Data Mining Temporal and Indefinite Relations with Numerical Dependencies

Ethan Richard Collopy

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of the

University of London.

Department of Computer Science

University College London

January 1999

ABSTRACT

We propose that data mining, the search for useful, non-trivial and previously unknown information within a database, can be successfully performed with Numerical Dependencies (NDs), a generalisation of Functional Dependencies (FDs), to model the data, together with resampling, a computationally intensive statistical sampling process, which allows us to make inferences from temporal and indefinite databases.

We use NDs to model relations containing temporal and indefinite information. We extend the theory of NDs by presenting measures for data mining and generalise the chase procedure, a method for updating a relation to satisfy a constraint set, for NDs. We motivate NDs in real-world applications by introducing a database design tool.

The consistency problem, that of attempting to find a relation satisfying a set of FDs within an indefinite relation, known to be NP-complete, is studied in the context of using NDs for approximation. We employ resampling, based on taking samples of definite relations from indefinite ones, on incremental sample sizes until an approximate fixpoint is reached, denoting an upper bound on the required sample size. Extensive simulations highlight that resampling to find upper bounds in conjunction with the chase for indefinite relations returns valid approximate solutions.

We also study NDs in temporal sequences of relations for knowledge discovery purposes. Each relation within a sequence is mined for a set of NDs which evolve with updates in data. We introduce a temporal logic for the discovery of rules and properties within these sequences, or subsequences, which includes statistical functions within the temporal operators for time series analysis. We also show that time series data may be analysed using a restricted set of the logic. We apply discovery algorithms to both sequences and resampled sequences, allowing smoothing for trend detection. Investigations, presented herein, show these rules to provide interesting and practicable results.

ACKNOWLEDGEMENTS

The work in this thesis was funded by the EPSRC via a quota award. Financial support from the Department of Computer Science, UCL, and the UCL Graduate School is gratefully acknowledged.

I am indebted to my supervisor, Mark Levene, not only for his continual support, advice, and encouragement, throughout the term of my research but also for the numerous stimulating discussions we have had on databases, data mining, and many other research issues. Similarly, thanks go to Sean Holden, my second supervisor. I would also like to thank Jose Borges, Wilfred Ng, Nadav Zin, and many other colleagues at UCL for making my studies here even more enjoyable.

Thanks to the various anonymous reviewers of our papers who have unknowingly improved the content of this thesis.

I would like to thank my parents, to whom I dedicate this thesis, and without whose love and support I would surely not now be in the position of writing these thesis acknowledgements. I also thank Deborah for her patience and support.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

SYMBOL INDEX

Symbol	Meaning
X, Y, Z	sets of attribute names
A, B, C	attribute names
DOM(A)	the domain of an attribute A
R	the set of attribute names in relation R
r(R)	a relation r defined over attributes R
t	a tuple t in a relation
t[X]	the projection of a tuple t onto a set of attributes X
$\pi_X(r)$	the projection of a relation r onto a set of attributes X
r[X, x]	partition of a relation r whose X value is x
$X \rightarrow Y$	X functionally determines Y
$X \rightarrow^k Y$	X numerically determines up to k different Y values
$r \models \sigma$	r logically implies σ
$r \not\models \sigma$	r does not logically imply σ
$r \models\!\!\approx \sigma$	indefinite relation r logically implies σ
Σ	a set of data dependencies
Σ^+	the set of dependencies logically implied by Σ
SAT(Σ)	the set of all relations which satisfy Σ
ATT(d)	the set of attributes appearing in an FD, denoted by d
$\equiv$	logical equivalence
X^+	the closure of the set of attributes X
DEP(X)	the dependency basis of X
$\mid X \rightarrow Y \mid$	the *size* of an FD $X \rightarrow Y$
$\parallel F \parallel$	the sum of all the sizes of all the FDs in F
FD	Functional Dependency
ND	Numerical Dependency
AR	Armstrong Relation
(R)DBMS	(Relational) Database Management System
ADS	Active Domain Size
$\top$	the top of a lattice
$\bot$	the bottom of a lattice
$\mid S \mid$	the cardinality of a set S
$\mathcal{P}(S)$	the powerset of a set S
$n!$	the factorial of n
$\forall$	universal quantification
$\exists$	existential quantification
$\Diamond\sigma$	at some point in the future σ
$\Box\sigma$	at all points in the future σ

$\bigcirc\sigma$	at the next point in the future σ
$\sigma_1 \mathcal{U} \sigma_2$	σ_1 until σ_2
$\sigma_1 \mathcal{S} \sigma_2$	σ_1 since σ_2
Δ	temporal relation sequence or temporal database
$s \preceq \Delta$	sequence s is contained in Δ
$s1 \lessdot s2$	$s1$ starts before $s2$ and $s2$ ends after $s1$
$s1 \gtrdot s2$	$s2$ starts before $s1$ and $s1$ ends after $s2$
$(\Delta, r) \models^w \sigma$	relation r in Δ satisfies σ over a moving average window of size w
$(\Delta, s) \models^w \sigma$	sequence r in Δ satisfies σ over a moving average window of size w
$\boxminus^n \sigma$	all sequences of size n satisfy σ
$\diamondminus^m \sigma$	some sequence of size n satisfies σ
$\diamondminus^n \boxminus^m \sigma$	some sequence of size n satisfies $\boxminus^m \sigma$, where $m \leq n$
$\boxminus^n \diamondminus^m \sigma$	all sequences of size n satisfy $\diamondminus^m \sigma$, where $m \leq n$
$\sigma_1 \rightsquigarrow \sigma_2$	A sequence, say $s1$, satisfying σ_1, starts before a sequence, say $s2$, satisfying σ_2, and $s2$ ends after $s1$
$\sigma_1 \wedge^k \sigma_2$	A sequence satisfies σ_1 and σ_2 and there is a maximum correlation with lag value k

CHAPTER 1

Introduction

Knowledge Discovery in databases is currently a particularly fast growing area of computing research, not least because it can be said to be the hybrid of a number of other research disciplines as shown in Figure 1.1, primarily statistics, machine learning, and database theory, with direct real-world application.

In this thesis we propose a general approach to knowledge discovery problems in databases which contain either indefinite or temporal information. Throughout we use *Numerical Dependencies* (NDs), a generalisation of the *Functional Dependency* (FD), and show how they are applicable in numerous domains. We also use and develop some resampling processes, which are computationally intensive statistical procedures, well suited to inferring information from databases containing temporal and indefinite information.

In Section 1.1 we present the goal of the thesis, moving on to discuss knowledge discovery in databases in 1.2, where we place our work in context and present a brief example for overview. We detail the contribution of this work in Section 1.3 and outline the rest of the thesis in 1.4. Lastly, we detail notation in Section 1.5.

1.1 The Goal of the Thesis

The ability to discover *knowledge* from a database which is not explicitly represented in the data is clearly a desirable goal. We propose that such *data mining* can be achieved using NDs, generalisations of the FD, which themselves have a well-defined semantics for application within the relational model. The application of NDs allow data mining principles to be exercised on categorical data, often the bulk of many corporate databases, or a combination of categorical and numerical data.

We show how relations containing indefinite or temporal data satisfy numerous ND sets, being either definite instances of indefinite data or possibly changing ND sets over time. The ND

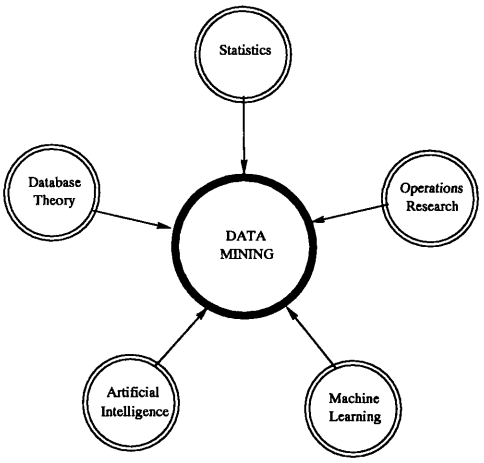

Figure 1.1: Components of the Data Mining Process

sets satisfied are obtained from an initial template of FDs which is supplied by the user; alternatively, it would be possible to mine for ND set satisfaction. In the indefinite domain the ND sets are satisfied in definite instances of the same indefinite relation. We show how *resampling*, a computationally intensive sampling procedure, may be applied on increasing sample sizes to determine an approximate fixpoint upon which a heuristic based hill-climbing algorithm is employed to find a suitable ND set approximation to functional satisfaction. In the temporal domain the ND sets may change over time for the same attributes; we show how resampling and other time series statistics may be employed to determine *properties* which may hold over time, using a logic we have developed. Our data mining framework thereby discovers information using many ND set approximations upon which statistics are applied, varied for the domain in question, to make further inferences from the data.

1.2 Knowledge Discovery in Databases and this thesis

The following widely accepted definition is due to (Fayyad et al., 1996d):

Definition 1.2.1 (Knowledge Discovery in Databases) Knowledge Discovery in Databases is defined as *the nontrivial extraction of valid, previously unknown, potentially useful, and ultimately understandable information from a database.* □

Knowledge Discovery may only provide *potentially* useful information given that it may frequently discover relations, possibly weak, between unconnected real-world information or even relations that do not serve the interests of the user. This includes the possible discovery of redundant information. For instance, in a medical database a system may discover a dependency $pregnant \rightarrow female$ implying that all pregnant patients are female; obviously such information is superfluous. Methods to prevent such redundant information generation may include the spec-

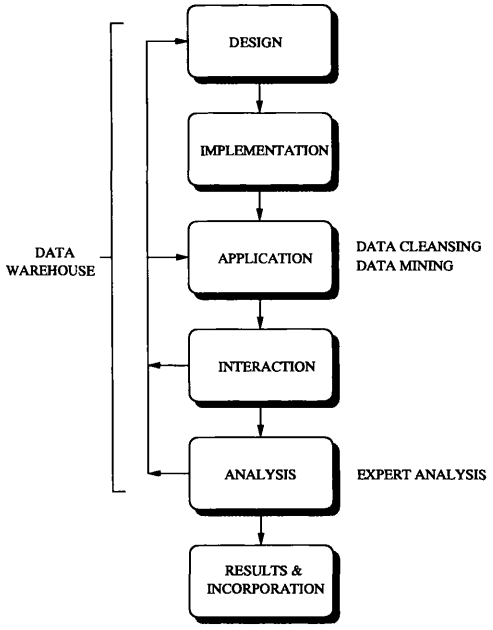

Figure 1.2: The Knowledge Discovery Application Cycle

ification of trivial associations before the mining process takes place and continuous interaction with an expert, a much understated component of knowledge discovery, defined as *data archaeology* (Brachman and Anand, 1996), as well as provision of a dependency template upon which knowledge is discovered.

Knowledge discovery comprises a number of component parts including data cleansing, design, warehousing and mining as well as expert analysis, detailed in Figure 1.2. Databases within data warehouses are now frequently designed with a view to data mining operations; where the goal of the database is *reliable storage* the goal of the data warehouse is *decision support* (Fayyad, 1998b). The process of data cleansing includes collecting data from different sources and processing it into a homogeneous form. The design and implementation of a capable knowledge discovery tool will handle these stages. Figure 1.2 displays a generic cycle for knowledge discovery. It highlights the requirement that interaction and analysis of the system may need to return to the design stage if initial results suggest comparison with new data, which may have been omitted, or the data cleansing stage has to be repeated with new error parameters, perhaps to adjust the error and confidence for noise within the database. Recent work has included research on formalising the data warehouse (Hammer et al., 1995; Inmon, 1996) as well as significant data cleansing research. Interesting as the issues of data cleansing and warehousing are, we do not consider them further within this thesis, concentrating on data mining.

Knowledge discovery refers to the process of extracting patterns and relationships from the data whereas data mining refers to the actual process of applying these algorithms to the data, though many of the boundaries are vague. In data mining applications a key requirement is the preparation of data for analysis. In Figure 1.2, an example of a KDD application cycle, we assume that the design and application components handle any required data cleansing. Many data mining algorithms may also divide the data into suitable training and validation subsets. Data Mining encompasses a number of different approaches, such as clustering, data summarisation, learning classification rules, finding dependency networks, and anomaly detection. Data Mining is seen as a research frontier for both database research and machine learning. Many other AI based techniques, such as Natural Language Processing and Distributed AI methods, are also increasingly included (Fayyad et al., 1996d). Machine Learning can be said to be the use of *sophisticated* algorithms to generate and then process information, for eventual understanding. (Hu, 1995) characterises learning from a database as a triple $\langle$ D, C, A $\rangle$ where D is the data, C the *concept biases*, and A the language in which to phrase the definitions. He also notes that as a database stores no negative information induction should be performed cautiously to avoid over generalisation.

Data Mining has been defined as the application of algorithms, within the limits of computational efficiency, that produce a set of expressions E which represent patterns expressed in a well-defined language over a data set F (Fayyad et al., 1996d). There are a number of ways to represent E, including: association rules (Agrawal et al., 1993; Toivonen, 1996), rough sets (Ziarko, 1991; Alagar et al., 1993), temporal logic (Padmanabhan and Tuzhilin, 1996; Berger and Tuzhilin, 1998), and FDs (Kivinen and Mannila, 1995). This latter class consists itself of many well-developed approximation techniques including Fuzzy FDs (Bosc et al., 1994), PAC-approximation (Akutsu and Takasu, 1994; Kivinen and Mannila, 1995), and probabilistic approximations (Piatetsky-Shapiro and Matheus, 1993; Pfahringer and Kramer, 1995; Huhtala et al., 1998). To this category we add NDs for pattern expression (Collopy and Levene, 1998d). Their suitability to this task is shown via their satisfaction of *lattice properties* from which measures for approximation can be formed and the desirability of mining NDs from databases. Mined NDs, expressed as *cardinality constraints* (equivalent to NDs with empty left hand sides), have recently been used to reverse-engineer ER-models in (Soutou, 1998). We also develop a restricted temporal logic for discovery of patterns using NDs as our atoms (Collopy and Levene, 1998b).

To date knowledge discovery research has focused on application within relational databases containing definite information. However, the advanced functionality of DBMSs extend the set of data types beyond that of strict numerical or categorical data to include NULL value represen-

tation (Lipski, 1979; Imielinski and Lipski, 1984), the most common interpretation being that a value exists but we do not currently know what it is. The literature has extended this to handle indefinite or disjunctive information where a value might now be, for example, Tuesday or Wednesday, implying that we know the value is one of a finite set (Imielinski et al., 1991; Vadaparty and Naqvi, 1995). To allow DBMSs to handle scheduling and planning processing and querying directly the ability to store indefinite information is paramount. Data Mining techniques will then have to be extended to encompass these data types. We show for the case of NDs in (Collopy and Levene, 1998d) how definite instances (or possible worlds) of an indefinite relation satisfy different ND sets; other FD approximation methodologies can be applied similarly.

Data Mining algorithms are bounded both by potentially huge data sets and the limits of a computationally efficient methodology. Therefore, sampling and the use of randomised algorithms for data mining have been utilised (Kivinen and Mannila, 1994; Gunopulos et al., 1997; Collopy and Levene, 1998c). Sampling within data mining has been well studied (Kivinen and Mannila, 1994; John and Langley, 1996); to this we include the use of resampling for data mining. Resampling is a computationally intensive sampling methodology for non-parametric data; the distribution is unknown for most data sets. Its use allows for information about the distribution of data to be made and has a wider range of application than standard sampling. Many data mining algorithms seek to make inferences from sample data; classical statistics refers to this as *estimation.* Our work incorporates resampling processes (Efron, 1979; Efron and Tibshirani, 1986; Efron and Tibshirani, 1993) to achieve this. Heuristics are often required for knowledge to be discovered; in such cases randomised algorithms may allow the efficient processing of data for discovery. Indeed for a theory to be verified without error the complete data set needs to be examined though only one violating occurrence is required for falsification. Randomised algorithms and sampling allow for efficient analysis to be achieved utilising this fact. We use randomised algorithms and resampling to find approximate solutions to the *consistency problem*, known to be NP-complete, which is the problem of searching for a possible world within an indefinite relation that satisfies a given FD set (Vadaparty and Naqvi, 1995; Collopy and Levene, 1998c; Collopy and Levene, 1998d).

Temporal Databases have been a significant area of research in the last few years (Tansel et al., 1993; Clifford and Tuzhilin, 1995), unsurprising given the need for temporal support in real-world applications, particularly in the financial and medical domains. Building on this work is the rapid rise of, and need for, temporal data mining applications. Much of Temporal Knowledge Discovery relates to forecasting events in the future and analysing patterns which occur over time.

Indeed, these goals are closely related to time series analysis, a well established research field in statistics and econometrics (Enders, 1995; Nazem, 1988). Nearly all databases in use contain a temporal component and there has been much recent data mining work on temporal knowledge discovery (Agrawal et al., 1995; Padmanabhan and Tuzhilin, 1996; Berndt and Clifford, 1996; Berger and Tuzhilin, 1998).

Data mining and statistics have been substantially analysed (Fayyad et al., 1996a; Glymour et al., 1997). One of the prime goals of data mining is that of predicting values and this is inherently related to the representation of temporal data and *time series analysis*. Time series analysis when applied in a database is used for anything from identifying demand and modifying supply accordingly or for calculating patterns and changes in salary over time to predicting the expense of projects within different time periods. Until recently there was minimal use of time series analysis techniques within temporal databases (Schmidt et al., 1995).

The Classic multiplicative model views time series as containing four parts, namely trend, cycles, seasonal, and irregular patterns. Trends may be up or down and can be used to characterise the time series over a long time irrespective of short term fluctuations. Cycles display a recurring up and down movement around trend levels, including expansion and contraction. Seasonal patterns complete within a given time period. Finally, irregular patterns account for erratic changes in a time series and may be modelled as noise in the data.

The above may be understood using many techniques including trend moving averages, ratio-to-moving averages (for deriving the seasonal component), and difference equations for model representation. Using these time series may be extrapolated. Related issues are (1) granularity changes (2) application of moving windows, and (3) attribute value transformation. The temporal logic we present, as well as the related work we survey (Faloutsos et al., 1994; Laird, 1993; Agrawal et al., 1995; Das et al., 1997; Das et al., 1998), is closely linked with many aspects of (stationary) time series analysis. (Glymour et al., 1997) also tackles the issues of how data mining is extending and not simply repeating previous statistical research. There have been significant use of data mining algorithms incorporating statistical functions, not least in the scientific domain in applications as diverse as astronomical cataloguing through to geological sensing for earthquake detection (Fayyad et al., 1996a).

Recently there have been a number of different approaches to temporal data mining, stemming from machine learning work on pattern matching (Laird, 1993; Agrawal et al., 1995). There seems to be a demarcation between research based on data mining from temporal databases and research using time series as the input data set from which knowledge is to be discovered. Our

approach allows for discovery from either or a combination of the two, given that in each case only series of numbers change over time.

Temporal logic is used for temporal data mining in (Padmanabhan and Tuzhilin, 1996; Berger and Tuzhilin, 1998). It has been shown to be sufficient for expressing temporal relationships that have been discovered. If more complex relationships are required, such as, say, correlation between two data sets temporal logic does not have the functionality to express this without an explicit *correlation function*. Therefore we present a logic with statistical functionality so that such values are embedded within the logic. Given that the sentences of the logic express results of statistics we do not have to present *confidence* and *frequency* values (akin to work on association rules (Agrawal et al., 1993; Klemettinen et al., 1994; Holsheimer et al., 1995)) for the rules that we discover as our rule discovery itself is representative of specific statistical values. Our work on temporal relations is pattern focused; we do not attempt to discover a global model, the undoing of many time series analysis studies, but logical rules which describe local behaviour on subsequences of the temporal data set. Of course these can be, if desired, extended to global conditions. It is interesting to note that discovery in this sense is closely linked to the power of the query language. Similarly, we may view rule discovery using temporal logic as directly dependent on the expressiveness of the logic. Patterns within a (temporal) database may be referred to as *properties* which model the data. Temporal logic for property satisfaction is a well-researched area within program verification. Properties used to express the correctness of a temporal system also have application in data mining where a database with many states may be viewed as a temporal system. We claim that these properties are therefore suitable as *candidate patterns* for potentially interesting knowledge discovery.

The real-world desire for ever more information and knowledge precludes data mining from being anything but a significant research area. Many different mechanisms for expressing patterns have been developed and we believe that NDs, though not a panacea for expression within data mining, are widely applicable and easily understood. To illustrate, an ND $STUDENT \rightarrow^{5} COURSE$ in a timetable relation specifies that a student can take at most 5 different courses; it is clear that such data may often need to be represented and in current DBMSs this data would satisfy no built-in constraints. The progression towards a standardised query language for data mining (Chaudhuri, 1998) would benefit from their inclusion as we note their utility in different domains.

1.3 Main Results and Contribution

We provide a novel approach to data mining. We show how, in databases containing indefinite and temporal information, given an FD set F we form sets of approximations to F which may be satisfied for different definite instances of the indefinite relation or over time in a temporal relation. We choose to express these approximations as sets of NDs; other expressions may also be appropriate, such as a probabilistic approximation (Piatetsky-Shapiro and Matheus, 1993). In either indefinite or temporal databases we can use these sets to obtain statistics to determine how they may change in the indefinite relation or over time. We show how resampling can be applied to these sets to make inferences from the data. For indefinite relations we present a dynamic resampling process which allows for resampling on increasingly large sample sizes until an approximate fixpoint is reached. This provides an upper bound on sample size which is then used in a heuristic based hill-climbing algorithm. For temporal relation sequences we may take moving averages or create resampled sequences to determine how patterns, expressed in the form of *properties*, are satisfied at various time points.

We now outline our methodology as a general framework. We take a large set α of approximations to a given FD set and then apply resampling to α to draw conclusions on the nature of the data in the database. The exact method of the application of resampling, and the use of other statistical functions, differs with the type of data we are mining. This framework is applicable to other domains, extending the traditional mining approaches of simply using approximations to dependencies to infer information. Indeed, we assume that an FD set is provided by the user as a template in both indefinite and temporal domains; this FD set may be modified by a system user to compare results for different dependency sets. We shall demonstrate how it is possible to utilise temporal and indefinite domains from which the approximations, in our case NDs, are taken to make further discoveries from the relation which is being mined.

This thesis makes the following specific contributions:

1. NDs are shown to be effective and useful for data mining in that they provide a clear notion of proximity to FDs. NDs are shown to be able to efficiently and accurately approximate FDs in a relation with an easily understood semantics. An evolutionary database design procedure is introduced as a motivation for real-world ND applications as a precursor to their application in non-standard database domains. The lattice properties of NDs are exploited to provide a metric for data mining which we use in this work.

2. We provide a detailed study on an approach which uses NDs to provide approximations to the Consistency Problem, namely the NP-Complete problem of finding a definite world

that satisfies a set of FDs within a relation containing indefinite data.

3. Procedures for applying the Bootstrap, a resampling methodology (Efron and Tibshirani, 1993), within relations containing indefinite and temporal data are defined and shown to be useful via extensive simulations. They include a dynamic procedure for application of the bootstrap in indefinite relations for sample size determination.

4. A temporal logic for NDs is presented. This logic is then used for mining sequences of relations. We examine the sequences for proximity to FD set satisfaction, expressed as NDs, and compare this to standard time series analysis. The logic is transferable to standard time series and other linearly ordered numerical data sequences.

5. We present a model for the application of our temporal data mining system to a sequence of temporal relations. Results using financial time series of stock prices from the oil, finance and retail sectors are presented and analysed.

The thesis also presents a taxonomy of standard and temporal dependency data mining, placing our work in context, as well as making suggestions for future research.

1.4 Outline of the Thesis

After this introduction, Chapter 2 formally introduces the required relational database theory so that the remainder of the work is self-contained. All of the relevant theory presented is placed in the context of this research and related work. Additionally, we survey related data mining research, focusing on three areas:

1. We examine functional dependency data mining and the methods used to find approximations to FDs (Kivinen and Mannila, 1995; Akutsu and Takasu, 1994; Mannila and Räihä, 1992a; Savnik and Flach, 1993; Hale and Shenoi, 1995; Pfahringer and Kramer, 1995; Bell, 1995; Piatetsky-Shapiro and Matheus, 1993), of which using NDs is part of our contribution, shown in Chapter 3.

2. We briefly examine work conducted on indefinite information, related to our study of the consistency problem.

3. Temporal data mining research is discussed so that the reader is able to appreciate the contribution of the work in Chapters 5 and 6.

Chapter 2 concludes with a brief presentation of resampling in statistical applications, which is then expanded upon in Chapters 4 and 6.

Chapter 3 presents ND theory with regard to data mining, including a *chase procedure* for NDs. The chase procedure may be used to modify a relation to satisfy a given ND set allowing us to test whether or not an ND set implies a specific ND. We show how a data mining distance function is used for assessment, related to approximation work presented in Chapter 2. Additionally, research on applying NDs for mining within relations is discussed and compared with other approaches. We also present a practical database design tool for randomly evolving example relations which satisfy FD sets.

Chapter 4 then introduces the consistency problem and its applications. We present randomised algorithms which use NDs and the chase procedure for indefinite relations together with a novel application of resampling to determine sample size. Results of extensive simulations applied to randomly generated indefinite relations are examined. We also discuss the usefulness of the chase procedure as a heuristic.

Chapter 5 moves on to temporal data mining. We motivate the need for rules in temporal data mining, introduce our logic for temporal data mining, examine the logic and introduce the notion of temporal properties which we use in our temporal data mining environment. This logic is then assessed against a standard time series analysis which could be conducted on any time series data set and also on any temporal sequence of relations satisfying ND sets in each state over fixed intervals. Chapter 6 presents the details of our temporal rule discovery system, the use of resampling, and results from data sets studied, concluding with a discussion of future work together with an analysis of the utility of our temporal data mining approach.

Finally in Chapter 7 we give our concluding remarks and present a final discussion of the work, introducing avenues for further research and stating the open problems that remain.

1.5 Notation

We presented an index of the symbols used at the beginning of the thesis in the symbol index.

This thesis adheres to the standard notational convention generally followed in relational and deductive database texts, notably (Ullman, 1988). R refers to a relation schema, denoting a finite set of attributes, and r to a relation over R, denoting a finite set of tuples. Uppercase letters (possibly subscripted) refer to attributes if they are from the beginning of the alphabet such as A, B, C and to attribute sets if they are from the end of the alphabet such as X, Y, Z. Tuples are referred to by lowercase t and u (possibly subscripted).

Lowercase letters (possibly subscripted) refer to constants if they are from the beginning of the alphabet such as a, b, c and to variables if they are from the end of the alphabet such as x,y,z.

Predicate symbols (possibly subscripted) of arity ≥ 0 are referred to by p, q and r. We use $| X |$ to refer to the cardinality of set X and simply X to denote the singleton set $\{X\}$. The nonempty powerset of a set X is denoted by $\mathcal{P}(X)$. From the relational database literature, we refer to the union of two sets $X \cup Y$ by XY. The end of a definition or proof is denoted by $\square$.

CHAPTER 2

Relational Database, Data Mining, and Statistical Theory

The aims of this chapter are to provide the requisite background to be able to read the thesis as a self-contained body of work as well as enabling the reader to appreciate this research within the wider fields of both relational database theory and data mining.

