = 3, X_t1 = 5.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 6) :- OriginalDecision_t1 = 2, X_t1 = 7.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 7) :- OriginalDecision_t1 = 1, X_t1 = 7.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 7) :- OriginalDecision_t1 = 3, X_t1 = 6.
```

```
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 7) :- OriginalDecision_t1 = 3, X_t1 = 7.
class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 7) :- OriginalDecision_t1 = 0, X_t1 = 7.

class(Y_t1, F_t1, OriginalDecision_t1, X_t1, 0).


/* ule unifies the left hand side or performs =< */
A ule B :- var(A), A = B.
A ule B :- A =< B.

/* ug unifies the left hand side or performs > */
A ug B :- var(A), A is B + 1.
A ug B :- A > B.

/* umember unifies a variable, or performs member() */
umember(A, B) :- var(A), [X|_] = B, A = X.
umember(A, B) :- member(A, B).

member(A, [A|_]).
member(A, [_|B]) :- member(A, B).
```