In Section 2.1 we present the relationship of this work to both database and data mining theory. In Section 2.2 we introduce the relational database theoretic concepts relevant to this thesis and in Section 2.3 we introduce the area of data mining, concentrating firstly on dependency data mining so that the reader can fully appreciate the context of Chapter 3 and then temporal data mining for the background of Chapters 5 and 6. In later chapters we will refer to the definitions presented in 2.2 and 2.3 as and when they are initially used.

2.1 Database Theory for Data Mining

There has been significant work in the data mining community on the mining of *data dependencies*, both in standard (Piatetsky-Shapiro and Matheus, 1993; Kivinen and Mannila, 1995) and temporal environments (Bettini et al., 1996). Much of this concentrates solely on the discovery process, in effect working totally within a machine learning (ML) context, i.e. (Shen, 1991); scant regard is paid to the database theory upon which the dependencies are based. Though we do not question the quality of this work because of this omission we believe that NDs which fit into the relational model, both for design and, as we show in this thesis, data mining, are a valuable tool. Until recently, much data mining research was disjoint from database theory, based within statistics or machine learning though there is now a body of work on unifying these areas (Chaudhuri, 1998); this thesis requires an appreciation of both. We introduce the background material on database theory in Section 2.2 to clarify later work on Armstrong relations and the chase procedure, a theorem proving tool for FDs in a relation, as well as our use of indefinite information

for the consistency problem. Theoretical work on FD behaviour has directly led to the creation of numerous data mining methodologies which we introduce in 2.3.1. Section 2.2 concludes with a presentation of temporal databases and dependencies.

Section 2.3 introduces aspects of dependency data mining, including a discussion of the relationship to NDs. We provide a discussion of measures based on aspects of FD theory in 2.3.1. We then introduce temporal databases and dependencies before moving on to temporal data mining and rule discovery from time series, closely related to work in Chapters 5 and 6. This section concludes with a brief overview of sampling in data mining followed by an informal introduction to resampling, useful for later work presented on indefinite and temporal relations.

2.2 Relational Database Theory

We now present the relational database theory required within this thesis. The reader is referred to (Abiteboul et al., 1995; Atzeni and De Antonellis, 1993; Maier, 1983; Ullman, 1988) for a complete coverage of the area.

2.2.1 The Relational Model

In 1970, E. F. Codd introduced the relational model (Codd, 1970), with relations as the data structure, so that database users need not concern themselves with the physical storage of data. This allowed independence between programs and their machine representations by providing a sound basis for describing the structure of data and operations for data manipulation without the need for consideration of the internal machine representation. Subsequently other data models have been developed, including the Entity-Relationship model, for high level conceptual database modelling, and object-oriented data models (Kim, 1990; Abiteboul et al., 1995). The latter were primarily developed to combat the growing requirements for complex data manipulation; we do not make further reference to these data models and remain within the confines of the relational model in this thesis. Its universality and ease of data manipulation does not require further justification. We now formalise the relational model.

Definition 2.2.1 (Universe) A universe $\mathcal{U}$ is a finite, fixed set of symbols that represent the column names which can appear within a relation. They are referred to as attributes. □

Definition 2.2.2 (Attribute Domain) The domain of an attribute $A \in \mathcal{U}$, denoted by DOM(A), is the countable set of possible values which can be members of A. This is the set of values which can appear in a column of A. □

Definition 2.2.3 (Relation Schema and Relation) A relation schema R is a subset of the universe $\mathcal{U}$. The elements of a relation schema are denoted by $\{ A_1, \ldots, A_n \}$. A *tuple* over R is an

element of DOM(A_1) $\times \ldots \times$ DOM(A_n), where $\times$ refers to the cartesian product. An instance of a relation over R is a finite set of tuples defined over R. □

A relation consists of a finite set of tuples where each tuple represents an entity. A relation is therefore simply an entity set. Each tuple can be considered a row if we assume the table representation of a relational database.

Definition 2.2.4 (Database Schema and Database) A Database Schema over R is a finite set of relation schema { $R_1, \ldots, R_n$ }. A database over R is a finite set $d = \{r_1, \ldots, r_n\}$ such that each $r_i \in d$ is a relation over $R_i \in R$. □

The relational algebra is presented by Codd (Codd, 1970) in the context of deriving desired result relations from other relations. The operations include *selection*, *projection*, defined below, *join*, *union*, *difference*, and *renaming*; all are defined in (Abiteboul et al., 1995; Atzeni and De Antonellis, 1993; Date, 1995; Maier, 1983; Ullman, 1988).

Definition 2.2.5 (Projection) The projection of an R-tuple t onto a set of attributes $Y \subseteq R$, denoted by $t[Y]$ (also called the Y-value of t), is the restriction of t to the attributes in Y. The projection of a relation r onto Y, denoted as $\pi_Y(r)$, is defined by $\pi_Y(r) = \{ t[Y] \mid t \in r \}$. □

We now move on to the representation of constraints in the relational model required to ensure the maintenance of integrity within a database. Data mining now often uses such constraints and constraint approximations to discover previously unknown and non-trivial information (Fayyad et al., 1996d).

2.2.2 Functional Dependencies

Integrity constraints, or data dependencies, allow a database to have associated with it an intended meaning or semantics for the tuples within the database. The most common constraint is the FD introduced in (Codd, 1972), its prevailing application in practice is as a key dependency. FDs were given a *sound* and *complete* axiomatisation in (Armstrong, 1974). We note that *soundness* implies that each dependency, which is derived using a finite number of applications of an axiomatisation from a given set, holds. *Completeness* implies that valid each dependency which holds can be derived using the axiomatisation. FDs are restricted first order logic (FOL) sentences shown in (Sagiv et al., 1981) to be equivalent to Horn clause statements, relating determinations to logical implication (Fagin, 1977; Lloyd, 1987; Makowsky, 1987). There has been extensive work on the theory of FDs, of which some seminal contributions are (Armstrong, 1974; Fagin, 1977; Beeri and Bernstein, 1979; Sagiv et al., 1981). Although work on FD theory has somewhat exhausted itself there has recently been extensive work in data mining for approximating

FDs (Mannila and Räihä, 1992a; Savnik and Flach, 1993; Bell and Brockhausen, 1995; Huhtala et al., 1998).

Definition 2.2.6 (Data Dependency) A data dependency is a restricted integrity constraint incorporating a (specified) property that is to be satisfied by all instances of the database schema. □

Dependencies within the relational model allow for the incorporation of a more complex semantics via meta-data representations. We now formalise the FD, its axiom system, and the closure of FD attribute sets.

Definition 2.2.7 (Functional Dependency (FD)) A *functional dependency* over R (or simply an FD) is a statement of the form $X \rightarrow Y$, where $X, Y \subseteq R$. □

F *is known as a set of FDs over* R and $X \rightarrow Y$ *is a single FD over* R. We denote logical implication by $\models$. A key dependency is an FD of the form $X \rightarrow R$ for some $X \subseteq R$.

Definition 2.2.8 (Satisfaction of an FD) Given r, a definite relation over R, an FD $X \rightarrow Y$ is *satisfied* in r, denoted by $r \models X \rightarrow Y$, whenever $\forall t_1, t_2 \in r$, if $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$. A set of FDs F is *satisfied* in r, denoted by $r \models F$, whenever $\forall\, X \rightarrow Y \in F$, $r \models X \rightarrow Y$. □

FDs obey a set of axioms, shown to be sound and complete in (Armstrong, 1974), which are:

Definition 2.2.9 (Armstrong's Axioms for functional dependencies) Given a relation schema R and $X, Y, Z \subseteq R$:

Reflexivity If $Y \subseteq X$, then $X \rightarrow Y$

Augmentation If $X \rightarrow Y$ then $XZ \rightarrow YZ$

Transitivity If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$ □

Definition 2.2.10 (Closure of an attribute set) Given a set F of FDs over a set of attributes X in schema R the closure of X under F, denoted X^+, is the set $\{ A \in R \mid F \models X \rightarrow A \}$. □

X_F^+ refers to the closure of X with respect to F, that is the set of all attributes $A \in R$ such that $X \rightarrow A$ holds in F. We define F^* to be the closure of F such that trivial FDs of the form $X \rightarrow Y$, where $Y \subseteq X$, are excluded.

Definition 2.2.11 (Non-trivial closure) $F^* = \{ X \rightarrow Y \mid XY \subseteq R \text{ and } Y \subseteq X_F^+ - X \}$. □

Definition 2.2.12 (Closure of a set of attribute sets) Given an FD set F we denote the closure of all possible attribute sets under F by CL(F). This is defined as CL(F) = { X | X $\subseteq$ R and X_F^+ = X }. □

Note that the schema R is always included in the closure of attribute sets for any FD set. The next lemma shows that $F^\star$ and CL(F) are equivalent in characterising a set of FDs F. The non-trivial closure is relevant in data mining measures, discussed in Section 2.3.1.

Lemma 2.2.1 Given two sets of FDs, F and G, we then prove $G^\star \subseteq F^\star \equiv CL(G) \supseteq CL(F)$

Proof. (if) Assume, to the contrary, that CL(G) $\not\supseteq$ CL(F). Therefore $\exists$ X $\in$ CL(F) such that X $\notin$ CL(G), implying that X is not closed in G. Then $X_G^+ = XY$, for some attribute set Y. This implies that X $\rightarrow$ Y is in G but not in F, yet $G^\star \subseteq F^\star$, leading to a contradiction.

(only-if) Assume, to the contrary, that $G^\star \not\subseteq F^\star$. Then $\exists$ X $\rightarrow$ Y $\in G^\star$ such that X $\rightarrow$ Y $\notin F^\star$. Then Y $\subseteq X_G^+$ but Y $\not\subseteq X_F^+$, and so $X_G^+ \neq X_F^+$. Given that CL(G) $\supseteq$ CL(F) it must be the case that any closed set in F must be closed in G, and so we have a contradiction. □

Definition 2.2.13 (Closure of a set of FDs) Given an FD set F we denote the closure of F by F^+. This is defined as

$$F^+ = \{X \rightarrow Y \mid XY \subseteq R \text{ and } F \models X \rightarrow Y\} \qquad \square$$

An algorithm to compute the closure of a set of FDs is in (Abiteboul et al., 1995; Atzeni and De Antonellis, 1993) which runs in time linear to the size of the set FDs. The concept of a maximal set is now introduced; its data mining applications will be briefly discussed in Section 2.3.1 and Chapter 3.

Definition 2.2.14 (Maximal Set) Given X, a subset of schema R, and A $\in$ X then a set Y $\subseteq$ X is a *maximal* set for A, if F $\not\models$ Y $\rightarrow$ A and for any Z $\subseteq$ X such that Y $\subset$ Z we have F $\models$ Z $\rightarrow$ A. □

Definition 2.2.15 (The set of all maximal sets) max(F,X,A) = { Y $\subseteq$ X | Y is a maximal set such that F $\not\models$ Y $\rightarrow$ A }. If F is understood from the context then it is written simply max(X,A); max(X) denotes the union of max(X,A) where A $\in$ X. □

A maximal set is an attribute set X which for some attribute A is a largest possible set *not* determining A. We also define generator sets. The generator function, GEN, omits those sets from the closure of an attribute set which can be formed by the intersection of other sets in the closure to obtain a more concise representation. Theorem 13.1 of (Mannila and Räihä, 1992a) shows that max(X) = GEN(X).

Definition 2.2.16 (The generator function) The generator function produces a set for X such that GEN(X) = { Y ∈ CL(X) | Y ⊂ ⋂{ W ∈ CL(X) | Y ⊂ W }} □

We now define the cover of a set of dependencies, useful for discovering equivalent FD sets.

Definition 2.2.17 (Cover of a set of Dependencies) Given sets F and G of FDs, F is a *cover* of G if $F^+ = G^+$. A *cover* G is minimal for F if there does not exist a cover H of F such that | H | < | G |. A minimal cover is necessarily nonredundant, that is, $\forall d \in G$ we have $G \setminus \{d\} \not\models d$, though nonredundancy does not imply minimality. □

Example 2.2.1 From (Mannila and Räihä, 1992b), the set F = { A → BC, B → AD, CD → E, E → CD } and the set G = { A → BE, B → A, CD → E, E → CD } are equivalent. This is proven by showing that G ⊨ F and F ⊨ G. The non-trivial cases are showing that F ⊨ A → E and G ⊨ { A → C, B → D }. To illustrate, A → C may be shown to hold from G as we know A → E holds, by transitivity A → CD holds, and therefore A → C is known to be satisfied by G.

To test if two covers, F and G, are equivalent we can check that every X → Y ∈ F is satisfied in G and vice versa. Using the algorithm presented in (Mannila and Räihä, 1992b) this can be done in time $O(|F| \, \| G \| + |G| \, \| F \|)$, where || X || denotes the number of attributes in X including repetitions. Alternatively, we can check for equivalence of the maximal sets. We now introduce some notation to aid the reading of the next section.

Definition 2.2.18 (Agreement set of two tuples) Given a relation r over R, where t_1,t_2 are two tuples in r the agreement set is defined as $ag(t_1, t_2) = \{ B \in R \mid t_1[B] = t_2[B] \}$.
The disagreement set is defined dually, disag(t_1,t_2) = $R \setminus ag(t_1, t_2)$. □

Given an attribute A in disag(a,b) let X be the disagreement set of all attributes for tuples a and b apart from A, i.e. X = disag(a,b) \{A}. Then any set in the left-hand side of A must contain at least one attribute of X. Why is this so? Let us assume that it does not hold and that for a member of the left-hand side of A an attribute of X is not contained. This implies, however, that there exist two tuples which disagree on A when they have the same left-hand side. Obviously this violates F and so is not the case. X is therefore said to be a necessary set for A, used in dependency mining (Mannila and Räihä, 1992a).

Definition 2.2.19 (Agreement set of a relation) Given a relation r over attribute set R, the agreement set is defined as $agr(r) = \{ag(t_1, t_2) \mid t_1, t_2 \in r\}$. □

We now define Armstrong Relations (AR) and follow this with a discussion of database design and its relationship to data mining.

2.2.3 Armstrong Relations

(Armstrong, 1974) introduced the concept of an *Armstrong relation*:

Definition 2.2.20 (Armstrong Relation) An Armstrong relation for F is a relation r which satisfies F^+ and is such that for every FD $\sigma \notin F^+$ for which $F^+ \not\models \sigma$, then r violates σ. □

In theory, Armstrong relations (Fagin, 1982; Beeri et al., 1984; Demetrovics and Thi, 1995; Gottlob and Libkin, 1990; Levene, 1995; Mannila and Räihä, 1986) serve as "ideal" example relations, since they satisfy exactly the set of all logical consequences of the set of FDs specified, say F. Thus an Armstrong relation provides an example for all FDs that are logically implied by F and a counterexample for all those FDs that are not logically implied by F. One of the problems with Armstrong relations is that, in general, their cardinality is exponential in the size of F and the set of attributes, R, over which F is defined (Beeri et al., 1984). An Armstrong relation for a set of FDs, if deterministically generated (Mannila and Räihä, 1992a), always provides the same resulting relation. It would be highly desirable if varying Armstrong relations of different domain and tuple sizes may be generated as a side effect of the forming of example relations.

(Fagin, 1982) presents a survey of Armstrong Databases including descriptions of the techniques for generating Armstrong Relations from a set of FDs. These are: (1) Use *disjoint union* to create an isomorphic copy of each relation and then form the union of all of the tuples in all of the relations. For each FD σ which is not a logical consequence of the relations create a relation r_σ which obeys F but not σ. Then form the union for all *standard* FDs, where the left hand side is non-empty, to give an AR. (2) Create *agreement sets*. The agreement set is formed such that GEN(F) $\subseteq$ $agr(r)$ $\subseteq$ CL(F). (Beeri et al., 1984) construct an Armstrong relation by firstly computing the closure of the FD set F, CL(F), and then constructing a relation such that $agr(r)$ = CL(F). (3) *Direct products* (used by (Grant and Minker, 1985b) to prove no Horn clause representation exists for NDs). A relation is created for each σ outside of CL(F) which violates σ and satisfies CL(F). The direct product of these is then formed. (4) Use the *chase procedure*, presented in Section 2.2.5. Given an Armstrong relation which obeys an FD set F and violates all FDs outside of CL(F) form a model where all FDs are violated using the chase which can cause new tuples and/or constants to be added to the database. We shall see in Chapter 3 how an evolutionary technique using mutation and guided by ND satisfaction may often generate Armstrong relations (Collopy and Levene, 1996; Collopy and Levene, 1998a).

(Fagin and Vardi, 1983) shows that an Armstrong database may be generated for a set of inclusion dependencies and standard FDs. An inclusion dependency states that if some combination of values occurs in one part of a database it must also occur in another part.

Lemma 3.1 (Beeri et al., 1984) shows that if Σ is a set of FDs and σ a single FD such that $\Sigma \not\models \sigma$ then there exists a two tuple relation that obeys Σ but not σ. A by-product of this result is that it is always possible to add a tuple to a relation r satisfying Σ which violates σ. A deficiency of deterministic processes for AR generation are that only one specific Armstrong relation is ever returned for a given FD set. (Beeri et al., 1984) present an analysis on the upper and lower bounds of the size of an Armstrong Relation based on the number of distinct entries in the relation, referred to as the generator sets which (Mannila and Räihä, 1986) later refine. (Mannila and Räihä, 1986) show that the size of a minimal Armstrong relation for a normalised scheme R depends strongly on the number of keys for R. The possible exponential size of a minimal Armstrong relation depends only on the number of dependencies, and not on the number of attributes.

An Armstrong relation should be as small as possible, as should the set of values used, though the smaller the relation the more difficult it becomes for the designer to locate all of the anomalies as opposed to an Armstrong relation which lists all examples of dependency violations in a pairwise format.

2.2.4 Relational Database Design

We now mention relational database design related to work presented in Chapter 3. Informally, database design attempts to remove redundancy and facilitate querying by the use of normalisation. A relation can be constructed to adhere to a series of increasingly restrictive normal forms introduced so as to prevent redundancy and (update) anomalies within the database, discussed in (Codd, 1972; Abiteboul et al., 1995; Atzeni and De Antonellis, 1993; Date, 1995; Maier, 1983; Ullman, 1988).

Keys provide the only method for tuple identification in the standard relational model, and they are therefore central to the retrieval of information and good database design. There are many key related properties whose determination is computationally intractable (Lucchesi and Osborn, 1978). We now present the superkey class, used within Boyce-Codd Normal Form.

Definition 2.2.21 (SuperKey) Given a relation scheme R and a set Σ of FDs which apply to it, a set of attributes X is a superkey for R if the FD $X \rightarrow R \in \Sigma^+$. □

Definition 2.2.22 (Boyce Codd Normal Form) Given a relation scheme R and a set of FDs Σ which apply to it, R is in Boyce Codd Normal Form (BCNF) if for every non-trivial FD $X \rightarrow A \in \Sigma^+$, X is a superkey. □

We assume that all relations discussed in this thesis satisfy first normal form (1NF), where each relation is flat, and present a database or relation satisfying BCNF as the ideal normal form,

where each non-trivial FD has a superkey as its left hand side. BCNF attempts to overcome the deficiencies in 3NF by dropping the constraint that *non-prime* attributes, those not in any key, which are allowed on the right hand side of FDs may violate the normal form . (Beeri and Bernstein, 1979) present an analysis method to achieve a BCNF relation by splitting relations successively which violate BCNF. No such procedures exist which are guaranteed to be constraint preserving. A non-mathematical treatment of normal forms is given in (Kent, 1983) which are then extended for temporal relations in (Jensen et al., 1992).

(Silva and Melkanoff, 1981) introduced the idea of example relations generated from a set of FDs and MVDs for database design purposes. Example relations give the database designer a guide to the information within a relation associated with a given set of dependencies. (Silva and Melkanoff, 1981) formalise a design technique which attempts to provide the database designer an iterative method of obtaining the FD set which most *characterises* a relation. More recently (Mannila and Räihä, 1986; Mannila and Räihä, 1992b), approached various database design problems with the goal of formalising methods and tools to produce schemas with specific properties. They introduce the technique of using example relations within the design process, notably as "an application of ARs", by presenting an algorithm to deterministically generate ARs for the benefit of the database designer. (Beeri et al., 1984) note how an Armstrong relation, perhaps generated automatically from a set of FDs, is of much use in the design process from an application point of view. Automated database design has been seen as a goal for dependency theory (Beeri and Vardi, 1984).

In (Collopy and Levene, 1998a), summarised in Chapter 3, we present a probabilistic extension of this work, allowing the database designer to view many different example relations, though not necessarily ARs, for any given FD set specified over R. The size of the relation is governed by the database designer. (Mannila and Räihä, 1986) state, "A good example relation should not leave the designer any illusions about what can be stored in the database." Our algorithm for generating example relations achieves this. It is based on the following loop which we envisage during the database design process:

1. The database designer specifies a set of FDs, F, the maximum number of tuples in the example relation, m, and the maximum domain size, d, for a relation. (The designer has the options of specifying m and d so that relations of different structure can be viewed.)

2. A random example relation satisfying F, having at most m tuples, and a domain ranging from 2 to d values is generated. The quality, in terms of its proximity to that of an Armstrong relation, for the FD set is measured and returned to the designer.

3. The database designer either accepts F or modifies the parameters F, m and d, and then returns to step (2).

Two aspects of this work are discussed subsequently; the evolutionary hill climbing algorithm which uses NDs in a hill climbing fashion to obtain a relation satisfying an FD set is presented in Chapter 3 and the *quality* function used to obtain a proximity to an Armstrong relation for the output, which may be viewed as the data mining component of this work, is introduced in 2.3.3.

2.2.5 The Chase Procedure

If we have an attribute set R, an FD set F over R and a relation r which does not satisfy F, $r \not\models$ F, we can use the chase procedure to modify r so that it satisfies F. This technique is known as the chase, introduced in (Maier et al., 1979) and generalised to tuple and equality generating dependencies in (Beeri and Vardi, 1984). We assume in algorithm 1, without loss of generality, that our domains are *linearly ordered.*

Algorithm 1 (CHASE(r, F))
1. **begin**
2. Result := r;
3. Tmp := $\emptyset$;
4. **while** Tmp $\neq$ Result **do**
5. Tmp := Result;
6. **if** $\exists$ X $\rightarrow$ Y $\in$ F and $\exists t_1, t_2 \in$ Result such that t_1[X] = t_2[X] but t_1[Y] $\neq$ t_2[Y] **then**
7. $\forall A \in$ Y−X, t_1[A], t_2[A] := max(t_1[A], t_2[A]);
8. **end if**
9. **end while**
10. **return** Result;
11. **end.**

Figure 2.1: The Chase procedure for FDs

Given that NDs and FDs are expressible in First-Order Logic (FOL) any FOL proof procedure may be applied. The chase is however suitably specialised for FDs avoiding costly theorem proving procedures. Additionally, the chase procedure is a decision procedure in that it always halts. We can discover the closure of a set of attributes X by creating a two-tuple relation which agrees on X and disagrees on all other attributes. After the chase procedure halts (proven to occur in (Maier et al., 1979)) the agreement set in the relation r consists of exactly the closure of X. We illustrate this with a small example for the FD set F = { T $\rightarrow$ H, H $\rightarrow$ C } and the relation r_1 in Table 2.1. We wish to obtain the closure of T, shown in Table 2.2 after application of the chase where T^+ = THC.

T	H	C
2	1	1
2	2	4

Table 2.1: r_1 before the chase

T	H	C
2	2	4

Table 2.2: r_1 after the chase, T^+ = THC

In Chapter 3 we generalise the concept of the equality generating chase to cover NDs and in Chapter 4 we extend it to accept relations which contain indefinite information. Examples of the chase in use are given in (Abiteboul et al., 1995; Mannila and Räihä, 1992a). In (Lerat, 1986) it is noted that the chase turns a database consisting of extensional and intensional data into one containing extensional data only.

2.2.6 Numerical Dependency Theory

(Grant and Minker, 1985b; Grant and Minker, 1985a) introduced the concept of NDs as extensions of FDs for providing the database designer with additional flexibility. They have a clear intuitive semantics which can easily be accommodated for many database representation issues, including cardinality constraints.

Definition 2.2.23 (Numerical Dependency (ND)) A *numerical dependency* over R (or simply an ND) is a statement of the form $X \rightarrow^k Y$, where X, Y $\subseteq$ R and $k \geq 1$. □

We let N be *a set of NDs over* R and $X \rightarrow^k Y$ *is a single ND over* R, with $k \geq 1$. Intuitively, an ND $X \rightarrow^k Y$ is satisfied in a definite relation r over R, if each X-value in r is associated with at most k Y-values in r; when $k = 1$ then the ND $X \rightarrow^1 Y$ reduces to the FD $X \rightarrow Y$. For an ND $X \rightarrow^k Y$ we refer to k as the branching factor. The satisfaction of $X \rightarrow^k Y$ with $k > 1$ is equivalent to the satisfaction of a Functional Independency (Gottlob and Libkin, 1990). NDs are generalisations of FDs which allow an attribute set to uniquely determine up to k different attribute set values, noting that $k = 1$ in the case of FDs. For any given FD set F and a relation r the set of all possible approximations forms a *complete lattice* (Davey and Priestly, 1990); this is the basis for a metric we define in Chapter 3 and use for how well an ND set approximates F in Chapter 4.

Definition 2.2.24 (Satisfaction of an ND) Given a definite relation r over R, an ND $X \rightarrow^k Y$ is *satisfied* in r, denoted by $r \models X \rightarrow^k Y$, whenever $\forall t_1, t_2, \ldots, t_k, t_{k+1} \in r$, if $t_1[X] = t_2[X] = \ldots = t_k[X] = t_{k+1}[X]$ then $\exists i, j$ such that $1 \leq i < j \leq k+1$ and $t_i[Y] = t_j[Y]$. A set of NDs N is *satisfied* in r, denoted by $r \models N$, whenever $\forall\, X \rightarrow^k Y \in N$, $r \models X \rightarrow^k Y$. □

We now present an example of the application of NDs in Table 2.3 in a teaching relation $PLAN$(Lecturer, Course). The intended semantics for this relation is that a lecturer can teach up to, but not more than, 2 different courses, written as Lecturer $\rightarrow^2$ Course.

Lecturer	Course
Mark	C320
Robin	B11a
Robin	B151
Mark	B151
Sean	C340

Table 2.3: relation $PLAN$(Lecturer, Course)

We now define cardinality constraints and show in lemma 2.2.2 that cardinality constraints restricted only by upper bounds are equivalent to NDs with empty left hand sides.

Definition 2.2.25 (Cardinality Constraint) A *cardinality constraint* over R (or simply a CC) for an attribute set $X \subseteq R$ is a statement of the form $c_1 \leq | \pi_X(R) | \leq c_2$ where c_1 and c_2 are constants. A cardinality constraint is satisfied if the formula holds. The formula may be restricted to just having either an upper (c_2) or lower (c_1) bound. □

Cardinality constraints were introduced in (Kanellakis, 1980). (Liddle et al., 1993) surveys cardinality constraints and shows their widespread application in numerous data models.

Lemma 2.2.2 A cardinality constraint $| \pi_X(R) | \leq c$ is equivalent to the ND $\emptyset \rightarrow^c X$.

Proof. Trivial, given that $\emptyset$ is a unique partition. □

Cardinality constraints are applied, as restricted NDs, in Chapter 6. NDs were themselves generalised to branching dependencies in (Demetrovics et al., 1992). Informally, a *branching dependency* over R is a statement $X \overset{(p,q)}{\rightarrow} Y$ which states that there do not exist $q+1$ different tuples such that for at most p different values on X there are not $q+1$ different values on Y.

Definition 2.2.26 (Branching dependency) A *branching dependency* over R (or simply a BD) is a statement of the form $X \overset{(p,q)}{\rightarrow} Y$ where $X, Y \subseteq R$ and $p \geq 1, q \geq 1$. □

Definition 2.2.27 (Satisfaction of a BD) Given a definite relation r over R, a BD $X \overset{(p,q)}{\rightarrow} Y$ is *satisfied* in r, denoted by $r \models X \overset{(p,q)}{\rightarrow} Y$, whenever $\forall t_1, t_2, \ldots, t_q, t_{q+1} \in r$, if $| \{ t_1[X], t_2[X], \ldots, t_{q+1}[X] \} | \leq p$ then $| \{ t_1[Y], t_2[Y], \ldots, t_{q+1}[Y] \} | \leq q$. □

Note the special cases of branching dependencies where $p = 1$ such that the BD is $A \overset{(1,q)}{\rightarrow} B$ is equivalent to a standard ND and when $p = 1, q = 1$ such that $A \overset{(1,1)}{\rightarrow} B$ then this is equivalent to a FD. We now present an example of a BD:

Example 2.2.2 In Table 2.3 the BD Lecturer $\overset{(2,2)}{\rightarrow}$ Course is violated. This is highlighted in the first three tuples where we have Robin and Mark as the lecturers for { C320, B11a, B151 }. Lecturer $\overset{(2,2)}{\rightarrow}$ Course implies that at most two Lecturers teach at most two courses. Table 2.3 satisfies Lecturer $\overset{(2,3)}{\rightarrow}$ Course.

The following lemma is a restatement of lemma 3.2 of (Demetrovics et al., 1992). Based on this we do not consider BDs any further within the mining process due to all cases satisfying NDs.

Lemma 2.2.3 Any BD X $\overset{(p,q)}{\rightarrow}$ A satisfied in a relation r also satisfies X $\overset{(1,q)}{\rightarrow}$ A.

Proof. No single *partition* on X contains more than q different values, therefore the relation $r \models \mathrm{X} \rightarrow^q \mathrm{A}$. □

In the sequel we frequently refer to partitions on attributes, implying the semantics of Definition 2.2.28, and we define *mean NDs* in Section 3.1.3 to be the sum of branching factors for an ND in all partitions divided by the number of partitions. We define an ND X $\rightarrow^k$ A to be *vacuously satisfied* if there does not exist a block $\mathcal{B} \in r$ with $\mathcal{B}$ having at most k different values on A. We define the *size* of a set or NDs N to be the number of attributes appearing in N including repetitions.

Definition 2.2.28 (Partitioning of a relation) The *partitioning* of a relation r with respect to the ND X $\rightarrow^k$ A, is the partition $\{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_w\}$ of r, such that for each X-value, $x \in \pi_{\mathrm{X}}(r)$, there exists exactly one *block* $\mathcal{B}_i$ in the partition having the single X-value x, i.e. such that $\pi_{\mathrm{X}}(\mathcal{B}_i) = \{x\}$. We denote the block whose X-value is x by $r[\mathrm{X}, x]$. The projection on X of $\mathcal{B}_i$ is $\pi_{\mathrm{X}}(\mathcal{B}_i) = \{t[\mathrm{X}] \mid t \in \mathcal{B}_i\}$. □

In Chapter 3 we focus on the theory of NDS, NDs for data mining, and for NDs in a database design context. We now move on to a general outline on indefinite information in databases, the background for the work on the consistency problem in Chapter 4.

2.2.7 Indefinite Relations

Lipski's 1979 paper (Lipski, 1979) formalised many of the methods for representing incomplete information within a database. Incomplete information theory must formalise the relationship between the external and internal representations of knowledge, the former corresponding to the real world and the latter to the database representation of it. (Abiteboul et al., 1995) define a database with incomplete information as a set of possible worlds where the table contains null values that may be replaced by domain value sets. The set of possible worlds of an incomplete

database, given a table T, a relation with NULL values, is defined by (Abiteboul et al., 1995) as $rep(T) = \{v(T)|v$ is a valuation of variables in $T\}$. In this thesis, incomplete relations are restricted to *indefinite relations* where a cell c may contain a set of values, denoting a disjunction of the values in c.

OR-objects (Imielinski et al., 1991) are a generalisation of *marked nulls*. An unmarked null value states that the value exists but is at present unknown. Null values with identifiers also allow for comparison between nulls. Additionally, OR-objects can be viewed as expressing disjunction which can be applied in many applications. Frequently a particular attribute value may be known to be one of a number of options though it may be unknown precisely which one, possibly until a later date or inference from data dependencies which are known to hold. For example we may wish to express the fact that *Ship 23* sets sail from either *Dover* or *Portsmouth*. This is achieved using an OR-object, o_1, inside a tuple, such as t(*Ship 23*,o_1) with the domain of o_1, $Dom(o_1)$ = $\{Dover, Portsmouth\}$ to represent the disjunction. OR-objects are introduced in (Imielinski et al., 1991) where formalisations are presented for querying databases that contain OR-objects either against the possible worlds or the database, containing OR-objects, itself and details of a practical application for scheduling are provided.

Definition 2.2.29 (OR-Object) An OR-object, o_1, refers to a finite domain set of values, entitled $Dom(o_1)$, that is a disjunctive set where each element may replace the OR-object to obtain an instance, or possible world, of the database. A database containing OR-objects is called an OR-database. □

Definition 2.2.30 (Possible World) A possible world W of an OR-database D with a set of OR-objects O is obtained by replacing every OR-object $o \in O$ with a value from the respective $Dom(o)$. □

Definition 2.2.31 (Conforming World) A possible world W of an OR-database D is conforming with respect to a set of FDs F if it satisfies every $f \in$ F. If a world W violates at least one $f \in$ F it is said to be *non-conforming*. □

Definition 2.2.32 (Redundant Element) A member $c \in Dom(o_1)$ of an OR-object o_1 is redundant under a set of FDs F if every possible world that assigns c to o is a non-conforming possible world with respect to F. □

(Vadaparty and Naqvi, 1995) present a number of algorithms using OR-objects to improve query optimisation processes via the processing of OR-objects with respect to the set of FDs F

that hold for a database. In Chapter 4 we compare a pre-processing algorithm for a relation with OR-objects to the chase algorithm we define for indefinite relations.

In the context of this thesis we refer to OR-objects as indefinite cells; they are equivalent to OR-objects. Related work on FDs in relations with incomplete information, using NULL values is presented in (Levene and Loizou, 1998; Levene and Vincent, 1997). Another interpretation for indefinite information semantics in the relational model is that of it being a probabilistic relation, which we now formalise. A probabilistic interpretation allows for the likelihood of possible worlds to be calculated.

Definition 2.2.33 (Indefinite relation) Let $\mathcal{D}$ be a countable set of domain values. An *indefinite tuple* t over R is a total mapping from R into $\mathcal{P}(\mathcal{D})$ such that $\forall$ A $\in$ R, t(A) $\in \mathcal{P}(\mathcal{D})$. A tuple t over R is *definite* if $\forall$ A $\in$ R, $\mid t(\mathrm{A}) \mid = 1$, i.e. t(A) is a singleton. □

An *indefinite relation* over R is a finite (possibly empty) set of indefinite tuples over R. A relation r over R is *definite* if all of its tuples are definite. In the following definitions we assume a uniform distribution and stochastic independence of tuples.

Definition 2.2.34 (The probability of a tuple) The probability of a value $v \in S$, where $S \in \mathcal{P}(\mathcal{D})$, denoted by $p_S(v)$, is $\frac{1}{|S|}$.

The set of *possible* definite tuples of an indefinite tuple t, denoted by POSS(t), is the set of tuples given by $\{u \mid u$ is definite and $\forall \mathrm{A} \in \mathrm{R}, u[\mathrm{A}] \in t[\mathrm{A}]\}$. The *probability* of a tuple $u \in$ POSS(t), denoted by $p_t(u)$ is given by $p_t(u) = \prod_{\mathrm{A} \in \mathrm{R}} p_{t[\mathrm{A}]}(u[\mathrm{A}])$. We observe that $\sum_{u \in \mathrm{POSS}(t)} p_t(u) = 1$. □

Definition 2.2.35 (The probability of a relation) The set of *possible* relations (or *possible worlds*) of a relation $r = \{t_1, t_2, \ldots, t_n\}$, denoted by POSS($r$), is the set of relations given by $\{s \mid s = \{u_1, u_2, ..., u_n\}$ and $u_1 \in \mathrm{POSS}(t_1), u_2 \in \mathrm{POSS}(t_2), \ldots u_n \in \mathrm{POSS}(t_n)\}$.

The *probability* of a relation $s \in$ POSS(r), denoted by $p_r(s)$, is given by

$$p_r(s) = \prod_{u \in s} p_t(u), \text{ where } u \in \mathrm{POSS}(t) \text{ and } t \in r$$

We observe that $\sum_{s \in \mathrm{POSS}(r)} p_r(s) = 1$. □

In Tables 2.4 and 2.5 we see indefinite data in a relation. The OR-object model uses the labels o_1 and o_2 to denote or-object sets, equivalent to $\{1, 2\}$ and $\{3, 6\}$, respectively. Tables 2.6 and 2.7 represent, respectively, non-conforming and conforming possible worlds for the FD $A \rightarrow B$. Note that in Table 2.6, a non-conforming possible world, the ND $A \rightarrow^2 B$ is satisfied.

A	B
o_1	3
2	4
5	o_2

Table 2.4: OR-object indefinite relation

A	B
{1,2}	3
2	4
5	{3,6}

Table 2.5: Indefinite relation

A	B
2	3
2	4
5	3

Table 2.6: Non-conforming possible world

A	B
1	3
2	4
5	3

Table 2.7: Conforming possible world

We now motivate NDs and indefinite information by providing an example where the traditional FD is too strict and a *weaker* integrity constraint is required. For this we claim that the ND is a worthwhile generalisation. Table 2.8 shows how we might want to represent indefinite information in a teaching relation $PLAN$(Lecturer, Course). Irrespective of whatever courses Mark and Robin decide to teach no definite relation extracted from $PLAN$ will satisfy the FD Lecturer $\rightarrow$ Course though all satisfy the ND Lecturer $\rightarrow^2$ Course. This may be the desired goal of the database designer who wishes to represent the fact that a Lecturer can teach up to two courses in a year. We formalise this notation in Chapter 4 where we use ND set satisfaction within the possible worlds in indefinite relations to approximate FD set satisfaction.

Lecturer	Course
Mark	{B11a, C320}
Robin	B11a
{Robin, Mark}	B151

Table 2.8: An indefinite relation $PLAN$

2.2.8 Temporal Databases and Temporal Dependencies

There are a number of different methods for modelling temporal data within the relational model; not least due to the fact that there may be many different applications and requirements of a temporal database. These may range from financial data storage to recording sales data or simply storing dates for birthdays.

A temporal relation can be considered as one which adds a third dimension, time, to a standard relation. A two-dimensional relation corresponding to a given time is referred to as a *snapshot* relation. There are two prime modes for interpreting time in a relation, *valid* and *transaction* time, which we now define, extracted from (Jensen et al., 1998):

Definition 2.2.36 (Valid Time) Valid time represents the time over which the fact to which the tuple is attached is true within the world or reality which we are modelling. Valid time is usually supplied by the user. □

Definition 2.2.37 (Transaction Time) The transaction time attached to a tuple is the time at which the tuple was entered into the database until it is logically deleted. A transaction time may be associated with any database object, implemented via a system generated transaction commit time. □

In the past transaction time may have been represented implicitly though with the increase of data mining and data warehousing systems this is more likely to now be explicit also. We assume time, within the context of this thesis, to be valid time. We also assume that the reader is familiar with the interval representation of time (Allen, 1984). The other key issue in temporal databases (and also temporal reasoning and planning research) is that of temporal granularity (Bettini et al., 1996). We are not concerned specifically with granularity problems though we refer to problems that granularity issues might pose as and when they may occur. There are also many options for representing the time domain; we assume that time is discrete.

There has been a growing body of work on dependencies in the temporal domain. In a temporal database model dependencies extend to cover dynamic behaviour within the database, dynamic implying changes over time. As such, a temporal dependency may restrict the evolution of the database. Much work on temporal dependencies makes use of temporal logic (Emerson, 1990; Manna and Pnueli, 1992) which we also assume the reader is familiar with. The field of temporal dependency satisfaction is directly relevant to temporal data mining.

(Vianu, 1987; Vianu, 1988) introduced dynamic FDs with a view to integrating dependencies into the relational model which constrain the evolution of a database. We now define an Action Relation so that we can easily present dynamic FDs.

Definition 2.2.38 (Action Relation) Given two relations r_1, r_2 such that r_2 is the relation generated after an update (insertion, deletion, and/or modification) is applied to r_1 we form the action relation r_a from r_1 and r_2. For all attributes $A \in r_i$ we add a subscript i to each attribute so that it corresponds with the temporal state it is in and we then create the *action relation* r_a where $r_a = \{t \times \delta(t) \mid t \in r_1\}$ for all tuples t in r_a where $\delta(t)$ is the updated tuple $t \in r_2$. □

Definition 2.2.39 (Dynamic Functional Dependency (DFD)) A dynamic functional dependency $X \rightarrow Y$ across states r_1, r_2 is an FD over the action relation r_a formed by r_1 and r_2 such that for each $A_i \in Y$ we have $XA_i \cap r_1 \neq \emptyset$ and $XA_i \cap r_2 \neq \emptyset$. □

Note that the states formed by the action relation need not be contiguous. An example presented in (Vianu, 1987) is the DFD $MERIT_1SALARY_1 \rightarrow SALARY_2$, implying that the old merit and salary of an employee determine the new salary. We also note that dynamic dependencies could be easily extended to dynamic numerical dependencies. Examples for dynamic numerical dependencies might include $MANAGER_1GRADE_1 \rightarrow^k EMP_2$, stating that the grade of a manager determines subsequently the number of employees he is allowed to manage, perhaps as part of a career development rule. We assume the interim time period between the two states is of a fixed length.

Another extension for dynamic NDs may be a change of the branching factor in an ND being determined by old dependencies. Using $\Rightarrow$ to denote implication, we may form a rule $(MANAGER_1GRADE_1 \rightarrow^k EMP_1) \Rightarrow (MANAGER_2GRADE_2 \rightarrow^m EMP_2)$, stating that the grade of a manager taken together with the number of employees he currently manages (k) determines subsequently the number, m, of employees he is allowed to manage taken together with his grade. This type of ND is different from a FD in that the dependency itself is modified based upon the structure of the data in a previous state. Though potentially useful we do not consider these dependencies further.

(Jensen et al., 1996) introduce the TFD, $X \xrightarrow{T} Y$, which holds in a temporal relation schema if for all snapshots the FD $X \rightarrow Y$ holds. $X \xrightarrow{T} Y$ refers only to temporal data models though (Jensen et al., 1996) note that FDs are *intensional* in that they apply to every possible extension which the TFD represents. (Wijsen, 1995) defines two temporal dependencies using operators based on standard temporal logic, noting that $maxtime$ represents the final point in time, assuming time is represented by a finite set:

1. A *Temporal Functional Dependency* $\Box (X \rightarrow Y)$ which is satisfied at time i if $X \rightarrow Y$ holds in all states from i to $maxtime$.

2. A *Dynamic Functional Dependency* $\bigcirc (X \rightarrow Y)$ which is satisfied at time i if $X \rightarrow Y$ holds in $i, i+1$ or $X \rightarrow Y$ holds if $i = maxtime$.

We remark that the temporal dependencies of (Vianu, 1987; Wijsen, 1995) may be mined, at each state, similarly to standard FDs.

(Gertz and Lipeck, 1995) introduces *transition graphs* for analysing state sequences. These graphs are labelled such that transitions within a sequence have constraints attached to them; in such a way an *admissible lifecycle* of an object, as well as other concepts, can be represented in a temporal database. Transition graphs may be used to recover from information gaps, which may

occur from either an update or a delete, providing the *time instants* fill the gap exactly and that the loop label is valid when the gaps meet.

(Chomicki, 1994) introduces the use of linear temporal logic (LTL) (Emerson, 1990) to represent integrity constraints in a database. Temporal integrity constraints can be easily stated in LTL though there is no notion of any *transition constraints*. (Chomicki and Toman, 1998) discusses the application of temporal logic in databases noting that it allows querying without explicit reference to time. Temporal logic is easily applied for the specification of temporal integrity constraints in a relational database. Temporal logic may also be used to define constraints on the evolution of a database. An example follows.

Example 2.2.3 We represent the intuition in a company database containing relations *employee*(name) and *trainee*(name), extendible with time attributes, that all employees must have, *at some time in the past* ($\blacklozenge$), been trainees:

$$\neg\exists x(employee(x) \wedge (\neg\blacklozenge trainee(x)))$$

Given that a temporal constraint may relate to an event that has not yet occurred, for example, there may be trainees who are not yet employees, then updates are only allowed within such a database if all of the constraints may be *potentially satisfied* (Chomicki and Toman, 1998). There has been a large amount of work on restricted classes of temporal logic for dependency satisfaction including the restriction of temporal operators to past connectives which apply only to finite histories. There are numerous methodologies for temporal dependency representations (Jensen et al., 1996). We do not consider these directly in our work though our sequence logic allows us to view specific formulas as NDs holding over certain time periods, detailed in Chapter 5. The temporal logic we introduce is restricted to expressing patterns within temporal sequences, defined with regard for knowledge discovery purposes.

2.2.9 Time Series and Temporal Databases

As we shall see, much of our temporal work has a relationship with time series analysis. We now introduce time series and formalise the relationship with temporal databases. The dichotomy between temporal database research and time series analysis, partially addressed in (Schmidt et al., 1995), is now disappearing given the incorporation of data mining and statistical functions into DBMSs and related increases in querying and computation speed.

(Segev and Shoshani, 1993) defines the properties of time sequences for the creation of a temporal data model. These properties include their type, granularity and *lifespan*, which specifies the start and end time of a time sequence. A temporal data value is defined as a triplet $\langle s, t, a \rangle$

where s is an identifier for an object, t is the time, and a the attribute value. In the model these values are totally ordered in time. For example, a time sequence may be the midday price of a given share over a period of two years. The triplet may be reduced to a sequence of $\langle t, a \rangle$ for a known object, referred to as a time sequence collection. The object s may also denote complex instances corresponding to values obtained for composite clauses. A class of retrieval operators are developed for this model, of which the closest to time series functions are the aggregation operators for max, sum etc. Clearly these operators are too naive for time series analysis.

Given a relation over R = ABT, where T is time, and there exists an attribute over a numerical domain, say A, and each tuple occurs with a constant time between each point then $\pi_{\mathrm{A}}(\mathrm{R})$ will represent a time series. If each tuple does not occur at constant times the relation may be *folded* or *unfolded* to obtain records over fixed intervals.

Definition 2.2.40 (Time Series) A sequence $\{x_n : 0 \leq n \leq N\}$ of N observations, indexed by the time at which they were taken. These may be modelled by random processes. □

Time series analysis usually requires accounting of the order of observations; which are in general not independent implying that forecasting is possible. A deterministic time series is one which can have its future predicted exactly, though it is obviously of minimal worth. In practice, time series occur frequently in the economic domain, for example, in successive share prices. There also exist numerous meteorological and geological time series, for example.

Essentially, any time series analysis attempts to predict $y(N+1), y(N+2)$, and so on, using the values in the sequence $y(1), y(2), \ldots, y(N)$. The quality of predictions in a time series context is given by:

$$\frac{\Sigma_t(observation_t - prediction_t)^2}{\Sigma_t(observation_t - observation_{t-1})^2}$$

If the above is less than one then this implies an improvement over the random walk. A clear overview of many of the uses of time series is presented in (Weigend and Gershenfeld, 1994) which also denotes the three main aims of time series analysis as being: (1) *forecasting* which attempts to predict short term system evolution, (2) *modelling* which attempts to describe long term system behaviour, and (3) *characterisation* which attempts to determine the fundamental properties of a system. We might say that we use our logic for property discovery to *characterise* one or more temporal sequences, and possibly to aid forecasting.

2.3 Dependency and Temporal Data Mining

We now move on to present the background on data mining related to this thesis. Firstly, we introduce functional dependency data mining, before considering its relationship with ND approxi-

mations in Chapter 3. We discuss similarity measures for FDs, sampling procedures for databases and temporal data mining, in particular temporal rule discovery research.

2.3.1 Functional Dependency Data Mining

Dependency inference is a key area in the rapidly developing field of data mining. This section focuses on the inference of functional dependencies. Dependency mining is also concerned with inference of inclusion, join, multivalued and algebraic dependencies, not examined in this thesis. In recent years work has progressed from the inference of FDs in relations to methods to infer approximations to FDs, based on the real-world requirements where many large databases contain noise making exact FD inference unfeasible. We first highlight a potential problem with FD inference when there are FDs with multiple attributes on their left hand side.

Lemma 2.3.1 A relation schema R with $|\ R\ | = n$ has $n2^{n-1}$ possible non-trivial FDs.

Proof. For each attribute $A \in R$ there are 2^{n-1} attribute sets in $R \setminus \{A\}$. □

As lemma 2.3.1 shows, there is an exponential number of possible FDs in the number of attributes which may hold in a relation. We note however that many real-world databases have numerous attributes, notably many of the datasets in (Blake et al., 1998). However, in a relation containing, say, 11 attributes it is highly likely that AB ... GH → I is satisfied functionally or close to functionally, due to the attribute set AB ... GH having a significant number of value combinations even within a binary database. We therefore suggest restricting the left hand side of attributes to a reasonable number for dependency discovery. This is a motivation for a use of a dependency *template*, provided by the user, which contains those FDs that he wishes to see approximated. We use this in our work in both indefinite and temporal relations.

Definition 2.3.1 (Dependency Inference problem) Given a relation r, the dependency inference problem is is to find a small (if not the smallest possible) *cover* for the set of all dependencies in r. □

The functional dependency inference problem, initially described in (Mannila and Räihä, 1986), is to find a set of functional dependencies equivalent to the set of all functional dependencies that hold for a given relation r.

To test for satisfaction of a FD in a relation requires $O(n^2)$ comparisons, where n is the number of tuples within the relation. For a set of FDs this is computationally expensive and so techniques for approximating the set of functional dependencies are discussed, presented in (Mannila and Räihä, 1994; Kivinen and Mannila, 1995; Piatetsky-Shapiro and Matheus, 1993; Savnik and Flach, 1993; Schlimmer, 1993; Shen, 1991). (Schlimmer, 1993), (Savnik and Flach, 1993)

and (Piatetsky-Shapiro and Matheus, 1993) use probabilistic measures. (Savnik and Flach, 1993) infer dependencies from a database using dependencies which are known to be invalid in the database as well as the valid dependencies. (Bell and Brockhausen, 1995) discusses the problem of dependency inference within real world databases where issues concerning dependency inference after updates are considered. Recently (Huhtala et al., 1998) has looked at improving the efficiency of approximate FD data mining. They maintain information about which rows agree on a set of attributes, partitioning the relation on these different values. This then allows FD inference to reduce to checking that the rows agree on the left hand side whenever they agree on the right hand side. Each set within a partition is known as an equivalence class.

Example 2.3.1 In relation $PLAN$, given in Table 2.3, we note that attribute *Lecturer* agrees on tuples t_1 and t_4 as well as t_2 and t_3. Attribute *Course* agrees only on tuples t_3 and t_4. The partition with respect to *Lecturer* is $\pi_{\{L\}} = \{\{1,4\},\{2,3\},\{5\}\}$. The partition with respect to *Course* is $\pi_{\{C\}} = \{\{1\},\{2\},\{3,4\},\{5\}\}$.

A partition π_0 refines partition π_1 if every equivalence class in π_0 is a subset of some equivalence class in π_1. (Huhtala et al., 1998) show that an FD $X \rightarrow Y$ holds if and only π_X refines π_Y. This work may be extended with regard to the mining of NDs by examination of equivalence classes, noting that an ND $X \rightarrow^k Y$ will hold by counting the number of subsets each member of the equivalence classes of π_X has in π_Y where k is the maximum number across all equivalence classes. Optimisations, used by (Huhtala et al., 1998), principally the removal of equivalence classes of size 1, noting that they can not violate an FD, and pruning of the search space, enhance the efficiency of FD mining to increase linearly with increases in the number of rows.

Uses of dependency inference include database design, query optimisation, determinations and various constraint satisfaction procedures. Machine learning can be said to be the inference of general rules from instances of data and, as such, dependency inference is an attractive branch given that for the instances of data, the database itself, there always exists a concept which fits the data set, namely the dependency set (Mannila and Räihä, 1994). (Mannila and Räihä, 1986) show the dependency inference problem to be the converse of the generation of an Armstrong relation for a given set, F, of FDs. (Bell and Brockhausen, 1995) notes three possible approaches to the dependency inference problem: (1) Enumerate and verify all possible data dependencies, (2) infer as much as possible and prevent unnecessary queries, and (3) draw inferences from the verified and invalid data dependencies.

In (Kivinen and Mannila, 1995) the problem described is to find a *cover* F_c of the set of FDs which hold in r, where F_c is a minimal set. Algorithms to do this are in the worst case

exponential in the size of the smallest cover of the dependency set, as presented in (Mannila and Räihä, 1994). Therefore (Kivinen and Mannila, 1995) suggest an approximation algorithm. The prime results of their work are measures on the error of a dependency f holding in a relation r and an algorithm for finding, with high probability, a set of FDs, F, such that $d(\mathrm{F}, dep(r)) < \epsilon$, where d is a distance measure, ϵ is the allowed error and $dep(r)$ denotes the FD set holding in r. The algorithm which Kivinen and Mannila have implemented works in polynomial time with respect to $\frac{1}{\epsilon}$ and the size of the smallest cover of F. Of particular interest are the dependency error measures which are used, to which we refer to in Section 2.3.3.

(Mannila and Räihä, 1992b) present an algorithm for dependency inference of a cover in a relation r, using *hypergraph transversals*, or *hitting sets* (Eiter and Gottlob, 1995). We briefly introduce this procedure. A hypergraph is a family of subsets of R. A set $\mathcal{H}$ of subsets of R is a simple hypergraph if no element of $\mathcal{H}$ is empty and if $X, Y \in \mathcal{H}$ and $X \subseteq Y$ imply that $X = Y$. The elements of $\mathcal{H}$ are referred to as the edges of the hypergraph and the elements of R are the vertices. A transversal T of $\mathcal{H}$ is a subset of R intersecting all of the edges of $\mathcal{H}$ such that $T \cap E \neq \emptyset$ for all $E \in \mathcal{H}$. A minimal transversal of $\mathcal{H}$ is a transversal T such that no $T_l \subset T$ is a transversal. We denote the minimal transversals by $Tr(\mathcal{H})$.

(Mannila and Räihä, 1992a) prove that the complement of the set of maximal sets, $cmax(A) = \{R \backslash W \mid W \in max(A)\}$, is a hypergraph. (Mannila and Räihä, 1992a) presents an algorithm for dependency inference, of polynomial time in the size of $\mid r \mid$, $\mid R \mid$, and the product of the sizes of the $cmax$ sets. The algorithm computes $cmax$ for the max sets which hold in a relation and then forms a cover of these dependencies using transversals. Lemma 13.3 of (Mannila and Räihä, 1992a) shows that $Tr(cmax(\mathrm{A})) = lhs(\mathrm{A})$ where $lhs(\mathrm{A})$ is exactly the set of elements $\mathrm{X} \subseteq \mathrm{R}$ such that, for an FD set F, $\mathrm{F} \models \mathrm{X} \rightarrow \mathrm{A}$, there does not exist $\mathrm{Y} \subset \mathrm{X}$ where $\mathrm{F} \models \mathrm{Y} \rightarrow \mathrm{A}$. Hypergraph transversals may therefore be used for dependency discovery. We illustrate this procedure with a small example.

Lecturer	Course	Room
Robin	C320	G11
Mark	B11a	227
Robin	B11a	G11

Table 2.9: Relation $PLAN_2$(Lecturer, Course, Room)

Example 2.3.2 In relation $PLAN_2$, given in Table 2.9, we abbreviate the respective attributes Lecturer, Course, and Room to L, C, and R. Firstly we form the disagreement sets and remove any

subsets to obtain the *cmax* values for each attribute. This gives $disag_{\mathrm{L}}$ = { LCR, LR }, $disag_{\mathrm{C}}$ = { LCR, C }, and $disag_{\mathrm{R}}$ = { LCR, LR }. We have, after superset removal, $cmax_{\mathrm{L}}$ = { LR }, $cmax_{\mathrm{C}}$ = { C }, and $cmax_{\mathrm{R}}$ = { LR }. We now form the hypergraph transversals such that lhs(L) = { L, R }, lhs(C) = { C }, and lhs(R) = { L, R }. Therefore we may infer the dependency set F = { L $\rightarrow$ R, R $\rightarrow$ L } from $PLAN_2$, assuming removal of trivial FDs, implying that a Lecturer teaches only in one Room and that a Room only has one Lecturer teach in it.

(Mannila and Räihä, 1992b) state that it is one of the aims of dependency inference to obtain algorithms which work in polynomial time in the number of different minimal left-hand sides of each attribute. (Bitton et al., 1989; Mannila and Räihä, 1992a) also present algorithms using the behaviour of disagreement sets to optimise the discovery process, as does (Savnik and Flach, 1993). (Savnik and Flach, 1993) provides a bottom-up inductive approach with a view to automating data dependency creation via discovery of the dependencies from the existing relations within the database. Savnik and Flach define the process of inducing FDs using *invalid dependencies*, which will all be contradicted by a given relation. We now formalise this:

Definition 2.3.2 (Invalid Dependency) A FD is *invalid* in a relation r if it is contradicted by two or more tuples within r. □

Definition 2.3.3 (Positive Cover) A set of dependencies F is a positive cover for relation r *if and only-if*

1. All FDs are of the form X $\rightarrow$ A where A is a single attribute.
2. For all functional dependencies that are satisfied in r there is a more general dependency in the positive cover so that if X $\subseteq$ Y then X $\rightarrow$ A is more general than Y $\rightarrow$ A. □

(Savnik and Flach, 1993) presents the notion of *negative cover* so that every pair of tuples need not be examined for contradiction of a dependency which allows inference of all dependencies contradicted by the relation. In contrast to the above, in a negative cover all invalid dependencies in r there is a more specific dependency in the positive cover so that if X $\supseteq$ Y then X $\rightarrow$ A is more specific than Y $\rightarrow$ A. Invalid dependencies are identified by comparing each pair of tuples within a relation and splitting their attributes into two partitions, one for equivalent attribute values the other for non-equal attribute values. The checking of dependency satisfaction then becomes a simple search for more specific dependencies in the negative cover. A problem with this approach is the removal of meaningless and useless data, the former relating to trivial dependencies and the latter relating to information that can be deduced using Armstrong's axioms. (Bell and Brockhausen, 1995) present procedures for FD discovery in standard SQL. The

results are shown to be poor with respect to efficiency but the methods can be applied to large databases and SQL is eminently portable. Fuzzy FDs (Bosc et al., 1994; Hale et al., 1994) may also be viewed as approximations to FDs. A fuzzy weight can represent either the degree to which the tuple belongs in the relation or the global confidence level in the information that is stored in the tuple. To prevent multiple weights over R for the same attribute it is assumed that weight is a special attribute and the FD R $\rightarrow$ $Weight$ holds. Fuzzy subsets can be viewed as a collection of weighted subsets. Rough sets, introduced in (Wong and Ziarko, 1986), may also be used for FD approximation. Each attribute A has a class description which is the set of regions into which each value of X will fit. From this *upper* and *lower* approximations may be formed based on exact and minimal set membership of the attributes. These classifications may then be used to test for FD satisfaction (Beaubouef et al., 1995).

We present some results for mining of NDs for approximating FDs in Chapter 3.

2.3.2 Temporal Dependency Data Mining: A review

A significant amount of work has been carried out on data mining within temporal databases. *Temporal Data Mining* generally takes the form of finding interesting patterns (Berger and Tuzhilin, 1998) or rules (Das et al., 1998; Mannila and Toivonen, 1996a). Our approach uses a number of features, similar to and developed independently from previous work. We introduce these components followed by a brief outline of knowledge discovery research conducted on time series. It is important that the reader appreciates the highly disparate goals between our work (and the associated work presented here) and that of time series methodologies using neural network or connectionist architectures (Weigend and Gershenfeld, 1994; Faraway and Chatfield, 1995). Naively, we demarcate this from knowledge discovery research in that its goal is to create neural networks, using many different mechanisms, which successfully forecast the values of a time series. It is not concerned with *understanding* the time series but simply forecasting future values. Alternatively, knowledge discovery research is user-oriented, attempting to provide understandable rules that a *data miner* can easily follow without a significant knowledge of statistics or time series analysis techniques. This may be said to be a key goal of our own research, presented in Chapters 5 and 6.

Nearly all temporal data mining research breaks an input temporal sequence into subsequences. The ability to find a global model describing a sequence is very difficult for any non-trivial time series (Enders, 1995).

(Mannila et al., 1995; Mannila and Toivonen, 1996a; Mannila and Toivonen, 1996b) define episodes for modelling event sequences. An *event* is a tuple with a timestamp attached. Also, x occurring within $[t_1, t_2]$ where $t_1 \leq t_2$ implies that x holds at all points p where $t_1 \leq p \leq t_2$.

Attached to each *episode rule* is a frequency of each episode occurring within a sequence. Therefore (Mannila and Toivonen, 1996a) states the episode rule discovery task is to find all frequent episode rules, where the frequency may be specified by the user. We shall show how properties discovered by our logic are related to issues of frequency in Chapter 6. (Mannila and Toivonen, 1996a) restrict the rule discovery task to serial or parallel episode discovery where parallel implies that there are no conditions on the relative order of events. Mannila and Toivonen prove that finding whether a serial or parallel episode holds within a sequence is an NP-complete problem therefore the discovery is necessarily restricted. This definition of episode is different from the accepted definition of episode in temporal logic.

Definition 2.3.4 (Episode of (Mannila and Toivonen, 1996a)) An episode is a conjunction $\wedge_{i=1}^{k}\phi_i(y_i, z_i)$ where y_i, z_i are event variables and $\phi_i(y_i, z_i)$ is of the form $\alpha(x.\mathrm{A}), \beta(x.\mathrm{A}, y.\mathrm{B})$ or $x.\mathrm{T} \leq y.\mathrm{T}$ denoting a unary predicate on the domain of A, a binary predicate on the domains of A and B, or a temporal ordering relationship, respectively. □

Definition 2.3.5 (Episode Rule) An episode rule takes the form $P[V] \Rightarrow Q[W]$ where P, Q are episodes and V, W are real numbers denoting that if P occurs throughout the interval $[t_1, t_2]$ with $V \geq t_2 - t_1$ then Q occurs in $[t_1, t_3]$ with $W > t_3 - t_1$. □

Algorithms presented are based upon discovery of simple episodes, namely those without binary predicates, using minimal occurrences. An occurrence of a simple serial episode is minimal over an interval $[t_i, t_j]$ if it does not hold over any subinterval of $[t_i, t_j]$. The discovery process exploits this by finding minimal occurrences and increasing the episode size for serial episode discovery. We now illustrate this with an example.

Example 2.3.3 An example of a rule found is (dept. page, spring term 96 $[15s] \Rightarrow$ classes spring 96 $[30s]$) (confidence 0.83). This rule tells us that 83% of cases where the department page and the spring term 96 page were accessed within 15 seconds resulted in the classes spring 96 page being visited within 30 seconds.

The results are shown to be useful for expressing connections between events. Our temporal logic was similarly defined to express connections between events.

(Padmanabhan and Tuzhilin, 1996) claims to extend this work to the discovery of temporal logic patterns. This work simply uses temporal logic to represent episodes expressed in clausal form, for example $holds(stock) \rightarrow value\_increase(stock, 25)$. We agree that temporal logic is an expressive and valuable mechanism for rule discovery though the implementation using datalog, given in (Padmanabhan and Tuzhilin, 1996), provides no results due to inefficiency; if any-

thing, this shows how we need to be careful of efficiency considerations when constructing data mining algorithms, particularly in the temporal domain, where there may be many possible rules.

(Berger and Tuzhilin, 1998) uses a subset of propositional temporal logic, incorporating operators $A\mathcal{B}_k B$, $A\mathcal{U}B$, $A\mathcal{N}B$, to denote k events before, until, and next respectively. They attempt to discover patterns attached to a defined measure of *interestingness*, defined as the actual number of occurrences of a pattern exceeding the expected number of occurrences. This is equivalent to specifying a required frequency though it is complicated by attaching probabilities to each event. However, with probabilities attached the discovery of larger from smaller patterns becomes non-monotonic. Therefore, given a temporal logic pattern containing only Before ($\mathcal{B}$) operators, the discovery of interesting patterns is shown to be NP-complete (by reduction to an instance of CLIQUE (Garey and Johnson, 1979)). To deal with this a restriction is placed on the temporal logic and the maximum length of patterns discovered. We present a simple example from (Berger and Tuzhilin, 1998).

Example 2.3.4 Given the 20 item string $ABABABABCCCCCCCCCCCC$ where A, B, and C are events, the expectation of both A and B is 5 implying that both have a 0.25 probability of occurring. The expectation E of $A\mathcal{N}B$ is then $Pr(A)Pr(B)$(N - 1) = (0.25)(0.25)(19) where N is the length of the string. Given that $A\mathcal{N}B$ occurred 4 times the *interestingness* is 4/((0.25)(0.25)(19)).

Naive algorithms presented are based on expanding an interesting pattern with prefix and suffix operators and then examining the interestingness of the generated rules. The non-monotonicity of the approach prevents interesting patterns being expanded by anything more than a single literal preventing conjunctions (not included in their syntax) of temporal operators being discovered. Results show that the length restriction prevents significant knowledge from being discovered once the data set grows too large. In the case of simulations conducted on web log data this was 1400 points. We compare this work with our own in Section 6.8.

(Srikant and Agrawal, 1996) essentially applies the discovery of association rules in a temporal setting. We now present data mining research on time series rule discovery. (Faloutsos et al., 1994) present the goal of mining a time series as that of searching for a subsequence in the series which matches a given query. The discrete Fourier transform is used for mapping the time series into the *frequency domain* and then forming a trail in multi-dimensional feature space based on the first f coefficients so that the time series can be clustered into rectangles in feature space. This allows similarity queries to then be answered. Results show these procedures to be more efficient than standard sequential scanning processes. (Agrawal et al., 1995; Das et al., 1997;

Rafiei and Mendelzon, 1997) compare the similarity of time series by examing non-overlapping time ordered pairs of subsequences. Again the subsequences are similar if the number of matches exceeds a given threshold. Offsets, gaps, and scaling are all addressed by this model. (Das et al., 1997) presents a number of transformation functions specifically to handle outliers and scaling. The goal here is to *approximately* map one sequence into another. (Keogh and Smyth, 1997) has the same goal and uses templates which are deformed by a probability distribution. We refer to the use of our logic for similarity assessment in Section 6.9.

(Das et al., 1998) is closely associated with our research primarily in that it attempts to discover rules from time series. Given that we may view ND sequences as time series then our logic can be said to have the same goal. (Das et al., 1998) initially discretise the series and then attempt to cluster them according to similarity of the pattern. The discretisation creates a sequence of primitive shapes related to the chosen window size. The measures for clustering may range, in the simplest instance, from Euclidean distance to more sophisticated measures, not discussed here. A frequency is then attached to produce rules which are similar to association rules.

Definition 2.3.6 (Temporal Rule of (Das et al., 1998)) A temporal rule is of the form $A \overset{\Rightarrow}{_{\mathrm{T}}} B$ which denotes that *if* A *occurs, then* B *occurs within time* T. A frequency of the number of occurrences is associated with the rule as is a confidence in the rule obtained from the frequency divided by the number of occurrences of A, the left hand side, in the sequence. □

(Das et al., 1998) also discusses extensions to multivariate series by having conjunctions of different patterns on the left hand side of the rule. This extends the applicability of their method and is discussed more fully in Chapter 5.

2.3.3 Similarity Measures for Functional Dependency sets

Data mining tools often require a *quality function* which assesses and classifies the knowledge discovered in a form which is understandable by the user (Holsheimer and Siebes, 1994). In this section we briefly present a synopsis of measures used to approximate the distance from ND satisfaction in a relation, of which NDs are a category, and then we present methods to compare distance between FD sets themselves.

In Table 2.10 we present some approximation measures used for FDs. The *error measures* of (Kivinen and Mannila, 1995) are all based, in some sense, on the proportion of a relation which violates an FD $X \rightarrow Y$. The measure of (Piatetsky-Shapiro and Matheus, 1993) requires a frequency table, shown in Table 2.11, which sums the different values for each partition on the FD, which we detail for the relation PLAN in Table 2.3. The values given for FD approximation vary significantly depending on the choice of measure. The different results depend on how we choose

FD Approximation Methods	Values from Table 2.3
(Kivinen and Mannila, 1995)	$\mid r \mid = 5$ in all cases
Error measure for the number of violating tuple pairs in a relation r for an FD X $\rightarrow$ Y: $g_1(\mathrm{X} \rightarrow \mathrm{Y}) = \frac{\mid \{(t_i, t_j) \mid t_i, t_j \in r, t_i[\mathrm{X}] = t_j[\mathrm{X}], t_i[\mathrm{Y}] \neq t_j[\mathrm{Y}]\} \mid}{\mid r \mid^2}$	$\frac{4}{25}$
Error measure for the number of violating tuples in a relation r for an FD X $\rightarrow$ Y: $g_2(\mathrm{X} \rightarrow \mathrm{Y}) = \frac{\mid \{t \mid t \in r, \exists t_i \in r \text{ such that } t[\mathrm{X}] = t_i[\mathrm{X}], t[\mathrm{Y}] \neq t_i[\mathrm{Y}]\} \mid}{\mid r \mid}$	$\frac{4}{5}$
Approximation measure for the size of the largest partition s in a relation r which satisfies an FD X $\rightarrow$ Y: $g_3(\mathrm{X} \rightarrow \mathrm{Y}) = 1 - \frac{max\{\mid s \mid\mid s \subseteq r \text{ and } s \models \mathrm{X} \rightarrow \mathrm{Y}\}}{\mid r \mid}$	$\frac{2}{5}$
(Piatetsky-Shapiro and Matheus, 1993)	
Conditional probability that any two rows in r agree on Y, given they agree on X, $pdep$(X,Y) = $p(t_1[\mathrm{Y}] = t_2[\mathrm{Y}] \mid t_1[\mathrm{X}] = t_2[\mathrm{X}])$ where $pdep(\mathrm{X}, \mathrm{Y}) = \frac{1}{\mid r \mid} \sum_{i=1}^{K} \sum_{j=1}^{M} \frac{n_{ij}^2}{c_i}$ where K, M are the number of different values in attributes X and Y, c_i is the number of X values where X = i and n_{ij} is the number of tuples with X = i and Y = j	$\frac{3}{5}$
Numerical Dependency	
X $\rightarrow^k$ Y	L $\rightarrow^2$ C
Mean ND value, see definitions 3.1.7, 3.1.8	
X $\rightarrow^k$ Y	L $\rightarrow^{1.66}$ C

Table 2.10: A comparison of FD Approximation Techniques

to approximate FD set satisfaction. Table 2.10 shows that this may be achieved via counting violating tuple pairs, violating tuples, counting the rows of the largest partition which satisfies an FD or by assessing the conditional probability that two rows agree on Y given that they agree on X for X $\rightarrow$ Y. An ND for the FD X $\rightarrow$ Y finds the maximum number of different Y values (k) for partitions which agree on X. The Mean ND value is formally defined in Section 3.1.3.

	Course				
Lecturer	**C320**	**B11a**	**B151**	**C340**	c_i
Mark	1	0	1	0	2
Robin	0	1	1	0	2
Sean	0	0	0	1	1
l_j	1	1	2	1	5

Table 2.11: Frequency Table for relation PLAN

We note how these measures consider FDs to be the goal of the dependency search. Our use of NDs, though they approximate NDs, considers them in their own right as possible dependen-

cies which may hold when an FD is too strict. Therefore NDs are expressing a general constraint which may hold in a relation, such as a teacher can teach at most two courses, and in this case we do not consider any part of the relation to be *erroneous*. This demonstrates the general applicability of NDs. We define a metric for NDs in Chapter 3 and use this within our simulations in Chapter 4.

In our work on the use of example relations, outlined in 3.5, within the database design process we assessed the evolved relations via the use of a quality function for FDs (Collopy and Levene, 1996). This quality function can be used to describe the proximity of relation s to an Armstrong relation for a set F of FDs, being one when the evolved example relation is an Armstrong relation; it may also be used to generate the distance between two FD sets. This was taken to be the symmetric difference of GEN(F) and GEN(dep(s)), where GEN(F) is the set of generators for a set of FDs F (Mannila and Räihä, 1986), and dep(s) is the dependency set holding in s. It is stated as:

$$quality(\mathrm{F}, s) = \frac{|\ \mathrm{GEN(F)} \cap \mathrm{GEN(dep}(s))\ |}{|\ \mathrm{GEN(F)} \cup \mathrm{GEN(dep}(s))\ |} \tag{2.1}$$

For example, two customer relations for different supermarkets may need to be assessed against a hypothetical optimally performing supermarket and/or against themselves. Obviously, a reliable distance measure is needed. Other areas of application occur in a database design context. We aim to assess this measure. We seek to characterise the distance from Armstrong that a relation can be. In this way, for a relation r, we can infer exactly how *distant* r is from the best relation that can hold for any relation of the same size satisfying the same FD set.

In the remainder of this section we briefly define distance measures and a metric before presenting a brief analysis of an FD measure using the symmetric difference of the closure of FD sets. We characterise this in Figure 2.2 to enforce the point that information about the behaviour of measures we are using themselves aid the data mining process. (Toivonen et al., 1995) discuss a distance measure for association rules. Association rule discovery in (Agrawal et al., 1993) assumes a binary database. For relation r with schema R, and given a set of attributes $X \subseteq \mathrm{R}$ and a tuple $t \in r$ if $\forall A \in X\ t[A] = 1$ then $t[X] = \bar{1}$. The set of tuples matched by X is $m(X) = \{t \in r \mid t[X] = \bar{1}\}$. The distance between two association rules $X \Rightarrow Z$ and $Y \Rightarrow Z$, where $\Rightarrow$ denotes implication, is defined as:

$$\begin{aligned} d(X \Rightarrow Z, Y \Rightarrow Z) &= |\ (m(XZ) \cup m(YZ)) - m(XYZ)\ | \\ &= |\ m(XZ)\ | + |\ m(YZ)\ | - 2\,|\ m(XYZ)\ | \end{aligned}$$

(Tuomela, 1978) provides a general overview of distance in logical terms without any regard for

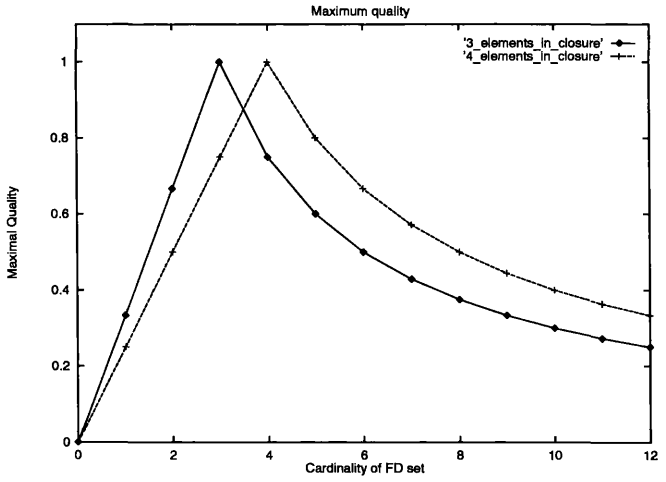

Figure 2.2: Max Quality for FD sets with 3 and 4 elements in closure

practical methods of distance evaluation. He notes, "We can finally say that the more overlap or common content (information) [theories] T_1 and T_2 have, the closer they are." Within database design or mining, a designer may attach weights to the set of FDs implying his level of desire for a particular FD to be satisfied in preference to another or others. A normalised (the sum of all weights) distance can then be calculated based on these input factors.

A function $d(F_1,F_2)$ is a metric iff it satisfies the following properties:

$$\begin{aligned} d(F_1, F_2) &= d(F_2, F_1) \\ d(F_1, F_2) &= 0 \quad \text{if and only if} \quad F_1 = F_2 \\ d(F_1, F_3) &\leq d(F_1, F_2) + d(F_2, F_3) \end{aligned}$$

A distance function which violates the last property, known as the triangle inequality, is known as a pseudo-metric, and of use within distance theory.

We now define a similarity measure between two FD sets F and G as a generalisation of the quality function previously defined, using the closure and not generator sets. Such a similarity measure is of use in data mining whenever we desire to compare two FD sets. (Kivinen and Mannila, 1995) define a metric d_p(F, G) = P(CL(F) Δ CL(G)), where Δ is the symmetric difference and P is a probability measure. The measure now defined is a ratio of the symmetric difference used directly. We study its properties and show how this might help the data miner.

Definition 2.3.7 (Similarity Measure) Given two sets of FDs, F and G over R, we define the measure of their similarity as:

$$sim(\mathrm{F}, \mathrm{G}) = \frac{|\ \mathrm{CL(F)} \cap \mathrm{CL(G)}\ |}{|\ \mathrm{CL(F)} \cup \mathrm{CL(G)}\ |} \qquad \square$$

We now seek to characterise the monotonicity properties of sim with respect to F and G. In equation 2.2 we consider the maximum possible values of sim where one FD set F is fixed, containing n elements in CL(F). Any other FD set G containing k elements in CL(G) has a maximal quality, if all FDs in G are in F whenever $k \leq n$, and if all FDs in F are in G whenever $k > n$. In Figure 2.2 we show these variations for two fixed FD sets with 3 and 4 respective elements in their closure.

$$sim_F(\mathrm{G}) = \begin{cases} \frac{n}{k} & : \quad n \leq k \qquad \text{when} \quad \mathrm{G}^\star \supseteq \mathrm{F}^\star \\ \frac{k}{n} & : \quad n > k \qquad \text{when} \quad \mathrm{F}^\star \supset \mathrm{G}^\star \end{cases} \tag{2.2}$$

The similarity measure is monotonically increasing and if a *core*, or intersection, of the two FD sets is increased by two different amounts, m and k, where $m \leq k$, then the value of similarity is larger for the larger core size increase. We now define some axioms of this similarity measure:

1. $sim(\mathrm{F}, \mathrm{F}) = 1$
2. $sim(\mathrm{F}, \mathrm{G}) = sim(\mathrm{G}, \mathrm{F})$
3. $sim(\mathrm{F}, \mathrm{G}) = 0$, if $\mathrm{F}^\star \cap G^\star = \emptyset$
4. $sim(\mathrm{F}, \emptyset) = 0$
5. $sim(\mathrm{F}, \mathrm{G}) = \frac{|\mathrm{CL(F)}|}{|\mathrm{CL(G)}|}$, if $\mathrm{F}^\star \subseteq \mathrm{G}^\star$
6. $sim(\mathrm{F}, \mathrm{G}) \leq sim(\mathrm{G}, \mathrm{H})$ if $\mathrm{F} \subseteq \mathrm{H}$ and $\mathrm{H} \Delta \mathrm{F} \in \mathrm{G}$.

Information concerning related similarities can now be formed. Assume we have three sets of FDs, F, G, and H, and that F is fixed. Now, if $sim_{\mathrm{F}}(\mathrm{G}) = 0$ and $0 < sim_{\mathrm{F}}(\mathrm{H}) < 1$ then we know that $sim(\mathrm{G}, \mathrm{H}) < 1$ given that there is a similarity between F and H. Essentially, this is stating that the core of F and H cannot form any part of the core of G and H. Using knowledge of the measure itself allows for inferences to be drawn easily based on the input FD set and resulting values of the measure. We have briefly presented an overview of a similarity measure for FD sets. In Chapter 3 we define a metric based on the lattice properties of NDs, used within our work on indefinite relations.

2.3.4 Relational Database Sampling Procedures

Many real-world databases are too large to consider applying standard data mining algorithms to. Therefore, as a solution, sampling from such databases has been promoted (Kivinen and Mannila, 1994; Toivonen, 1996). Samples drawn from a large database are mined for dependencies which

are then associated with error and confidence thresholds based on the size of the sample in relation to the database. Alternatively, results obtained from the mining of a sample may be verified against the database as a whole. In this manner sampling is a necessary trade-off between accuracy and efficiency of results.

(Kivinen and Mannila, 1994) addresses the problem of finding a suitable sample size. This is presented within a PAC-learning framework (Valiant, 1984). Based on an error measure, akin to the similarity measure presented in Section 2.3.3 or g_3 of Table 2.10, sampling is used to detect all sentences which have an error (or 1 - similarity) less than a given threshold ϵ. The probability that at least one sentence with an error greater than ϵ will not be formed is given by δ, the confidence parameter. FDs which hold are, obviously, never detected as false. (Toivonen, 1996) presents sampling within an exact discovery framework for association rules using a sample to find a superset of frequent associations subsequently verified by one pass over the database.

2.3.5 Resampling in Statistics

Statistical methods have evolved rapidly over the last 30 years, not least due to the harnessing of increasing computational power. In the 70's statistical modelling was based upon decomposing the data into a structure and noise. In the 80's non-parametric processes such as the *jackknife* were developed where n or more (possibly) correlated estimates of the quantity of interest are replaced by pseudovalues. Linear regression takes a linear combination of the available values whereas non-parametric models keep the data around and use it for estimating the response class of a new point.

The bootstrap (Efron, 1979; Diaconis and Efron, 1983; Efron and Tibshirani, 1993) is a data driven simulation method for estimating the sampling distribution of a statistic. It is a computationally intensive procedure that has been shown to provide good results which would not have been capable of being readily generated more than 30 years ago. In our experience, resampling has not previously been applied to solve database problems such as the consistency problem. Declining computational cost is altering the face of statistical analysis entailing a domino effect in other fields so that computer intensive statistical methods such as the bootstrap will become much more prominent in many areas of computer science over the next few years. Figure 2.3 shows how the bootstrap procedure may be applied to an indefinite relation r. The sample in the figure will consist of n possible worlds, each satisfying an ND set. We now introduce bootstrap resampling with a simple example; resampling indefinite relations is formalised in Chapter 4.

The following example is used for instruction and is similar to one described in (Efron and Tibshirani, 1993) but with a business application. If we have a relation depicting the number of

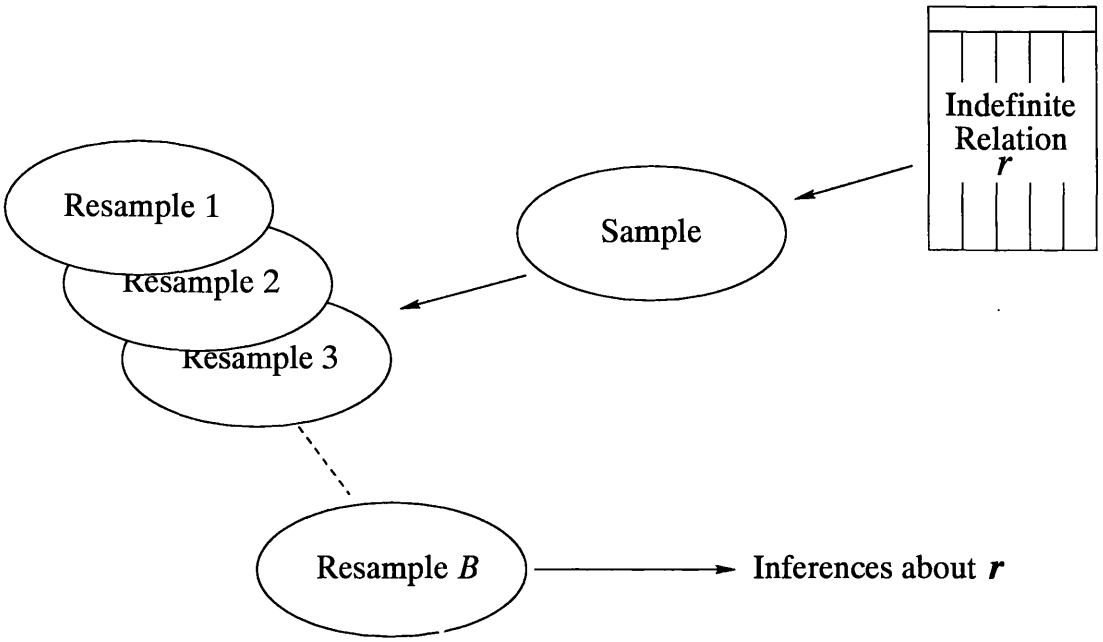

Figure 2.3: The Bootstrap Procedure as applied to an indefinite relation with a Bootstrap Replication size (BRS) B

pension plan subscribers (referred to as Clients) in two different companies with a number of the employees in each company as in Table 2.12 then we can form a ratio of success $\hat{\theta}$ based on the number of clients for the respective number of employees, given as follows:

$$\hat{\theta} = \frac{230/15746}{299/13430} = 0.66$$

Company	Clients	Employees
HAL co.	230	15746
JCN co.	299	13430

Table 2.12: Company Data Relation

So we can say that HAL co. is only 66% as successful as JCN co. when it comes to getting employees to take up its pension plan. Yet this is only an estimated ratio. To apply a bootstrap procedure to the above data we can create two sample populations for each company with 230 clients and (15746 - 230) employees and 299 clients and (13430 - 299) employees, respectively. These populations of 15746 and 13430 items may be represented with ones and zeros to represent clients and employees who are not clients. If we then draw randomly with replacement a sample of 15746 subjects and 13430 subjects from each population we can form what is known as a bootstrap replicate sample success ratio $\hat{\theta}^{\star}$. We can now repeat this, say, 1000 times, to obtain bootstrap standard deviation values or other statistics which are based on the distribution found and not naive assumptions on the distribution.

The majority of bootstrap applications in the statistical domain use resampling due to the unavailability of the complete domain. Likewise, in an indefinite relation, although we potentially have access to all possible worlds, there are generally too many to examine them all. We employ resampling in a dynamic manner on increasing sample sizes, elaborated upon in Chapter 4.

2.4 Discussion

Within the limits of our experience, there has been no work on the data mining of relations containing indefinite information, possibly due to the lack of availability of indefinite data. Catalytic relations, introduced in (Hale and Shenoi, 1995), those that are essentially the join of two or more relations, provide a possible avenue for indefinite information data mining if the join performed does not create the cartesian product but instead creates disjunction within cells which do not agree on their attributes, referred to in Section 5.4.3.

The work of (Vianu, 1987; Vianu, 1988) was seminal in the field of temporal dependencies. Possible extensions discussed herein for NDs in 2.2.8 warrant further study. Nearly all work on dependency mining (Mannila and Räihä, 1992a; Kivinen and Mannila, 1995; Savnik and Flach, 1993; Bell and Brockhausen, 1995; Huhtala et al., 1998) presents studies of the efficiency of dependency mining, frequently noting that large number of dependencies were discovered in relations. For example, (Savnik and Flach, 1993) reports the discovery of 1191 FDs in a relation with 471 tuples over 17 attributes. Obviously the majority of these FDs discovered will be near trivial due to large left hand side attribute sets functioning as keys. We remark that there has been little work assessing the real value of FD discovery in the data mining process. Such potentially meaningless FDs also motivate the use of a user supplied template to define FD approximations to dependencies which the user is interested in.

From the work of (Mannila and Toivonen, 1996a; Berger and Tuzhilin, 1998) we note the required restriction to simple pattern discovery otherwise an NP-complete problem is faced. Therefore it is necessary to restrict the discovery process. We choose, in Chapters 5 and 6, to restrict our discovery to patterns which correspond to temporal properties (Manna and Pnueli, 1992). We cite the work of (Das et al., 1998) and (Berger and Tuzhilin, 1998) as having closely related goals to our own work though their methodologies are different.

CHAPTER 3

Numerical Dependencies in Databases and Data Mining

We now concentrate on introducing Numerical Dependencies (NDs) and related theoretical and practical issues so that we are fully able to appreciate later work utilising NDs in indefinite and temporal relations.

Initially, in Section 3.1 we formalise the lattice of NDs. We also show how ND values may be *uninformative* for a given relation and define *mean* NDs to combat such problems. Section 3.2 introduces the chase for NDs as a precursor to the chase for NDs in indefinite relations in Chapter 4. We discuss and extend the ND axiomatisation of (Grant and Minker, 1985b) in Section 3.3 and show that the chase for NDs as an inference procedure is sound and complete. An algorithm for data mining of NDs in Section 3.4 is presented with respect to related work. Section 3.5 discusses, briefly, an evolutionary algorithm for database design presented in (Collopy and Levene, 1998a), which uses NDs within a hill-climbing procedure. We conclude with a general overview of this chapter and its implications for data mining in Section 3.6, together with a note on possible Armstrong relations for NDs.

3.1 Approximating FDs with NDs

We now define the lattice of NDs and then show how this may be used to form a metric for approximating proximity to a given FD set.

3.1.1 The Lattice of NDs

Firstly, we present the lattice of NDs. We begin with Definition 3.1.1 which is then used to define the lattice of NDs and Definition 3.1.2 which is used in our algorithm for climbing the lattice.

Definition 3.1.1 (More functional set of NDs) A set of NDs N_1 over R is *more functional* than a set of NDs N_2 over R, denoted by $N_2 \sqsubseteq N_1$, whenever $X \rightarrow^{k_2} Y \in N_2$ if and only if

$X \rightarrow^{k_1} Y \in N_1$ and $k_1 \leq k_2$. □

The set-theoretic relation, more functional than, is a partial order in the sets of NDs. Assume that we are considering only sets of NDs over a schema R which are more functional than a given set of NDs, N over R, each of the form $X \rightarrow^k Y$, for some $k \geq 1$. Then the family of sets of NDs that are more functional than N form a lattice whose bottom element is N and whose top element is the set of FDs induced by N, i.e. $\{X \rightarrow Y \mid X \rightarrow^k Y \in N\}$. The *least upper bound, lub*, of N_1 and N_2 is the set of NDs $\{X \rightarrow^{min(k_1,k_2)} Y \mid X \rightarrow^{k_1} Y \in N_1 \text{ and } X \rightarrow^{k_2} Y \in N_2\}$, where $min(k_1, k_2)$ is the minimum of k_1 and k_2, and the *greatest lower bound, glb*, of N_1 and N_2 is defined similarly using maximum. We call the lattice, whose top element is the set of FDs F over R and whose bottom element is the set of NDs $\{X \rightarrow^m Y \mid X \rightarrow Y \in F\}$, $\mathcal{L}_m(F)$ (or simply $\mathcal{L}_m$ if F is understood from context), with $m \geq 1$.

Therefore, we can *approximate* a set of FDs F by a set of NDs N such that $N \sqsubseteq F$. The *closer* N is to F in $\mathcal{L}_m$ the better the approximation is. From now on we let $\mathcal{L}_m$ *be the lattice of NDs whose top element is* F and, for a relation r, assume that $| r | = m + 1$, with $m \geq 1$. In Figure 3.1 we present a lattice for two NDs whose attributes are not specified, over a relation with a maximum domain size of 4 in the right hand side of each ND. The lattice size significantly increases with more NDs and larger domain sizes. The probability of an ND $X \rightarrow^k Y$ being satisfied in a relation r tends to one as k gets closer to $| r | -1$.

Definition 3.1.2 (Covered By) We say that N_2 is *covered by* N_1, denoted by $N_2 \prec N_1$, where $N_1, N_2 \in \mathcal{L}_m$, if $N_1 \neq N_2$, $N_2 \sqsubseteq N_1$ and $\forall N' \in \mathcal{L}_m$ such that $N_2 \sqsubseteq N' \sqsubseteq N_1$ we have $N' = N_2$. □

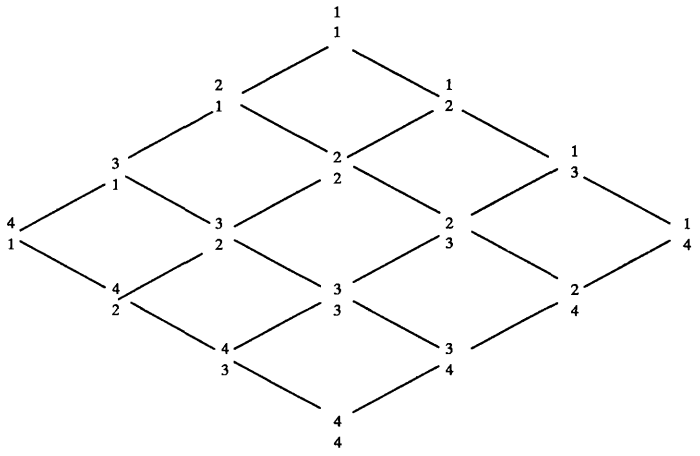

Figure 3.1: Lattice of NDs for a relation of 2 FDs (not shown) and maximum domain size of 4 for each dependency

Definition 3.1.3 (Maximal set of NDs) The *maximal* set of NDs of r with respect to F, denoted by $maximal(r, \mathrm{F})$, is the maximal set N of NDs in $\mathcal{L}_m(\mathrm{F})$ (with respect to $\sqsubseteq$) such that $r \models \mathrm{N}$. □

Given r and F, $maximal(r, \mathrm{F})$ can be computed in polynomial time in the sizes of r and F by a straightforward hill climbing procedure on $\mathcal{L}_m(\mathrm{F})$, illustrated in algorithm 2. For each $\mathrm{X} \rightarrow \mathrm{A} \in \mathrm{F}$ this procedure finds the minimal k such that $r \models \mathrm{X} \rightarrow^k \mathrm{A}$, starting from $\mathrm{X} \rightarrow^m \mathrm{A}$ which r trivially satisfies since $| r | = m$.

Definition 3.1.4 (Improvement set of a set of NDs) The *improvement set* of r with respect to F, denoted by $\mu(r, \mathrm{F})$, is defined as

$$\mu(r, \mathrm{F}) = \{\mathrm{X} \rightarrow^k \mathrm{A} \mid \mathrm{X} \rightarrow^k \mathrm{A} \in maximal(r, \mathrm{F}) \text{ and } k > 1\}.$$

Algorithm 2 returns the *improvement set* of r if any FDs satisfied in r are removed from N. □

Algorithm 2 (MU(r, F))
1. **begin**
2. $m := | r |$;
3. N := the bottom element of $\mathcal{L}_m$ (F);
4. **while** $\exists$ G such that $\mathrm{N} \prec \mathrm{G}$ and $r \models \mathrm{G}$ **do**
5. N := G;
6. **end while**
7. **return** N;
8. **end.**

Figure 3.2: The improvement algorithm for NDs

3.1.2 Similarity Measures and Numerical Dependencies

We introduce a measure for calculating the proximity of two ND sets using their position within the lattice. We show that this measure is a distance function, satisfying reflexivity and symmetry, and is also a metric, satisfying the triangle inequality. Firstly, we begin by defining the best approximation given by a set of NDs to their functional counterparts. We define the *size* of a set of NDs N to be the number of attributes appearing in N including repetitions and define a *step*, either up or down, to be exactly minus or plus one, respectively, to a single branch of one ND within an ND set.

Definition 3.1.5 (The best approximation of a set of FDs) A set of NDs N over R is the *best approximation* of a set of FDs F over R with respect to a relation r over R, with $| r | = m + 1$ (or simply the best approximation of F if r is understood from context), if $r \models \mathrm{N}$ and there does not exist a set of NDs, $N' \in \mathcal{L}_m$ such that $\mathrm{N} \prec N'$ and $r \models N'$. □

Proposition 3.1.1 (The number of NDs higher in the Lattice) Given an ND set $N = \{X_1 \rightarrow^{k_1} A_1, X_2 \rightarrow^{k_2} A_2, \ldots, X_n \rightarrow^{k_n} A_n\}$, the number of ND sets above this set in the lattice is $(k_1 \cdot k_2 \cdot \ldots \cdot k_n)$ - 1.

Proof. An ND is higher in the lattice if within a set of NDs none of the k_i branches have any values higher than any of those in the set *and* at least one of the NDs has some k_j branch value lower than one of those in the set. Each ND $X_i \rightarrow^{k_i} A_i$ within the set can take k_i values. We consider all permutations of these values to get $k_1 \cdot k_2 \cdot \ldots \cdot k_n$ from which we must ensure that the ND set values of N itself is not included to get $k_1 \cdot k_2 \cdot \ldots \cdot k_n$ - 1. □

This provides us with the basis for a distance measure between an ND set and its functional representation. However, using this technique allows, in some instances, ND sets which are the same number of steps below the FD equivalent to have different values. This is due to ND sets containing FDs or NDs which are close to being functional having less sets above them in the lattice. To illustrate, if we have two ND sets N_1, N_2 each containing two NDs such that N_1 has dependencies with 4 and 2 as branches whilst N_2 has dependencies with 3 and 3 as branches then N_1 will have fewer ND sets above it in the lattice though both are the same number of steps from their functional equivalent, shown in Figure 3.1. We now introduce the metric we used in our simulations and note that if we are interested in comparing ND sets with either more or less *near* FDs we can refer to the above measure whenever the metric provides the same distance.

In the following definition *distance* is defined as the number of *steps* in the lattice. In Definition 3.1.6 we use proposition 3.1.2 to prove that the denominator for this measure is normalised for any two NDs within a given relation. $p(N_1, N_2)$ provides a suitable measure of proximity between two ND sets. We use the measure of Definition 3.1.6 in our simulations presented in Chapter 4 in the following form: Given a set of NDs N_1 and a set of FDs F, which N_1 approximates, then the proximity between the two dependency sets is given by $p(N_1, F)$.

Definition 3.1.6 (Proximity between two ND sets) Given two sets of NDs N_1 and N_2 we define the metric as follows:

$$p(N_1, N_2) = \frac{\Sigma_{i=1,2} \text{ Distance from } N_i \text{ to } lub\{N_1, N_2\}}{\text{Maximum distance between any two ND sets to their } lub \text{ in the lattice}} \quad \square$$

We define the bottom of the lattice to be the set of NDs with each branching factor equivalent to the domain size of the attribute on the right hand side of each ND, assuming a finite domain size.

Proposition 3.1.2 The maximum distance between any two points in the lattice to their *lub* is always equivalent to the distance from the bottom to the top of the lattice.

Proof. We prove this by induction on the NDs within the two ND sets.

(*Basis*): We see that if N_1 and N_2 are empty then the result is immediate.

(*Induction*): We have two ND sets N_1 and N_2 which are distance d apart where $d \leq q$ and q is the maximum distance apart between any two ND sets. We add an ND $X \rightarrow^{k_1} Y$ to N_1 and $X \rightarrow^{k_2} Y$ to N_2 which differ only on their branching factor. Without loss of generality, if $k_1 < k_2$ then the distance apart between N_1 and N_2 becomes $d + k_2 - k_1$. This remains less than or equal to the maximum distance apart which is $q + k_2' - k_1'$ where, without loss of generality, $k_1' = 1$ (it is an FD) and k_2' is at the bottom of the lattice. □

The measure p is a distance function given that the distance between two NDs is zero only when they are equivalent and that $p(n_1, n_2) = p(n_2, n_1)$ always holds. It also satisfies the triangle inequality, whose proof we now outline. This implies therefore that p is a metric implying that sets with a common value can be compared.

Theorem 3.1.3 Given three ND sets, N_1, N_2, and N_3, $p(N_1, N_2) + p(N_2, N_3) \geq p(N_1, N_3)$.

Proof. We show that if N_1, N_2 and N_3 are non-empty and the triangle inequality holds then the addition of a new ND to each set which may differ only on its branching factor will still satisfy the triangle inequality. Assume we add three NDs $X \rightarrow^{k_i} A$ with $i = 1, 2, 3$ to N_1, N_2 and N_3, respectively. We also assume, without loss of generality, that $k_1 < k_3$. We denote each ND set $N_i \cup \{X \rightarrow^{k_i} A\}$ by N_i' for $i = 1, 2, 3$. We perform induction on the NDs in each set.

(*Basis*): If $N_1 = N_2 = N_3 = \emptyset$ then the result is immediate.

(*Induction*): We assume that $k_2 \leq k_1$, then $p(N_1', N_2')$ = distance from N_1 to $lub(N_1, N_2)$ + distance from N_2 to $lub(N_1, N_2)$ + $k_2 - k_1$. Similarly for $p(N_2', N_3')$ and $p(N_1', N_3')$ we have the additional components, $k_3 - k_2$ and $k_3 - k_1$. Therefore, we have $p(N_1', N_2') + p(N_2', N_3') = p(N_1, N_2) + k_1 - k_2 + p(N_2, N_3) + k_3 - k_2$ and $p(N_1', N_3') = p(N_1, N_3) + k_3 - k_1$. We know that $p(N_1, N_2) + p(N_2, N_3) \geq p(N_1, N_3)$ holds and we see that $k_1 - k_2 + k_3 - k_2 \geq k_3 - k_1$ holds if $k_1 \geq k_2$ which is true, based on our initial assumption. We can similarly prove the triangle inequality for the case when $k_1 \leq k_2$. □

3.1.3 Partitioning a Relation for Mean NDs

In many data mining tools it is important that there exist measures which accurately reflect the content of the database; this motivates us to define mean ND set satisfaction for some situations like that of Example 3.1.1.

In Chapter 2 we presented Definition 2.2.28 for partitioning of a relation into blocks for an ND X $\rightarrow^k$ Y which agree on X. The satisfaction of an ND X $\rightarrow^k$ Y implies only that there exists at

least one partition $\mathcal{B}$ which contains at least k tuples with at most k different Y-values. There may however be numerous other partitions on X which may have far less than k different Y-values and so the partition $\mathcal{B}$ dominates the relation and presents an inaccurate representation of the proximity to FD set satisfaction. We therefore define the *mean numerical dependency*.

Definition 3.1.7 (Mean Numerical dependency) A *mean numerical dependency* over R (or simply a mean ND) is a statement of the form $X \rightarrow^{\bar{k}} Y$, where $X, Y \subseteq R$ and $\bar{k} \geq 1$. We refer to $\bar{k}$ as the *mean branching factor*. □

Definition 3.1.8 (Satisfaction of a Mean ND) Let r be a relation over R. An ND $X \rightarrow^{\bar{k}} Y$ is *satisfied* in r, denoted by $r \models X \rightarrow^{\bar{k}} Y$, such that r is partitioned into blocks $\{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_w\}$ with respect to $X \rightarrow Y$ such that $\bar{k} = \frac{\sum_{i=1}^{w} |\pi_Y(\mathcal{B}_i)|}{w}$. A set of averaged NDs $\bar{N}$ is *satisfied* in r, denoted by $r \models \bar{N}$, whenever $\forall\, X \rightarrow^{\bar{k}} Y \in \bar{N}, r \models X \rightarrow^{\bar{k}} Y$. □

Example 3.1.1 We assume that a relation r over AB satisfies the ND $A \rightarrow^{14} B$ as its closest approximation to the FD $A \rightarrow B$. However r may, for example, only contain three partitions $\{\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3\}$ with each partition satisfying the NDs $A \rightarrow^{14} B$, $A \rightarrow^{1} B$, and $A \rightarrow^{1} B$, respectively. We note the last two are satisfied functionally. The mean ND set satisfaction is $A \rightarrow^{5.\bar{6}7} B$. Within a block $\mathcal{B}$ the number of tuples is not related to the branching factor value, given by $| \pi_B(\mathcal{B}) |$.

In our work on indefinite information in relations we remark that we are interested in exact ND set satisfaction only, given the nature of the problem. In the data mining of NDs in standard and temporal relations we may often be interested in the mean ND set satisfaction value. We note that if this value is vastly different from the exact satisfaction value then it is likely that one or more partitions from the relation *dominate* and remaining partitions will satisfy the NDs more functionally.

3.2 The Chase Procedure for NDs

We now show how CHASE(r, F) can be generalised to CHASE(r, N), where N is a set of NDs over R, shown in Figure 3.3.

We leave it to the reader to verify that when $k = 1$, i.e. $X \rightarrow^{k} Y$ is an FD, then CHASE(r, N) reduces to CHASE(r, F).

Lemma 3.2.1 Algorithm 3 terminates.

Proof. No new values are introduced into the algorithm at any step and therefore the algorithm must halt after executing the while loop a finite number of times. □

Algorithm 3 (CHASE(r, N))
1. **begin**
2. Result := r;
3. Tmp := $\emptyset$;
4. **while** Tmp $\neq$ Result **do**
5. Tmp := Result;
6. **if** $\exists$ X $\rightarrow^k$ Y $\in$ N, $\exists t_1, t_2, \ldots, t_k, t_{k+1} \in$ Result such that
   t_1[X] = t_2[X] = ... = t_k[X] = t_{k+1}[X]
   but t_1[Y] $\neq$ t_2[Y] $\neq$ ... $\neq$ t_k[Y] $\neq$ t_{k+1}[Y] **then**
7. **for each** $A \in$ Y$-$X **do**
8. t_i[A],t_j[A] := max(t_i[A],t_j[A]) for two distinct values $i, j \in 1, \ldots, k+1$;
9. **end for**
10. **end if**
11. **end while**
12. **return** Result;
13. **end.**

Figure 3.3: The Chase procedure for NDs

Theorem 3.2.2 Given a set of NDs, N, then $\forall n \in N$, CHASE(r, N) $\models n$.

Proof. Direct from the definitions of the algorithm and of ND satisfaction. □

We also note in theorem 3.2.2 that if $r \not\models$ N, for a relation r and an ND set N then CHASE(r, N) = r. We return to the chase procedure and show how it can be used as an inference procedure in Section 3.3.2, after discussion of the axiomatisation of NDs.

3.3 Inferences for Numerical Dependencies

The axiom system given in (Grant and Minker, 1985b) is shown to be sound and complete only in the special cases of, for a schema R, either |R| $\leq$ 3 or when the number of NDs with $k > 1$ is at most one. (Grant and Minker, 1985a) extends this result and shows that there is no finite sound and complete axiomatisation for NDs.

3.3.1 ND Axiomatisation

NDs allow a more general dependency relation than functional dependencies. (Grant and Minker, 1985b; Grant and Minker, 1985a) introduce NDs with regard to obtaining normal forms which avoid or minimise redundancy, from a database with k-dependency constraints. Grant and Minker also provide an axiomatisation for NDs which is a generalisation of the Armstrong axioms for FDs. (Grant and Minker, 1985b) presents a set of sound inference rules for NDs. (Grant and Minker, 1985a) shows that there does not exist a *finite* set of sound and complete inference rules for Numerical Dependencies. These axioms are shown to be complete for relations which have, at most, 3 attributes. It is shown that any relation with more than 3 attributes is not complete.

We show in Section 3.3.2 how the chase may be used as an inference procedure. If the relation utilises only FDs then the axioms contain the Armstrong rules as a subset.

For clarity, we now present inference rules 1-5 from (Grant and Minker, 1985a):

R1 If Y $\subseteq$ X then infer X $\rightarrow$ Y

R2 From X $\rightarrow^k$ Y infer ZX $\rightarrow^k$ ZY

R3(a) From X $\rightarrow^k$ Y and Y $\rightarrow^j$ Z infer X $\rightarrow^{k \cdot j}$ YZ

R3(b) From X $\rightarrow^k$ Y and Y $\rightarrow^j$ Z infer X $\rightarrow^{k \cdot j}$ Z

R4 From X $\rightarrow^k$ Y infer X $\rightarrow^{k+1}$ Y

R5$_m$ From $\{X \rightarrow^2 Y_i \mid 1 \leq i \leq 3m-2\}$
$\cup \quad \{Y_{i1}Y_{i2}\ldots Y_{im} \rightarrow Z \mid 1 \leq i1 < i2 < \ldots < im \leq 3m-2\}$ infer $X \rightarrow^2 Z$

Rules R1,R2 and R3(b) are extensions of the axioms for FDs. We now present another inference rule which generalises the rule R5$_m$ of (Grant and Minker, 1985a), R6$_{k,m}$, which can be viewed as a generalised transitivity rule for NDs wherein a bound on the number of attributes required for inference is created based on the branching factor of the NDs *and* the number of attributes on the left hand side of the FD which determines attribute Z. R6$_{k,m}$ is a useful extension to the class of transitive axioms for NDs.

(R6$_{k,m}$): From $\{X \rightarrow^k Y_i \mid 1 \leq i \leq \eta\} \quad \cup$
$\{Y_{i1}Y_{i2}\ldots Y_{im} \rightarrow Z \mid 1 \leq i1 < i2 < \ldots < im \leq \eta\}$
we can infer $X \rightarrow^k Z$ where $\eta = (m-1)\binom{k+1}{2} + 1$

Theorem 3.3.1 Each rule R6$_{k,m}$ is sound and has minimal hypothesis.

X	Y_i	Y_2	...	Y_η	Z
1					1
1					2
$\vdots$					$\vdots$
1					$k+1$

Table 3.1: Example relation for proof of axiom R6$_{k,m}$

Proof. We assume that we have a relation r with $\eta + 2$ attributes and $k + 1$ cells, as in Table 3.1, and that X agrees on all of its $k + 1$ cells. We also assume, from R6$_{k,m}$, that

$X \rightarrow^k Y_i$ holds for all $1 \leq i \leq \eta$ and that for a given m all possible FDs of the form $Y_{i1}Y_{i2}\ldots Y_{im} \rightarrow Z$ hold for $1 \leq i1 < i2 < \ldots < im \leq \eta$ and that $X \rightarrow^k Z$ does not hold. Each $X \rightarrow^k Y_i$ implies that each Y_i attribute must agree on at least two cells. In a relation with $k+1$ tuples there are $\binom{k+1}{2}$ ways in which a single attribute may agree on two tuples. For some set of attributes $Y_{i1}Y_{i2}\ldots Y_{im}$ in Y_1 to Y_η if $\eta = (m-1)\binom{k+1}{2} + 1$, then at least one FD of the form $Y_{i1}Y_{i2}\ldots Y_{im} \rightarrow Z$ must agree on all of its Y_i attributes. This is due to exhaustion of all possible combinations on which two tuples may agree. Whichever tuples agree on all m attributes imply that Z also agrees on these attributes. Therefore there may not be $k+1$ different values on Z and we have a contradiction.

We prove the minimal hypothesis by noting that if any FD $X \rightarrow^k Y$ or $Y_{i1}Y_{i2}\ldots Y_{im} \rightarrow Z$ for $1 \leq i1 < i2 < \ldots < im \leq \eta$ is omitted from our requirements then there exists a counterexample which does not imply $X \rightarrow^k Z$ for some combination of values on the attributes in Y_1 to Y_η. □

We can also prove this as for $R5_m$ in (Grant and Minker, 1985a) by explicitly proving that no counterexample relation exists. We note that when $k = 1$ then $R6_{k,m}$ reduces to transitivity of FDs, and when $k = 2$ then $\eta = 3m - 2$, the figure given in (Grant and Minker, 1985a) for $R5_m$.

3.3.2 The Chase as an Inference Procedure

We prove that the chase is a sound and complete inference procedure for NDs. In the sequel, we assume that NDs have singleton right hand sides.

Given a set of NDs N and an ND σ, we apply the chase as an inference tool to discover if $N \models \sigma$. We create a relation r_σ which for $\sigma = X \rightarrow^k A$ has $k+1$ tuples with $k+1$ different values on attribute set A, all values on X equivalent and all values in $R \backslash XA$ unique. We need to consider all possible iterations of the chase procedure, presented in algorithm 3, for a relation r_σ for the inference procedure to be sound and complete, given that one instance of the chase may not terminate with a unique end result, known as the *Church-Rosser* property (Maier et al., 1979). We refer to each complete application of the chase as a *chase sequence.*

	X_1	...	X_m	A	B_1	...	B_m
t_1	1	...	1	1	1	...	1
t_2	1	...	1	2	2	...	2
⋮	⋮	⋱	⋮	⋮	⋮	⋱	⋮
t_{k+1}	1	...	1	$k+1$	$k+1$	...	$k+1$

Table 3.2: Relation to be chased by ND set N with $\sigma = X \rightarrow^k A$, X = $\{X_1, \ldots, X_m\}$, R \ XA = $\{B_1, \ldots, B_m\}$ and m = | R \ XA |

We motivate theorem 3.3.3 by showing an example relation where different sequences

of tuples modified by the chase produce different results. We show this for an ND set N = { $X \rightarrow^2 B_1$, $X \rightarrow^2 B_2$, $B_1B_2 \rightarrow A$ }, $\sigma = X \rightarrow^2 A$, and relation r_1 in Table 3.3. For two different chase sequences, with different tuples modified, we have $r_1^c \not\models X \rightarrow^2 A$, shown in Table 3.4, and $r_1^c \models X \rightarrow^2 A$, shown in Table 3.5. Therefore N $\not\models X \rightarrow^2 A$, which may not have been discovered if we had only examined one chase sequence.

X	B_1	B_2	A
1	1	1	1
1	2	2	2
1	3	3	3

Table 3.3: r_1 before CHASE procedure

X	B_1	B_2	A
1	1	2	1
1	3	2	2
1	3	3	3

Table 3.4: Example CHASE(r_1,N) after CHASE procedure

X	B_1	B_2	A
1	1	1	1
1	3	3	3
1	3	3	3

Table 3.5: Counterexample CHASE(r_2,N) after CHASE procedure

Theorem 3.3.2 requires the notion of containment mapping cf. (Atzeni and De Antonellis, 1993). We define a function *dom* which returns the active domain of a relation.

Definition 3.3.1 (Containment Mapping) A containment mapping ϕ from r_σ to a relation r has each value in $dom(r_\sigma)$ mapped by ϕ to a value in $dom(r)$. This is extended to tuples over $R = A_1A_2 \ldots A_m$ as $\phi(t) = \phi(t[A_1]), \phi(t[A_2]), \ldots, \phi(t[A_m])$ and extends to a relation as $\phi(r) = \{\phi(t) | t \in r\}$. □

Theorem 3.3.2 Given a set of Numerical Dependencies N and a ND $\sigma = X \rightarrow^k A$, N $\models \sigma$ iff $\neg\exists t_1^c[A] \neq t_2^c[A] \neq \ldots \neq t_{k+1}^c[A]$ where $t_1^c, t_2^c, \ldots, t_{k+1}^c \in r_\sigma^c$ and r_σ^c = chase(r_σ,N) for all possible sequences of the chase.

Proof. (if) We assume that $A \not\subseteq X$. We let r be a relation over R such that $r \models N$; we show that $r \models \sigma$. Let $t_1, t_2, \ldots, t_{k+1} \in r$ such that $t_1[X] = t_2[X] = \ldots = t_{k+1}[X]$. We claim that for some $i, j \in \{1, 2, \ldots, k+1\}$ there exists $t_i[A] = t_j[A]$.

Let ϕ be a containment mapping from r_σ $\phi(u_1) = t_1$, $\phi(u_2) = t_2$, ..., $\phi(u_{k+1}) = t_{k+1}$. We shall prove that ϕ is additionally a containment mapping from CHASE(r_σ, N) to r so that for some $i, j \in r_\sigma^c$, $\phi(t_i^c[A]) = \phi(t_j^c[A])$ implies $t_i[A] = t_j[A]$. This is the case for all possible chase sequences on r_σ.

We prove this by induction on the number of steps, s, required to compute CHASE(r_σ, N).

(*Basis*): If $s = 1$ then for some ND $W \rightarrow^k B \in N$ two values are equated in B where $W \subseteq X$ and $A = B$ so therefore $u_i[A] = u_j[A]$ implying that $\phi(u_i[A]) = t_i[A]$ and $\phi(u_j[A]) = t_j[A]$ and so $t_i[A] = t_j[A]$ for some i, j in r.

(*Induction*): We assume that the result holds when s steps of the chase procedure are required. We now prove it to be true when $s + 1$ chase steps are required. We let the $s + 1$ chase step be for an ND $W \rightarrow^k B$ and u_i, u_j be two tuples in r_σ such that the $s+1$ step either modifies u_i or u_j. Thus, for B either $u_i[B]$ or $u_j[B]$ is modified so that $u_i[B] = u_j[B] = t_i^c[A] = t_j^c[A]$. Now, given $u_i[W] = u_j[W]$ then $\phi(u_i[W]) = \phi(u_j[W])$ by the definition of containment mapping. Therefore $\phi(t_i^c[A]) = \phi(t_j^c[A]) = \phi(u_i[B]) = \phi(u_j[B]) = t_i[A] = t_j[A]$ holds, since $r_\sigma \models W \rightarrow^k B$. Given that ϕ only differs from the result of s steps on $u_i[B]$ or $u_j[B]$ it is a containment mapping from r_σ^c in all possible chase sequences. In any sequence, if $t_i^c[A] = t_j^c[A]$ we have, by definition of ϕ, $t_i[A] = t_j[A]$.

(only-if) If, in some chase sequence, there does not exist $t_i^c[A] = t_j^c[A]$ for some $i, j \in 1, 2, \ldots, k+1$ then we can construct a relation r which satisfies all $n \in N$ but violates σ. Alternatively, the chase for r_σ can be shown to be isomorphic to a relation r which satisfies N but violates X $\rightarrow^k$ A by mapping each value in r_σ^c to a value in $dom(r)$. □

Corollary 3.3.3 The chase inference procedure for Numerical Dependencies is sound and complete.

Proof. Soundness and completeness of the chase as an inference procedure is a corollary of theorem 3.3.3. □

The existence of a sound and complete chase procedure shows that implication on NDs is decidable. The implication problem for NDs is in co-NP, used to denote that the complementary no/yes problem is in the set NP (Garey and Johnson, 1979); we know that the converse problem is NP given that we can *guess* a relation (equivalent to a complete sequence of the chase procedure) r_σ^c and verify in polynomial time whether or not $r_\sigma^c \models \sigma$. NP-completeness remains an open problem.

3.3.3 Armstrong Relations for NDs

Within a finite relation there does not exist an Armstrong Relation, defined for FDs in Definition 2.2.20, for a set of NDs N, unless N contains all possible combinations of attribute sets. We may prove this as follows. Assume that we are given a set of NDs N and that σ is an ND X $\rightarrow^k$ Y such that N $\not\models \sigma$. For any relation r such that $r \models$ N we have, at least, $r \models \sigma$ where $k = \mid r \mid$.

We therefore define a weak AR for an ND set N.

Definition 3.3.2 (A Weak Armstrong Relation) A relation r is a weak AR for a set of NDs N if $r \models$ N and $\forall\sigma$ such that N $\not\models \sigma$ then $r \models \sigma$ maximally with respect to k_σ implying that r satisfies all NDs σ with a branching factor no higher than k_σ. The choice of a value for k_σ may be related to branching factors of NDs in N (or the size of the relation). □

These weak Armstrong Relations would extend the practical application of NDs within database design tools by helping designers think of what NDs may be required via examination of an actual relation.

3.4 Numerical Dependencies in Data Mining

We now briefly present data mining for numerical dependencies in standard relations. We emphasise the following:

- In contrast with functional dependency approximation data mining we are not seeking to assess what proportion of a relation satisfies a functional relationship, cf. Table 2.10 and (Huhtala et al., 1998; Kivinen and Mannila, 1995). We seek to discover generalisations of FDs when the FD may be viewed as too strict.
- In our work on temporal and indefinite relations we assume the user provides a ND set upon which we seek instances of ND satisfaction. The use of NDs in a blind discovery context would generate ND satisfying instances for all possible attribute set combinations, which is not practical due to the complexity.

3.4.1 Dependency Mining Applications

Approximation of the dependency set, possibly approximated using numerical dependencies, on a large database in existence may reveal unknown information in the form of these dependencies which may hold in the database. A recent work on the reverse-engineering, or discovery if you will, of cardinality constraints for inference of the ER-model is presented in (Soutou, 1998). From lemma 2.2.2 this is another application for ND discovery.

Applications for dependency mining include a database design tool which the database designer can use in conjunction with a possible instance of the data to be stored within the database. Inference upon this example set will then provide the designer with vital information as to possible unknown dependencies that be satisfied in the relation. The approach of Bell and Brockhausen (Bell and Brockhausen, 1995) in making inferences from the verified and invalid data dependencies is aimed at *supporting* the database designer. Example 3.4.1 shows an application of ND discovery.

Example 3.4.1 In a patient database within a hospital every patient visit is independently stored within a *patient* details relation and a *disease/symptom/treatment* relation. Given a numerical dependency specified $DISEASE \rightarrow^{10} SYMPTOM$ stating that a disease can have at most 10 symptoms, it may however be approximated that $ADDRESS\ PATIENT \rightarrow^{6} SYMPTOM$ showing that at most 6 of the symptoms can occur at the same location for a patient.

3.4.2 Mining a relation for a set of NDs

We consider only singleton right hand sides. For a relation r over R with n attributes we have $n2^{n-1}$ NDs returned by this algorithm and so in Figure 3.5 we only give results obtained from restricting the left hand sides to a given arity of attributes. Algorithm 4 uses Algorithm 2 to generate the ND satisfied from an FD template.

Algorithm 4 (ND_mine(r, R))
1. **begin**
2. ND_set := $\emptyset$;
3. **for each** A $\in$ R **do**
4. **for each** W $\in \mathcal{P}$(R - A) **do**
5. { W $\rightarrow^k$ A } = MU(r, { W $\rightarrow$ A });
6. ND_set := ND_set $\cup$ { W $\rightarrow^k$ A };
7. **end for**;
8. **end for**;
9. **return** ND_set;
10. **end.**

Figure 3.4: The ND mining algorithm

The computational complexity of algorithm 4 is O($n^2 2^{n-1} \mid r \mid log \mid r \mid$). There are $n2^{n-1}$ possible NDs and it takes time O($n \mid r \mid log \mid r \mid$) to sort a relation into partitions. We can restrict the arity of the left hand side to a size m or even restrict to singleton left and right hand sides where we have a time of O($n^3 \mid r \mid log \mid r \mid$). When the lhs of any ND is $\emptyset$ then the ND corresponds directly to the domain size. The scale of the mining can be cut down by using axioms provided in (Grant and Minker, 1985a; Grant and Minker, 1985b) when exact details are not required for dependencies of the form $W \rightarrow^k A$ where $W = YV$ and we already know $Y \rightarrow^{k_1} A$ and $V \rightarrow^{k_2} A$ which would give $k \leq k_1$ or $k \leq k_2$, depending on the larger partition.

It is also possible that we could use the chase procedure for NDs to further improve the efficiency of the algorithm such that for a set of NDs N we do not mine for an ND σ if N $\models \sigma$.

In Figure 3.5 we see how increases in the arity of the left side of the NDs increase the time required to mine for sets of NDs in increasing relation sizes. The time increases represent the additional overhead of more dependencies due to more possible attribute combinations on the left

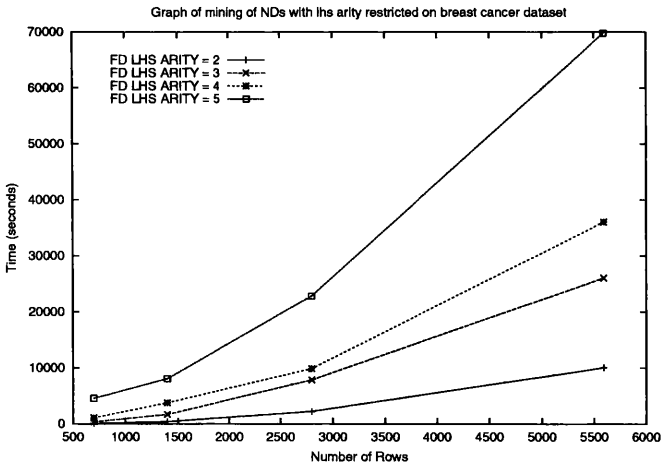

Figure 3.5: Results for mining Mean and standard NDs with arity of the lhs of each ND restricted upon the breast cancer dataset

hand side of NDs, with respect to the complete attribute set, and the time to compare additional attributes for insertion into partitions. We have applied this to the breast cancer data set (Blake et al., 1998) used in many data mining research papers. The data set has 11 attributes and 699 tuples. We increase these tuples by adding identifiers to each tuple and copying (as used by (Huhtala et al., 1998)). Obviously in a relation with 11 attributes there is little point in examining the powerset of attributes as it is highly likely that an FD with, say 7 or more, attributes on the left hand side will be satisfied functionally and will be meaningless. This point is reiterated in (Savnik and Flach, 1993).

3.4.3 Mining a relation for a set of Mean NDs

The time to mine for a set of mean NDs is the same as for standard NDs given that we need to examine each partition as before. The difference between the satisfied mean NDs and the standard ND satisfaction may tell us much about the dataset which we highlight in Example 3.4.2.

Example 3.4.2 In the breast cancer database there is an attribute *marginal_adhesion* with a domain of 10 elements. The ND $mitosis \rightarrow^{10} marginal\_adhesion$ tells us only that there is at least one partition containing all 10 elements on an attribute value of *mitosis*. The mean ND $mitosis \rightarrow^{5.\bar{6}7} marginal\_adhesion$ tells us most partitions have fewer elements.

3.5 Evolving Example Relations to Satisfy FDs

We now briefly present an example of applying NDs to approximate FDs in an evolutionary hill-climbing algorithm for creating probabilistic example relations for use within database design applications. This work summarises (Collopy and Levene, 1998d), and, as indicated in Section 2.2.4, is related to work in the Design-By-Example project of (Mannila and Räihä, 1986).

Example relations satisfying a given set of integrity constraints such as FDs are important during the database design activity in order to guide the designer towards the specification of a correct set of constraints for the application in hand (Silva and Melkanoff, 1981).

3.5.1 Motivation

If the example relation shown to the database designer is too large then the designer will *not* be able to assimilate all the knowledge embedded in that relation. Thus it would be useful to be able to generate random examples relations that satisfy F and whose maximal cardinality is specified by the database designer; in general, such an example may not be an Armstrong relation.

Our algorithm is evolutionary in that it incorporates a stochastic approach for altering a relation by a *mutation* operation. The algorithm proceeds as follows; initially a relation is randomly generated following the input of the designer and a given FD set. This relation is then mutated based on a probabilistic selection of an unsatisfied FD from the given set and an attribute which assists violation of this FD in the relation. We use NDs as an approximation of the unsatisfied FDs in the relation. The mutations steer the relation towards a final state wherein all of the FDs in the specified set are satisfied. It is a simple algorithm, and indeed a basic tenet of evolutionary programming is to create algorithms which do not constrain evolution too severely, much like organic evolution (Bäck and Schwefel, 1993). All evolved relations are then mined using a quality function, defined in 2.3.3, whose criterion is exact satisfaction of the given FD set.

A deterministic approach used to generate an Armstrong relation (Algorithm 14.2, (Mannila and Räihä, 1992a)) has the severe drawback in that the same relation is generated every time the algorithm runs. Our probabilistic approach is advantageous in that different example relations may be generated from equivalent domain sizes and the tuple size may be increased or decreased by the designer as desired. Moreover, as long as the number of tuples exceed the minimum size required for an Armstrong relation (Beeri et al., 1984; Mannila and Räihä, 1986) then one may be returned, although this is not guaranteed. Below this number and a deterministic approach fails whereas our evolutionary approach complies with the desires of the user and returns a relation which, if selected from a batch or *population* of evolutions, is likely to be as high a quality as possible given the domain and tuple restriction. From the user's point of view it may often be highly beneficial to examine a smaller relation of a high quality, but less than one, as opposed to a larger Armstrong relation (with a quality of one). Simulations emphasised the validity of this approach within database design, showing that many varying relations can be efficiently evolved for an FD set with numerous domain and tuple inputs. They also showed that it is extremely useful to know the quality of an example relation, and additionally that Armstrong relations are

often formed within a batch.

3.5.2 Mutating relations

Herein, we present an algorithm for *mutating* a relation. Informally, given a relation which does not satisfy F, MUTATE(r, F) randomly selects an ND, say $X \rightarrow^k A$, in the improvement set of r and stochastically modifies some of the tuple values in r. We then define the syntactic property of non-interfering NDs and show that if the selected ND and another ND $Y \rightarrow^g C$ in $maximal(r,$ F) are *non-interfering*, see Definition 3.5.2 then, after the mutation, r will still satisfy $Y \rightarrow^g C$. The non-interference property is important, since if we evolve a relation to satisfy a set of FDs by iterating the mutation operation, then the evolution process will be more efficient when the NDs in $maximal(r,$ F) are non-interfering.

The *mutation* of a relation over R with respect to a set of FDs over R denoted by MUTATE(r, F), is defined as the relation resulting from invoking Algorithm 5 presented below. In this algorithm we use LHS to denote the left hand side of an ND and RHS to denote the right hand side. This random selection of a side removes any bias which might otherwise have been incurred if the selection of an attribute to mutate were taken over the whole ND, given that the left hand side may be any length less than or equal to | R | but the right hand side is always singleton. $\mathcal{D}$ denotes the domain of values in the relation r.

The following definition provides us with a measure of how useful a mutation is in the evolution of a relation to satisfy a set of FDs.

Definition 3.5.1 (Useful, neutral and damaging mutations) Let s be the relation resulting from the mutation MUTATE(r, F). Then a mutation such as s is said to be *useful*, *neutral* or *damaging*, respectively, for an ND $Y \rightarrow^g C$, if the number of blocks $\mathcal{B}_i$ in the partitioning of s with respect to $Y \rightarrow^g C$ such that $\mathcal{B}_i \not\models Y \rightarrow^g C$ is less than, equal to or greater than, respectively, the number of blocks $\mathcal{B}_i$ in the partitioning of r with respect to $Y \rightarrow^g C$ such that $\mathcal{B}_i \not\models Y \rightarrow^g C$. □

Definition 3.5.2 (Non-interfering NDs) Two NDs $X \rightarrow^k A$ and $Y \rightarrow^g C$ are said to be *non-interfering* if either A = C and Y = X, or $YC \cap XA = \emptyset$, or $A \neq C$, X = C and YC = R. □

We call a set of NDs N over R such that every pair of FDs in N is non-interfering a *non-interfering* set of NDs. An attribute B in the left-hand side of an FD $X \rightarrow A$ is said to be *redundant* with respect to a set of FD F over R, if $A \in (X-B)^+$. Assuming that no left-hand sides of FDs in F have redundant attributes, it can be shown that when $X \rightarrow^k A$ is the ND chosen at line 3

Algorithm 5 (MUTATE(r, F))
1. **begin**
2. Result := r;
3. Uniformly randomly select an ND $X \rightarrow^k A \in MU(r, F)$ with $k > 1$;
4. Uniformly randomly select a tuple $t \in r$;
5. **if** $r[X, t[X]] \models X \rightarrow^{k-1} A$ **then**
6. **return** r;
7. **end if**
8. Uniformly randomly select RHS or LHS of ND
9. **if** LHS **then**
10. Uniformly randomly select an attribute $B \in X$
11. Uniformly randomly select a value $v \in \mathcal{D} - \{t[B]\}$;
12. **else** % B = A
13. B := A;
14. Uniformly randomly select a value $v \in \pi_A(r[X,t[X]]) - \{t[A]\}$;
15. **end if**
16. **for each** $u \in r[X, t[X]]$ such that $u[XA] = t[XA]$ **do**
17. $u[B] := v$;
18. **end for**
19. **if** Result $\models X \rightarrow^k A$ **then**
20. **return** Result;
21. **else**
22. **return** r;
23. **end if**
24. **end.**

Figure 3.6: The MUTATE procedure for evolving relations

of Algorithm 5, then the probability that any mutation MUTATE(r, F) is neutral for $Y \rightarrow^g C \in maximal(r, F)$, is at least $1/|XA|$. At times, it is necessary to accept mutations that are damaging to some of the NDs in N.

Theorem 3.5.1 Assuming that $X \rightarrow^k A$ is the ND chosen at line 3 of Algorithm 5, then for all relations r over R, any mutation MUTATE(r, F) is neutral for $Y \rightarrow^g C \in maximal(r, F)$, if and only if $X \rightarrow^k A$ and $Y \rightarrow^g C$ are non-interfering NDs.

Proof. We prove this by considering all possible relationships between XA and YC in example relations, given in (Collopy and Levene, 1996).

If. The only nontrivial case to consider is when $A \neq C$, $X = \{C\}$ and $YC = R$, implying that $A \in Y$. If the attribute chosen for mutation is A, then equating two or more A-values is neutral for $Y \rightarrow^g C$, since the C-values of the all the tuples, $u \in r[X, t[X]]$, are equal. On the other hand, if the attribute chosen for mutation is C, then forcing two or more C-values to be unequal is neutral for $Y \rightarrow^g C$, since there can only be one tuple in r having the same YC-values due to the fact that

YC = R.

Only if. We prove the result by contraposition, considering the various cases.

Case 1.1. Suppose that A = C and X and Y are incomparable, i.e. $X \not\subseteq Y$ and $Y \not\subseteq X$. Let X be the singleton B, Y be the singleton D, and r be the relation over ABD, shown in Table 3.6. Then, it can easily be verified that $r \not\models B \rightarrow A$ but $r \models D \rightarrow A$. On the other hand, the relation s shown in Table 3.7, which is a mutation resulting from MUTATE(r, F) assuming that $B \rightarrow^2 A$ is the ND chosen at line 3 of Algorithm 5, is damaging for $D \rightarrow A$, since $s \not\models D \rightarrow A$.

A	B	D
0	0	1
1	0	0
1	1	0

Table 3.6: Example relation for Case 1.1.

A	B	D
0	0	1
0	0	0
1	1	0

Table 3.7: A mutation of r shown in Table 3.6

Case 1.2. Suppose that A = C and $Y \subset X$, i.e. Y is a proper subset of X. Let Y be the singleton B, X = DB, and r be the relation over ABD, shown in Table 3.8. Then, it can easily be verified that $r \not\models DB \rightarrow A$ but $r \models B \rightarrow^2 A$. On the other hand, the relation s shown in Table 3.9, which is a mutation resulting from MUTATE(r, F) assuming that $DB \rightarrow^2 A$ is the ND chosen at line 3 of Algorithm 5, is damaging for $B \rightarrow^2 A$, since $s \not\models B \rightarrow^2 A$.

A	B	D
2	0	0
0	0	0
0	1	1
1	1	1

Table 3.8: Example relation for Case 1.2.

A	B	D
2	1	0
0	0	0
0	1	1
1	1	1

Table 3.9: A mutation of r shown in Table 3.8

Case 1.3. Suppose that A = C and $X \subset Y$, i.e. X is a proper subset of Y.

Let X be the singleton B, Y = DB, and r be the relation over ABD, shown in Table 3.10. Then, it can easily be verified that $r \not\models B \rightarrow A$ but $r \models DB \rightarrow A$. On the other hand, the relation s shown in Table 3.11, which is a mutation resulting from MUTATE(r, F) assuming that $B \rightarrow^2 A$ is the ND chosen at line 3 of Algorithm 5, is damaging for $DB \rightarrow A$, since $s \not\models DB \rightarrow A$.

Case 2.1. Suppose that $A \neq C$, $X \neq \{C\}$ and YC = R; in this case $C \in X$ may or may not hold. Let X be either the singleton B or X = BC, Y = AB. and r be the relation over ABC, shown in

A	B	D
0	0	0
1	0	1
1	1	0

Table 3.10: Example relation for Case 1.3.

A	B	D
0	1	0
1	0	1
1	1	0

Table 3.11: A mutation of r shown in Table 3.10

Table 3.12. Then, it can easily be verified that $r \not\models BC \rightarrow A$ but $r \models AB \rightarrow C$. On the other hand, the relation s shown in Table 3.13, which is a mutation resulting from MUTATE(r, F) assuming that either $B \rightarrow^2 A$ or $BC \rightarrow^2 A$ is the ND chosen at line 3 of Algorithm 5, is damaging for $AB \rightarrow C$, since $s \not\models AB \rightarrow C$.

A	B	C
0	0	0
1	0	0
0	1	1

Table 3.12: Example relation for Case 2.1.

A	B	C
0	1	0
1	0	0
0	1	1

Table 3.13: A mutation of r shown in Table 3.12

Case 2.2. Suppose that $A \neq C$, $X = \{C\}$ and $YC \neq R$. Let X be the singleton A, Y be the singleton C, and r be the relation over ABC, shown in Table 3.14. Then, it can easily be verified that $r \not\models C \rightarrow A$ but $r \models A \rightarrow C$. On the other hand, the relation s shown in Table 3.15, which is a mutation resulting from MUTATE(r, F) assuming that $C \rightarrow^2 A$ is the ND chosen at line 3 of Algorithm 5, is damaging for $A \rightarrow C$, since $s \not\models A \rightarrow C$.

A	B	C
0	0	0
1	0	0
0	1	0

Table 3.14: Example relation for Case 2.2.

A	B	C
0	0	1
1	0	0
0	1	0

Table 3.15: A mutation of r shown in Table 3.14

Case 2.3. Suppose that $A \neq C$, $X = \{C\}$ and $A \notin Y$. Let Y be the singleton D, and r be the relation over ABC, shown in Table 3.16. Then, it can easily be verified that $r \not\models C \rightarrow A$ but $r \models$

$B \rightarrow C$. On the other hand, the relation s shown in Table 3.17, which is a mutation resulting from MUTATE(r, F) assuming that $C \rightarrow^2 A$ is the ND chosen at line 3 of Algorithm 5, is damaging for $B \rightarrow C$, since $s \not\models B \rightarrow C$. □

A	B	C
1	0	0
0	0	0

Table 3.16: Example relation for Case 2.3

A	B	C
1	0	1
0	0	0

Table 3.17: A mutation of r shown in Table 3.16

3.5.3 An Algorithm for Evolving Relations to satisfy FDs

Herein, we present our algorithm for evolving a relation r to satisfy a set of FDs F. The algorithm, ITERATE(r, F) simply iterates the mutation operation on the current state of r until the set of FDs is satisfied. The number of iterations required is denoted by q. We show in (Collopy and Levene, 1996) that there always exists a finite number of states q such that ITERATE(r, F) satisfies F with a probability of one. The *iteration* of a relation, denoted by ITERATE(r, F), is defined as the result of invoking Algorithm 6, presented below. The mutations are repeated until F is satisfied in r. We say that ITERATE(r, F) *evolves* the relation it returns in q *steps*, and that r_2 *evolves* from r_1 if ITERATE(r_1, F) evolves r_2.

Algorithm 6 (ITERATE(r, F))

```
1.  begin
2.    Result := r;
3.    q := 0;
3.    while Result ⊭ F do
4.      Result := MUTATE(Result,F);
5.      q := q + 1;
6.    end while;
7.    return Result, q;
8.  end.
```

Figure 3.7: The ITERATE procedure for evolving relations

3.5.4 Simulation Results

We now detail the simulations conducted to examine the viability of evolving example relations from an initial random relation. The designer can select and vary the maximum tuple size of an example relation as well as the maximum domain size of the attributes for any FD set. With such a large possible input space it was necessary to perform extensive simulations to test the efficiency

of generating random examples as well as assessing the quality of the examples in terms of proximity to an Armstrong relation. We stress that the variation for generating relations is completely up to the designer; for an FD set he may wish to view example relations of any tuple or domain size. Analysing the differences between example relations may highlight the need for perhaps an additional dependency in the specified set, particularly if it is known exactly how close to an Armstrong relation each example is. This section also investigates FD sets whose examples tend to have a low quality.

Number of FD sets	72
For each FD set	1 batch for each domain/tuple combination
Batch Range	1,000 runs in each
Domain Range	$G/2 - 2G$ where $G =\mid$ GEN(F) $\mid$
Tuple Range	$G/2 - 3G$ where $G =\mid$ GEN(F) $\mid$

Table 3.18: Simulation details for evolving relations study

We describe the experiment in detail. In Table 3.18 a run refers to the process of mutating a randomly generated relation until the given FD set is satisfied. Each FD set was evaluated with respect to the average length of the evolution process and the average and maximal quality of the relations produced in batches of 1,000 runs. This was performed for many batches, varying over domain and tuple sizes, both held constant within a batch. As Table 3.18 shows, the batches ranged from having a domain and tuple size of *around* half the cardinality of GEN(F) to a domain and tuple size of double the cardinality of GEN(F). The spread of batches provided all of the useful information; outside this range and smaller relations satisfy the FD set trivially whilst results for larger relations can be gathered from extrapolating within our range. This spread also covered the algorithms behaviour relative to a deterministic generation of an Armstrong relation which always produces a relation with a tuple size of $\mid$ GEN(F) $\mid +1$.

We discuss the absorption rates (number of states to evolution) of two typical sets of FDs, interfering and non-interfering BCNF. Figure 3.8 shows the average number of evolutions to FD satisfaction over 1000 runs for two BCNF FD sets, $F_1 = \{A \rightarrow BC, BC \rightarrow A\}$ (non-interfering) over ABC and $F_2 = \{A \rightarrow BCD, B \rightarrow A, C \rightarrow A\}$ (interfering) over $ABCD$. All of the FD sets used here were comparable in size and complexity given that the larger the FD set the higher, on average, number of states to evolution required. F_2 has an average number of states which increases rapidly as the number of tuples is increased. This is due to the interfering nature of the sets. To describe a possible mutation for set F_2 a uniform random selection may choose to mutate violating attribute B for the FD $A \rightarrow BCD$. This however could be damaging for the FD $B \rightarrow A$ and so our algorithm rejects this mutation. As we can see from Figure 3.8 it is the

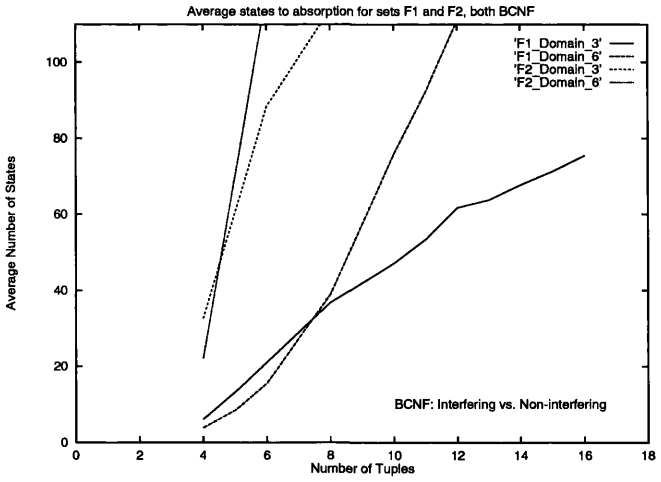

Figure 3.8: Average states to absorption for sets F_1 and F_2, Domain sizes: 3, 6

rejection of such possible mutations that causes the increase in the number of states to absorption. F_1 is non-interfering so that any mutations will only ever be neutral or useful for the other FDs in the set ((Collopy and Levene, 1996), Theorem 3.1) creating fewer rejected mutations and faster absorption rates. The absorption rate of an FD set also rises the more interfering FD pairs there are within the set. Figure 3.8 also highlights another aspect of our evolutionary process, namely that we can generally not determine a difference in the absorption rates between BCNF and non-BCNF FD sets of a comparable size.

Most evolved relations for FD sets achieved a quality of 1, using our similarity measure 2.1, once the tuple size was above | GEN(F) | +1, detailed in (Collopy and Levene, 1996). This is because the probability of evolving an Armstrong relation is evidently lower when the tuple size is below | GEN(F) | +1. With a larger domain the chance of an example relation being Armstrong is significantly lower, especially when the domain and tuple sizes are comparable, often leading to a trivial satisfaction of the FDs as well as FDs outside the specified set. For an empty FD set over R any random relation with schema R satisfies this set; in terms of quality every possible FD would need to be violated for such a relation to be Armstrong. With a null FD set | GEN(F) |=| R | and so anything larger than a binary domain is unlikely to ever be an Armstrong relation given the possible spread of all random relations. A measure of the pathology of an FD set F can be provided by the ratio of the determinations in F to the number of all possible determinations which can occur over the schema R. Given an attribute set and an FD set which explicitly specifies all possible non-trivial FDs which can hold amongst the attributes except for one FD then it is highly likely that many relations will be evolved which in addition violate this FD. Thus within such a batch it is likely that many example evolutions will be Armstrong relations.

The evolutionary procedure is now highlighted with a real world example. A greater understanding of the semantics of an FD set is reached by repeated examinations of different instances of the example relations; this is one motivating factor behind our probabilistic approach.

Example 3.5.1 We use the following non-BCNF FD set F = $\{Name \rightarrow Phone\ FlatNo.$, $FlatNo. \rightarrow Name, Postcode \rightarrow City\}$. We present a deterministic Armstrong relation for this set of dependencies in Table 3.19 together with two different evolved Armstrong relations of varying tuple and domain size. An evolved Armstrong relation is shown in Table 3.20 with the same domain size and tuple number as that used in a deterministic generation. A quick inspection of these two relations shows that the differences in Armstrong relations with the same domain and tuple sizes tend to be superficial, yet the stochastic nature of the generation of relations leads a more well-rounded view of the data. Table 3.21 contains another Armstrong relation, extending the domain size of the deterministic Armstrong relation slightly with an attribute domain size of 8 over 9 tuples. In this instance a larger relation highlights both the satisfied and violated dependencies, those which are not logically implied by F such as $Phone \rightarrow Name$, thoroughly.

Name	Phone	Flat no.	Postcode	City
Dave	1246	19	NW1	London
Dave	1246	19	YO2	York
Dan	3748	7	YO2	York
Dan	3748	7	YO1	York
Charles	3748	11	YO1	York

Table 3.19: Mannila's deterministic AR

We briefly introduce pathological sets, these being the sets for which an Armstrong relation was only rarely, or in some cases never, achieved are discussed more fully in (Collopy and Levene, 1996) We remark that these FD sets contain many FDs which determine few attributes without attributes on their lhs being determined, remembering that our algorithm is not concerned with such relationships.

To conclude, the results have shown that example relations which satisfy sets of FDs can

Name	Phone	Flat no.	Postcode	City
Dave	1246	19	NW1	London
Dave	1246	19	YO2	York
Dan	3748	7	NW1	London
Dan	3748	7	W14	London
Charles	1246	11	YO2	York

Table 3.20: An evolved AR with the same domain size

Name	Phone	Flat no.	Postcode	City
Dave	1246	19	NW1	London
Dave	1246	19	W14	London
Dave	1246	19	YO3	York
Dan	1246	7	BS8	Bristol
Dan	1246	7	BA1	Bath
Dan	1246	7	BA2	Bath
Charles	1246	11	YO2	York
Matt	8881	84	BA8	Bath
Fred	2383	24	YO3	York

Table 3.21: An evolved AR with 9 tuples

be efficiently evolved. The many different relations which can be studied for the same FD set also provide a more well-rounded view of the data in the designer's mind. Batches containing many evolutions can be run and a database designer would then be able to view many relations, including those that are Armstrong if the domain and tuple sizes satisfy the size bounds and an Armstrong relation was actually evolved. If they do not, either domain or tuple size being too low, then the designer can view an approximation to an Armstrong which a batch has provided. In non-pathological cases we conjecture that this will be the best, or close to the best, approximation to Armstrong which exists.

3.6 Discussion

In the chapter we have defined a metric for NDs which we will use in Chapter 4. We reiterate that the goal of data mining with NDs is not to determine a proportion of the database in which a functional relationship is not satisfied but a value for a numerical satisfaction which approximates functional satisfaction. Such mining has been shown to be of use with respect to cardinality constraints in the context of the ER model (Soutou, 1998). Efficient implementations of algorithm 4 warrant further investigation. Work in (Huhtala et al., 1998) which includes computing partitions as a product of previous partitions within the lattice of attribute sets as well a pruning the search space if an FD is found to hold would be directly applicable for ND mining.

For design purposes the evolution of example relations has been shown to be a potentially useful tool. A good database design tool is based on ease of use for the designer and example relations are a step in this direction. To study the applicability of a set of FDs the user can limit the number of tuples in a relation as well as the domain size. The simulations have shown that informative example relations can be evolved by our process. The average number of states to evolution is dependent on both the nature (non-interfering or interfering) and size of the FD set and the size of the relation. Example relations containing attributes independent of each other

are less likely to be evolved into Armstrong relations. For 63 out of the 72 sets of FDs used in simulations an Armstrong relation was evolved for some domain/tuple combination; this is an important side-effect of our approach and may form the basis for further research.

This work will be of use to the database designer as an auxiliary tool to complement the other stages of the design process. From a schema the designer is now able to evolve many varied example relations.

In the domain of Armstrong relations it would be highly interesting to study algorithms for the generation of weak Armstrong Relations, defined in Definition 3.3.2, in the manner of (Fagin, 1982; Beeri et al., 1984) which examine ARs for FDs in the context of improving database design.

CHAPTER 4

The Consistency Problem in Indefinite Relations

In this chapter we demonstrate how NDs may be applied within a heuristic chase based algorithm for approximating solutions to the consistency problem (Vadaparty and Naqvi, 1995). We also demonstrate how resampling may be applied in a dynamic fashion to decide upon suitable sample sizes for the indefinite relation in question.

Our approach to approximating the consistency problem is presented in Section 4.1. In Section 4.2 we motivate the application of indefinite information in relations, referring to the work of (Vadaparty and Naqvi, 1995; Imielinski et al., 1991; Imielinski et al., 1995). Section 4.3 details our approach to the consistency problem, detailing the chase procedure for indefinite information relations, the algorithms applied and the use of two resampling techniques, the bootstrap and the jackknife, for sample size determination. Section 4.4 presents the extensive simulations conducted on randomly generated indefinite relations, both uniform and biased with respect to indefinite cell appearance. We also detail how the simulations were assessed and the results achieved. We conclude in 4.5 with a discussion of further work and introduce how our work might be extended to search for phase transitions using our approximation technique for relations containing indefinite information.

4.1 Our Approach to the Consistency Problem

Given a set of FDs F and an indefinite relation r (a relation with one or more indefinite cells) we tackle the problem of attempting to find a definite relation extracted from r which satisfies F. This is widely known as the *consistency problem*. The consistency problem has been shown to be NP-Complete in general, and of polynomial time complexity in the case where indefinite information is only allowed in attributes which are present in the right hand side of FDs (referred to as a *good* database) or when the FDs have a singleton right hand side and attributes with a domain size of

at most arity two are allowed in the left hand side (Vadaparty and Naqvi, 1995). Henceforth, we refer to definite relations as *possible worlds*. An incomplete relation can be seen as a collection of possible worlds where each world contains a complete instance of the incomplete relation.

Definition 4.1.1 (The consistency problem) Given a set of FDs F and an indefinite relation r the *consistency problem* is the problem of deciding whether there exists a possible world in r which satisfies F, written as $r \models\!\!\approx$ F, see Definition 4.2.1. □

Our approach in attempting to solve the consistency problem is based on using a chase procedure, adapted from the standard chase of Section 2.2.5 for indefinite relations, as a heuristic in conjunction with a hill-climbing technique. We start by applying the chase procedure to remove inconsistent data from the relation which does not satisfy an initial ND set. For an ND $X \rightarrow^k Y$ the chase procedure will collect $k + 1$ tuples and remove values from indefinite cells which would otherwise prevent $X \rightarrow^k Y$ being satisfied and whose removal will not prevent the generation of worlds satisfying ND sets higher in the lattice. If there is *inconsistent* information, implying that $X \rightarrow^{k+1} Y$ is the closest ND to an FD which the relation satisfies, then the chase applied for $X \rightarrow^k Y$ will return an undefined relation containing empty cells, indicating that the result of this is *undefined*.

The algorithm applies this procedure in a hill-climbing manner whereby each iteration generates a possible world satisfying an ND set N from an indefinite relation r. After each iteration the chase is applied to r using the best ND set found so far. This procedure is repeated until the chase returns either an undefined result, stating that it can get no closer to an FD set, or the limit on the number of worlds to generate is exhausted. In contrast to this, a naive procedure was also used which randomly generates n possible worlds and stores the best approximation. For the purposes of this experiment we assume that all possible worlds are equally probable having a uniform distribution. Changing this assumption, for instance by assuming an increased weighting of a particular attribute domain value, leads to different results, briefly discussed in Section 4.4.4.

We wish to know what is a suitable limit on the generation of possible worlds to give the hill-climbing chase procedure. An appropriate size is one which is large enough such that the probability of obtaining the *best possible approximation* to the FD set is high. Though we may expect such a size to be exponential in the cardinality of the relation r, the schema, and the arity of the indefinite cells, it would be foolish to generate a figure without examination or sampling of the data in r. Therefore we use the Bootstrap procedure (Efron and Tibshirani, 1986; Efron and Tibshirani, 1993), a computationally intensive statistical procedure, introduced in Chapter 2. We initially take a sample of n observed possible worlds. Based upon this sample we perform a

number of bootstrap replications. Each bootstrap replication samples from the observed possible worlds with replacement. In this way the Bootstrap is used to provide a guide to the distribution of the possible worlds (Dopazo, 1994). The key assumption we make in this case is that our sample of observed possible worlds is representative of the indefinite relation. We then repeat the Bootstrap with an increasing sample size of observed possible worlds. After each bootstrap iteration we calculate the mean and standard error. The number of observed possible worlds (sample size) is increased until the Bootstrap procedure converges to an approximate fixpoint.

In this sense the convergence of the Bootstrap mean value tells us, with a high probability, that increasing the sample size further will not provide us with any additional information concerning the distribution of data within the indefinite relation. Our results have shown this convergence always occurs with a sample size that is an upper bound on the number actually used by the chase hill-climbing procedure. This is a novel application of sampling within databases, to our knowledge not previously used. To illustrate its usage, a relation with minimal indefinite information and therefore only a few possible worlds will have much less variance amongst the satisfaction of numerical dependency sets. In such a case the bootstrap will reach a fixpoint after few iterations with a final sample size of ρ. The chase and hill-climbing algorithm will then have ρ as a limit on the number of worlds to generate and apply heuristics to. This will be an upper bound based on the minimal variance within the relation.

In order to test the viability of our approach we conducted simulations over 12 sets of FDs, demarcated into *Boyce-Codd Normal Form*(BCNF), see Definition 2.2.22, and non-BCNF, ranging from small to large sizes. Each FD set was evaluated with respect to the average and maximum number of worlds generated and the final value of the *best* ND set. This was performed for around 100 batches, each containing 500 runs, a single run being the process of applying the chase and hill-climbing process until we can climb no further. Each batch was varied over domain, tuple and maximum cell arity size each held constant within a particular batch. The batches were all repeated for the naive procedures. The parameters were varied from a range of trivial satisfaction to trivial inconsistency within a relation. Across batches the weighting of the number of indefinite cells appearing in a relation was also varied from a 25% to a 75% likelihood with this weighting given to cells which are in an attribute present in the left hand side of an FD or not. The simulations emphasised the validity of the chase hill-climbing procedure noting that far fewer worlds are used (before any further chase iterations create an undefined relation) to provide a similar result to the generation of a very large number of possible worlds, the naive approach. Additionally, the run times for the chase and hill-climbing algorithm were much faster than the corresponding naive algorithm. The higher the degree of indefinite cells in a relation tended to provide better results

when using the chase hill-climbing approach. The simulations also showed that our use of the Bootstrap for parameter setting is both valid and useful. Indeed, the application of such statistics seems set to become more commonplace in data mining, as was recently expressed by U. Fayyad in a data mining journal, "I personally look forward to the proper balance that will emerge from the mixing of computational algorithm-oriented approaches characterizing the database ... with the powerful mathematical theories and methods for estimation developed in statistics" (Fayyad, 1998a).

(Imielinski et al., 1991) motivated the use of indefinite information within a relation using a scheduling application and in this context the consistency problem is equivalent to asking whether a particular schedule in invalid. (Vadaparty and Naqvi, 1995) presents a relationship between work on indefinite information and *constraint logic programming*. (Van Hentenryck, 1989) presents a number of logic programs which incorporate domain constraints and use them to aid solving various programs, ranging from simple puzzles to search algorithms. Our methodology could be applied to instances of such puzzles in cases where approximations to a final answer are satisfactory.

4.1.1 Intractability of the consistency problem

It was shown in (Vadaparty and Naqvi, 1995) that the consistency problem is, in general, NP-complete (Garey and Johnson, 1979). It follows that the corresponding consistency problem for NDs is also NP-complete, since FDs are a special case of NDs. In the special case when for all attributes A in the left-hand sides of the FDs in F the A-values of all the tuples in r are definite, then the consistency problem can be solved in polynomial time in the sizes of F and r (Vadaparty and Naqvi, 1995). The NP-complete nature of the consistency problem inherently implies that it would be fruitless to design an algorithm which searches for an exact solution for a relation and a set of FDs. Therefore our algorithm attempts to find an approximation to the best solution that is available.

(Imielinski et al., 1995) shows how query complexity across more than a single relation becomes co-NP-complete when the relations contain indefinite information. Also introduced are *typing functions*, which state whether an attribute can contain indefinite information or not and *degree of co-referencing*, which places restrictions on the type of indefinite information allowed in a relation. This ranges from no repetition of OR-objects, no repetition across columns, and unrestricted repetition. Due to our allowance of indefinite information directly within cells we inherently allow unrestricted repetition. (Imielinski et al., 1995) identifies a complete characterisation of queries for the different classes of database, based on the degree of co-referencing, which are evaluable in polynomial time. This is intended to provide an outline of the allowed use

of queries and indefinacy within a database so that query complexity remains within polynomial time.

The notion of *mutable* and *persistent* object identifiers is also introduced in (Imielinski et al., 1991). A *persistent* object identifier is where the cell containing indefinite information is taken as an object which contains a disjunction (i.e. a name for the object whose value is not yet known) whereas in a *mutable* object identifier the indefinite information is interpreted as disjunction across tuples, which is therefore assumed to have a *place-holder* representation. Mutability is required for structure sharing within indefinite data. We do not consider this in the context of our work. Mutable object identifiers generalise marked nulls.

4.2 Indefinite Information in Relations

In Section 2.2.7 we introduced the background on indefinite information representation in relations principally focusing on the use of OR-objects. We now discuss applications of indefinite information and formalise dependency satisfaction.

4.2.1 Applications

Indefinite information representation in relations has been shown to be a useful facility for incomplete specifications in design and planning applications (Imielinski et al., 1991; Imielinski et al., 1995; Vadaparty and Naqvi, 1995). We define *indefinite cells* as cells containing one or more values which represent a set of possibilities denoting the current limit of knowledge in the database. Any indefinite cell in column A which contains the complete domain allowed for A is equivalent to the traditional NULL value (Lipski, 1979). A definite relation extracted from one containing indefinite information is a relation with the same schema and definite cells, which are invariant throughout, but with each indefinite cell, say I, replaced with a definite cell containing one value from I. Associated with an indefinite relation may be a set of integrity constraints, primarily FDs, the most common integrity constraint in relational databases.

(Imielinski et al., 1991) introduced OR-objects for use within design, planning and scheduling operations, motivated by the lack of functionality in information systems to handle

- coexistence of objects in different stages within the design process
- the ability to evaluate hypothetical queries
- allowing choice within the data model

(Imielinski et al., 1991) presents differences between the *interpretational* and *structural* levels of a schedule. The interpretational level refers to the final designs, possible worlds in our interpretation, of an indefinite relation whilst at the structural level we are concerned with the indefinite

relation itself. In this work we provide a methodology for assessing interpretational information content of indefinite relations. Structural queries discussed in (Imielinski and Vadaparty, 1989; Imielinski et al., 1991) such as "are there two people in a travel relation c with common destinations," may be addressed by including NDs as constraints within the respective data model. (Imielinski et al., 1991) formalises views which allow for querying at either the interpretational or structural level, or a combination of the two. The data complexity of the query language is shown to be co-NP-complete, correlating with the proof given in (Vadaparty and Naqvi, 1995) that the consistency problem is NP-complete due to the fact that a query might ask if all schedules are invalid (a structural query), consistent with all schedules violating an FD set.

We now define FD and ND satisfaction in indefinite relations:

Definition 4.2.1 (Satisfaction of an FD in an Indefinite Relation) Let $s \in \text{POSS}(r)$, be a definite relation over R. An FD $X \rightarrow Y$ is *satisfied* in s, denoted by $s \models X \rightarrow Y$, whenever $\forall t_1, t_2 \in$ s, if $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$. A set of FDs F is *satisfied* in s, denoted by $s \models F$, whenever $\forall\, X \rightarrow Y \in F$, $s \models X \rightarrow Y$.

A set of FDs F is *weakly* satisfied (or simply satisfied whenever no ambiguity arises) in a relation r, denoted by $r \models\!\!\approx F$, whenever $\exists s \in \text{POSS}(r)$ such that $s \models F$. If $r \models\!\!\approx F$ we say that r is *consistent* with respect to F (or simply r is consistent if F is understood from context); otherwise if $r \not\models\!\!\approx F$ then we say that r is *inconsistent* with respect to F (or simply r is inconsistent). □

As for standard relations in Section 2.2.6, we generalise the concept of an FD by an ND.

Definition 4.2.2 (Satisfaction of an ND in an Indefinite Relation) Let $s \in \text{POSS}(r)$, be a definite relation over R. An ND $X \rightarrow^k Y$ is *satisfied* in s, denoted by $s \models X \rightarrow^k Y$, whenever $\forall t_1, t_2, \ldots, t_k, t_{k+1} \in$ s, if $t_1[X] = t_2[X] = \ldots = t_k[X] = t_{k+1}[X]$ then $\exists i, j$ such that $1 \leq i < j \leq k+1$ and $t_i[Y] = t_j[Y]$. A set of NDs N is *satisfied* in s, denoted by $s \models N$, whenever $\forall\, X \rightarrow^k Y \in N$, $s \models X \rightarrow^k Y$.

We define a set of NDs N to be weakly satisfied in a relation r in the same way as for FDs; similarly we define a relation r to be consistent with respect to a set of NDs if $r \models\!\!\approx N$ and otherwise to be inconsistent. □

The use of NDs in possible worlds to approximate FD set satisfaction in an indefinite relation has not previously been considered to our knowledge.

4.3 Algorithm design

We now present an overview of the component parts of our process for approximating solutions to the consistency problem. We begin with a presentation of the chase for NDs in indefinite relations,

followed by an overview of our use of resampling. The principal algorithms are then introduced.

4.3.1 The chase algorithm for indefinite relations

In Algorithm 7, ND_CHASE, we present the chase for NDs in indefinite relations, called within CHECK_CONS. It is a heuristic procedure extended from the standard chase procedure for FDs (Beeri and Vardi, 1984; Mannila and Räihä, 1992a) which, given a set of NDs, attempts to remove extraneous or redundant information that may otherwise prevent the ND set from being satisfied. The forward chase removes extraneous values from indefinite cells in attributes which are in the right hand side of a given FD which is satisfied numerically but not functionally and therefore generalised to an ND. Informally, a partition on attributes in the left hand side of the FD which has at least one more tuple than the branching factor of the ND is selected and indefinite cell values of attributes in the right hand side of the FDs, whose appearance in a possible world would cause the ND to be unsatisfied, are removed. The backward chase removes values from indefinite cells in attributes on the left hand side of the given FDs which would have otherwise prevented satisfaction of the current ND if that value had been selected for inclusion within a possible world.

Algorithm 7 (ND_CHASE(r, N))
1. **begin**
2. Result := r;
3. Tmp := $\emptyset$;
4. **while** Tmp $\neq$ Result **do**
5. Tmp := Result;
6. **if** $\exists$ X $\rightarrow^k$ Y $\in$ N, $\exists t_1, t_2, \ldots, t_k, t_{k+1} \in$ Result such that
   t_1[X], t_2[X] ..., t_k[X], t_{k+1}[X] are definite and t_1[X] = t_2[X] = ... = t_k[X] = t_{k+1}[X]
   but t_1[Y] $\neq$ t_2[Y] $\neq$ ... $\neq$ t_k[Y] $\neq$ t_{k+1}[Y] **then**
7. **for each** $A \in$ Y$-$X and $v \in t_{k+1}$[A] **do**
8. **if** $v \notin \bigcup_{i=1}^{k} t_k$[A] and $\forall i, j \in \{1, 2, \ldots, k\}$ such that $i \neq j, t_i$[A] $\cap t_j$[A] = $\emptyset$ **then**
9. t_{k+1}[A] := t_{k+1}[A] $-\{v\}$;
10. **end if**
11. **end for**
12. **end if**
13. **if** $\exists$ X $\rightarrow^k$ Y $\in$ N, $\exists t_1, t_2, \ldots, t_k, t_{k+1} \in$ Result such that
    t_1[XY], t_2[XY], ..., t_k[XY], t_{k+1}[Y$-$X] are definite and t_1[X] = t_2[X] = ... = t_k[X]
    and t_1[Y$-$X] $\neq$ t_2[Y$-$X] $\neq$ ... $\neq$ t_k[Y$-$X] $\neq$ t_{k+1}[Y$-$X] and
    sound(t_1[X],t_2[X],...,t_k[X],t_{k+1}[X]) **then**
14. $\forall$ A $\in$ X$-$Y, t_{k+1}[A] := t_{k+1}[A] $-$ t_i[A], for some $i \in \{1, 2, \ldots, k\}$;
15. **end if**
16. **end while**
17. **return** Result;
18. **end.**

Figure 4.1: Chase for Numerical Dependencies with forwards and backwards tests

We include the *sound* check for the attributes values on the left hand side of an ND

$X \rightarrow^k Y$ to ensure that values are not removed unnecessarily from cells which might otherwise form an attribute value combination over X which is not present in any of the tuples in $\{ t_1[X], t_2[X], \ldots, t_k[X] \}$ and therefore not redundant. If this step were not performed we might create an undefined relation unnecessarily. To illustrate this we present Table 4.1 showing an indefinite relation over ABC with an FD AB $\rightarrow$ C. If we apply the backwards chase to this relation without the *sound* check then, due to each tuple disagreeing on C with the indefinite tuple, this would result in an undefined relation. Table 4.2 depicts the satisfying instance.

A	B	C
0	0	0
0	1	0
1	0	0
{0,1}	{0,1}	1

Table 4.1: Indefinite relation r, FD AB $\rightarrow$ C

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Table 4.2: A satisfying world for AB $\rightarrow$ C

Definition 4.3.1 (Good Classes of Indefinite relations, (Vadaparty and Naqvi, 1995)) Given a relation r and a set of FDs F, r is *good* for F if it contains no indefinite cells in r on any of the attributes which are also on the left hand side of any FD in F. □

An $O(n^2)$ algorithm, DELETE_REDUNDANT, is presented in (Vadaparty and Naqvi, 1995) that takes a relation good for F, and pre-processes it such that the resulting relation has only conforming possible worlds. It is said to *fully incorporate* any set of FDs in a database D if the database is good for the set of FDs; it is equivalent to ND_CHASE with only a forward chase component. The algorithm maintains a cumulative domain of all the OR-objects; the relation r is partitioned for agreement on the attributes in the body of F and each partition has an intersected domain of all OR-objects within the attributes on the right hand side of F. It is shown to be sound given that (Vadaparty and Naqvi, 1995) accept, without comment, that a relation that is not satisfying will have a null-intersection on its OR-objects and will not therefore have any possible worlds.

4.3.2 Resampling for the Consistency Problem

Given that the number of possible worlds of an indefinite relation increases exponentially in the size of the relation it is impossible to examine all possible worlds for the best solution. The complete population distribution is unknown; otherwise we would know exactly how many definite relations to generate to have a specific probability of finding the closest ND set to the given FD set. This suggests applying a bootstrap procedure to a sample of definite instances to approximate the population distribution based on the sample distribution. Essentially we take a sample of n

possible worlds from an indefinite relation. Then we use the sample to construct pseudosamples of size n, obtained by selecting randomly from the sample with replacement for each pseudosample. We then increase the sample size by a small amount δ and repeat the bootstrap procedure with sample size $n + \delta$ until a fixpoint is reached and subsequent increases do not affect the variance any further. Informally, we use this incremental bootstrap procedure to tell us how many worlds we need to consider so that we have a high confidence that generating additional worlds will not improve our solution.

Independently of this work we refer the reader to (John and Langley, 1996) which defines dynamic sampling as, "the use of knowledge about the behaviour of the mining algorithm in order to choose a sample size." Within the context of this work, we prefer to define dynamic sampling as the use of knowledge about the data to choose a sample size. Our incorporation of resampling does exactly this. (John and Langley, 1996) note that in data mining and decision support it is important that the sample size is well chosen. Indeed, a poorly chosen sample size which may not accurately capture the information content of a database to within the correct degree of error, may result in a loss of much money. (John and Langley, 1996) therefore introduces the PCE (Probably Close Enough) inequality, a derivation of the PAC-learning criterion (Valiant, 1984; Anthony and Biggs, 1992), which states that

$$Pr(acc(D) - acc(s) > \epsilon) \leq \delta$$

where acc measures the accuracy of the mining algorithm, D refers to the database, s the sample and ϵ and δ are error and confidence limits, respectively. (John and Langley, 1996) assumes that whenever $acc(s_{i+1}) \leq acc(s_i)$ then further increases in sample size will result in a loss of accuracy and that n_i is a suitable sample size, given that the derivative of accuracy with respect to sample size has become non-positive. (John and Langley, 1996) state that it is necessary to estimate $acc(s_i)$ and uses leave-one-out cross-validation to achieve this; we choose instead to use bootstrap resampling for our estimation of a sample size. Similarly, our employment of dynamic resampling assumes that when we reach an approximate fixpoint no further increases in sample size will improve the knowledge of the indefinite relation.

4.3.3 The Bootstrap Process within Indefinite Relations

The bootstrap is a data driven simulation method for statistical inference. It is a computationally intensive procedure that has been shown to ably provide confidence limits which would not have been capable of being similarly generated more than 30 years ago. In our experience, statistical methods have not previously been applied in the solution of database problems such as the consistency problem. We now formalise the use of the bootstrap in indefinite relations.

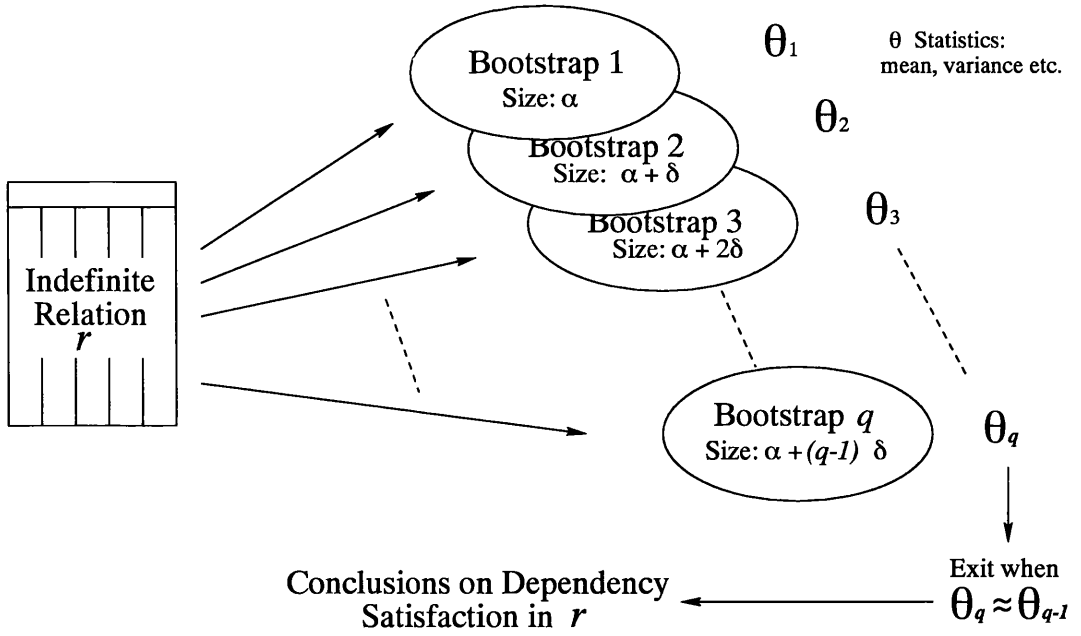

Figure 4.2: The Bootstrap procedure applied to increasing sample sizes for an indefinite relation

Definition 4.3.2 (The Bootstrap Sample) Given an indefinite relation r over schema R where $\mid R \mid = m$ and $\mid r \mid = v$ and the maximum arity in an indefinite cell is q, then r can have *at most* q^{mv} possible worlds. From r we uniformly randomly extract n possible worlds. Each of these worlds will satisfy a set of NDs (which may be FDs). These n possible worlds are referred to as the original sample or *observed possible worlds* and are written as $\tilde{p} = (r_1, r_2, \ldots, r_n)$. A bootstrap sample is $\tilde{p}^\star = (r_1^\star, r_2^\star, \ldots, r_n^\star)$ where for all $i = 1, 2 \ldots, n$ each $r_i^\star$ is randomly selected with replacement from the n observed possible worlds, $\tilde{p}$. □

The probability of an observed possible world *not* being present in a bootstrap sample of size n is $(1 - \frac{1}{n})^n$ assuming each world in the sample has a $\frac{1}{n}$ chance of being selected. Note that every observed possible world has an equal likelihood of being selected for each point in the Bootstrap sample. The ND set is restricted to expressing approximations to a given FD set in this work.

We denote each of the l NDs which may hold in r by $X_i \rightarrow^{k_i} Y_i$ where $1 \leq i \leq l$. We denote the branching factor k which holds for ND $X_i \rightarrow^k Y_i$ in r as $br_{X_iY_i}(r)$.

Definition 4.3.3 (The Bootstrap Sample Mean) Given a bootstrap sample $\tilde{p}^\star = (r_1^\star, r_2^\star, \ldots, r_n^\star)$, we calculate the mean $\bar{s}(\cdot)$, or any other statistic of interest, in exactly the same way as we would have for the original sample of ND sets, each containing m NDs,

$$\bar{s}(\tilde{p}^\star) = \{X_j \rightarrow^K Y_j \mid 1 \leq j \leq m\} \quad \text{where} \quad K = \Sigma_{i=1}^{n} br_{X_jY_j}(r_i^\star)/n$$

When we refer to the sample mean of a set of possible worlds we are implying the sample mean of the sets of NDs of the possible worlds. □

Definition 4.3.4 (The Bootstrap Replication Size (BRS)) The Bootstrap Replication Size, B, is the number of times a Bootstrap sample of size n is created from the observed possible worlds (the original sample) and evaluated on a parameter of interest. We denote the B bootstrap samples by $\tilde{p}_b^\star = (\tilde{p}_1^\star, \tilde{p}_2^\star, \ldots, \tilde{p}_B^\star)$. □

Definition 4.3.5 (The Bootstrap Mean of all Values) Given a set of B bootstrap samples $\tilde{p}_b^\star$, we calculate the mean $\bar{s}(\cdot)$, or any other statistic of interest, in exactly the same way as we would have for the original sample, $\bar{s}(\tilde{p}_b^\star) = \Sigma_{i=1}^{B} \bar{s}(\tilde{p}_i^\star)/B$. □

(Efron and Tibshirani, 1993) tackles how large the BRS should be. Given a BRS B, (Efron and Tibshirani, 1993) refers to the *ideal bootstrap estimate* which takes B equal to infinity. As B increases the empirical standard error tends towards the standard error of the original sample. Therefore the population distribution of the resamples are based on the population distribution found in the sample; this emphasises the *non-parametric* nature of the bootstrap. (Efron and Tibshirani, 1993) show that the amount of computation time required for increasing BRS sizes grows linearly. We show that this is also the case for increasing the BRS for indefinite relations in Figure 4.3 where new FDs, determining a new attribute, are added. Figure 4.3 represents a near *worst case scenario* where each FD added to the set determines a single attribute where all of its cells are indefinite, of arity 3, and intersect on only one value. The number of tuples in the relation and both the degree of indefinite cells and arity of these cells affects the gradient of these lines.

Definition 4.3.6 (The Bootstrap Standard Error for Indefinite Relations) The sample standard error in the values for B bootstrapped values is:

$$\hat{se}_B(\tilde{p}_b^\star) = \{\frac{1}{B}\Sigma_{i=1}^{B}(\bar{s}(\tilde{p}_i^\star) - \bar{s}(\tilde{p}_b^\star))\}^{1/2} \qquad \square$$

We now describe the methods of our Bootstrap application, detailed in Algorithm 10. The process is outlined in Figure 4.2. We start with a small initial sample size α and a Bootstrap Replication Size B. Having created B bootstrap samples we will have a bootstrap mean of all values in the form of an ND set. For this value we can use the bootstrap to calculate the standard deviation (and other statistics if desired). From this we can empirically infer the width of the interval in which a certain percentage of the relations occur. We continue to increase α by a fixed amount, δ, until we reach a point where the mean value of the NDs in the ND set converge. In Figure 4.2 this

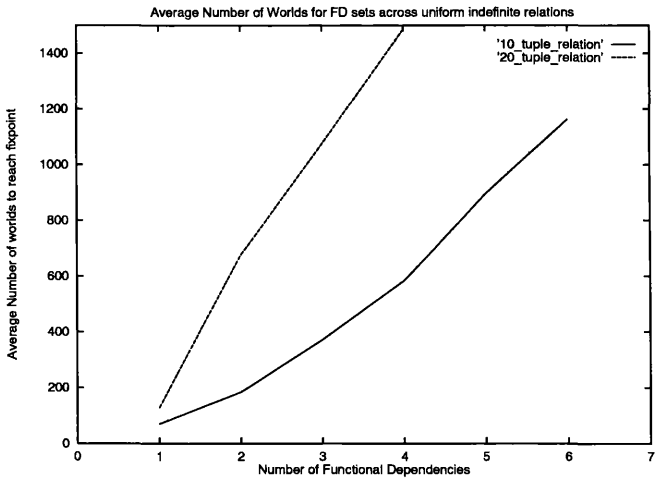

Figure 4.3: Average number of worlds to reach an approximate fixpoint of the mean bootstrapped ND values in 10 and 20 tuple random relations

occurs after q bootstrap applications on different sample sizes. This provides a parameter whereupon any samples larger than $\alpha + (q-1)\delta$ is unlikely to have any significant change in variance. It is unlikely, even for an ND set with just one dependency, for this to be reached randomly and running our simulations in batches of 500 implied that any erroneous fixpoint values as outliers would have a negligible impact on the final results.

We also examined the variance of the observed possible worlds, for a range of original sample sizes, as the bootstrap replication size was scaled from 20 up to 50,000 to decide on a suitable BRS, detailed more fully in Appendix B. As this was increased we noted that above 1000 there was negligible change in the variance. Extensive research on the bootstrap has shown that even for BRS of 25 useful inferences can be made, and that there is seldom a significant change once a BRS is over 200 (Efron and Tibshirani, 1986). For the purposes of our experiment setting B at 100 gave a suitable value which allowed sufficient repetitions of the complete bootstrapping process in a reasonable time, knowing that there would be only a minimal change in the variance for any increase in B. Additionally we experimented with using the original indefinite relation for each resampling iteration from which n possible worlds are sampled each time. The variance is much higher in this case as we have all possible worlds to select from for each sample of size n. In terms of reaching a fixpoint this takes much longer and was not used in the final simulations. It could be of use in situations where the bootstrap sample may be unrepresentative of the population.

4.3.4 Resampling Algorithms

We now formalise the bootstrap and jackknife algorithms used for resampling and the dynamic algorithm within which they were used for indefinite relations. Algorithm 8, BOOTSTRAP(N_{bag},s,B), describes the standard bootstrap procedure which returns the mean value of ND sets for a BRS value B of Bootstrap replications, sample size s and a sample population N_{bag} of ND sets. The value α provided by the bootstrap is used both in ND_GEN and CHASE_GEN. Algorithm 9, JACKKNIFE(N_{bag}), creates n resamples from N_{bag} where $\mid N_{bag} \mid = n$ and each resample is of size $n-1$. This was initially developed before the bootstrap, to which it has been shown to be an approximation (Efron and Tibshirani, 1986). It is of most use when a sample is likely to contain significant outliers. We discuss its use in Section 4.4.6.

Algorithm 8 (BOOTSTRAP(*nd_bag*, *n*, *B*))
1.**begin**
2. ND_mean := $\emptyset$;
3. **for** 1 to B **do**
4. ND_samp := Uniform Randomly select n ND sets from *nd_bag* with replacement;
5. Insert the mean of ND_samp into ND_mean;
6. **end for**
7. **return** the mean of ND_mean;
8.**end.**

Figure 4.4: The Bootstrap procedure for indefinite relations

Algorithm 9 (JACKKNIFE(*nd_bag*))
1.**begin**
2. ND_m := $\emptyset$;
3. n := $\mid$ *nd_bag* $\mid$;
4. **for** j := 1 to n **do**
5. ND_samp := *nd_bag* - nd_j;
6. Insert the mean of ND_samp into ND_m;
7. **end for**
8. **return** the mean of ND_m;
9.**end.**

Figure 4.5: The Jackknife procedure for indefinite relations

Algorithm 10, WORLD_LIMIT(r, F, B), implements our novel use of the Bootstrap procedure. The initial sample size we incorporated in our simulations was 10 possible worlds, sufficiently small for application to all indefinite relations. This could possibly be extended to using the degree of indefinite cells and the domain sizes to calculate a suitable initial sample size. We motivate our procedure on the assumption that different sample sizes are required according to the

variance within an indefinite relation in the different ND sets which may be satisfied in possible worlds. The number of dependencies in the given FD set also influences the results obtained from our use of the bootstrap. In Section 4.4.2 we show that this application of the bootstrap returns an upper bound on the number of worlds required for a good answer. Unlike many statistical operations, the BOOTSTRAP algorithm operates in exactly the same manner as a standard bootstrap procedure despite the fact that we potentially have all possible worlds within the indefinite relation, unlike many statistical applications from which inferences are made on incomplete populations. Based on this we conducted experiments whereby the bootstrap resamples were obtained not from the original sample but from the indefinite relation. As stated, the variance of resampling from the relation was much higher than resampling from the sample and in such cases the upper bound was much higher. Therefore we have found it to be suitable to use just one original sample from the indefinite relation within each iteration of WORLD_LIMIT. This is elaborated upon in Appendix B.

Algorithm 10 (WORLD_LIMIT (r, F, B))
1. **begin**
2. n := initial(r); % sample size, based on r
3. ND_bag := n ND sets from n possible worlds, each approximating F
4. $\hat{N}_0 := \emptyset$;
5. $\hat{N}_1$:= BOOTSTRAP(ND_bag, n, B);
6. $j := 1$;
7. **while** $\hat{N}_j, \hat{N}_{j-1}$ are not approx. fixpoint **do**
8. ND_bag := n ND sets from n possible worlds;
9. $\hat{N}_j$:= BOOTSTRAP(ND_bag, n, B);
10. $n := n + \delta$; % Increase the sample size by δ
11. $j := j + 1$;
12. **end while**
13. **return** n;
14. **end.**

Figure 4.6: The WORLD_LIMIT algorithm for incremental bootstrap sampling in indefinite relations

4.3.5 Finding an approximate solution to the consistency problem

We focus on finding an approximation N of an FD set F for an indefinite relation r such that $r \models\!\!\approx$ N using Algorithm 11, denoted by CHECK_CONS(r, F, B), where B is the bootstrap replication size (BRS). Recall that we have assumed that $| r | = m + 1$, where $m \geq 1$; if $| r | < 2$, then $r \models\!\!\approx$ F trivially holds; we refer the reader to Definition 4.2.2.

We now briefly describe the *naive* version of CHECK_CONS applied to an indefinite relation and an FD set F. This simply generates a fixed number of possible worlds, each satisfying an ND

set approximation of F, and returns the ND set with the closest proximity to that of F. We apply ND_CHASE before the generation to remove redundancy, even from the naive selection. It is quite feasible to consider the use of the bootstrap in conjunction with a naive approach though this would always generate exactly α definite worlds, exactly the figure returned by WORLD_LIMIT. Use of resampling in a naive procedure is unwarranted given that such computation time would be better spent generating new possible worlds and not resampling, which may generate possible worlds that satisfy close approximations but are not then detected again by a naive procedure.

Algorithm 11 (CHECK_CONS(r, F, B))
1. **begin**
2. BOT := the bottom element of $\mathcal{L}_m$(F);
3. s := CHASE(r, BOT);
4. **if** s is undefined **then**
5. **return** $\{X \rightarrow^{m+1} Y \mid X \rightarrow Y \in F\}$;
6. **end if**;
7. APPROX := BOT;
8. α := WORLD_LIMIT(r, F, B);
9. S := $\emptyset$;
10. **while** APPROX $\neq$ F and $|S| \leq \alpha$ **do**
11. **repeat**
12. gen_rel := ND_GEN(s, APPROX, α);
13. **if** gen_rel is not definite **then**
14. **return** APPROX;
15. **end if**
16. **until** gen_rel $\notin$ S;
17. S := S $\cup$ {gen_rel};
18. **while** $\exists$ G such that APPROX $\prec$ G and gen_rel $\models$ G **do**
19. APPROX := G; % hill climbing step
20. **end while**
21. **if** $\exists$ G such that APPROX $\prec$ G and CHASE(s, G) is defined **then**
22. s := CHASE(s, G);
23. **else**
24. **return** APPROX;
25. **end if**
26. **end while**
27. **return** APPROX;
28. **end.**

Figure 4.7: The CHECK_CONS algorithm for approximating solutions to the consistency problem

4.3.6 The Chase and Hill-Climbing Algorithm

Algorithm 11, CHECK_CONS(r, F, B), initially removes extraneous information from r via ND_CHASE. Algorithm 10 generates a suitable sample size α using the bootstrap dynamically. Then, until either a possible world satisfying F is found, or α is reached the following occurs:

ND_GEN is called to generate a definite world gen_rel; gen_rel is used in a hill-climbing fashion to obtain the best ND set $APPROX$ which it satisfies and the chase is reapplied to the indefinite relation using an ND set G which covers $APPROX$. If the chase is undefined for all sets covering $APPROX$ then this set is returned as the best approximation given that the indefinite information in the relation does not satisfy any *higher* ND set.

ND_GEN(r, N, α), invoked from CHECK_CONS, attempts to generate a possible world using uniform random selection in conjunction with chase procedures of CHASE_GEN. Using such random selection in this manner allows for a value to be removed from an indefinite cell which may then aid subsequent redundant values to be removed by CHASE_GEN. Algorithm 13, CHASE_GEN(r, N, α), applies a chase based heuristic to unify two tuples which have a non-null intersection on a determined attribute A, randomly selecting one value from their intersection. If we reach a point where $k + 1$ tuples have a null intersection then we have removed too much information for X $\rightarrow^k$ Y to ever hold and we return to the original indefinite relation. We use the WORLD_LIMIT sample size on the assumption that if we have to repeat this procedure α times we assume that X $\rightarrow^k$ Y will never hold based on the indefinite data. In ND_GEN we also assume that α is large enough such that a definite relation is returned if there exists a possible world in r which satisfies the given ND_set. The empirical results of our simulations show that using these heuristic algorithms generate, on average, equivalent if not better, approximations in a much faster time than naive selection.

4.4 Simulations and Results

We now discuss the simulations conducted to examine the viability of our methods for attempting to find a consistent possible world within indefinite relations, detailed in Appendix B. We concentrated on a few FD sets demarcated by the number of dependencies in the set and whether they were BCNF or non-BCNF. In Table 4.3 we present an overview of the parameter ranges for the simulations conducted. Batches containing 500 runs were executed so that we could find reliable averages for both naive and chase and hill-climbing algorithms. The range of possible inputs for an indefinite relation is very large. We limited the size of our relations to 50 tuples, and carried out the simulations with batches having a maximum indefinite cell arity of up to six elements and a domain size for each attribute of up to 10 elements, noting that the domain size must be higher than the maximum indefinite cell arity. The weighting of the likelihood of the presence of an indefinite cell was also varied for selected batches. In a *standard* batch we randomly generate a relation wherein each cell has a 50% chance of being indefinite. If it is selected to be indefinite then its arity, up to the maximum for the batch, is then randomly selected. The weighting was

Algorithm 12 (ND_GEN(s, N, β))
1. **begin**
2. gen_rel := CHASE_GEN(s, N, β);
3. **if** gen_rel is undefined **then**
4. **return** gen_rel;
5. **end if**
6. Fail := 0;
7. **while** gen_rel is not definite and Fail $\leq \beta$ **do**
8. Tmp := gen_rel;
9. **if** $\exists$ A $\in$ R and $u_i \in s$ such that $|u_i[A]| > 1$ **then**
10. $u_i[A] := \{v\}$, where $v \in u_i[A]$ is randomly chosen;
11. **end if**
12. gen_rel := CHASE_GEN(gen_rel, N, β);
13. **if** gen_rel is undefined **then**
14. gen_rel := Tmp;
15. Fail := Fail + 1;
16. **end if**
17. **end while**
18. **return** gen_rel;
19. **end.**

Figure 4.8: The ND_GEN algorithm for generating a possible world

varied in batches for suitable FD sets from a 25% to 75% likelihood of being indefinite on the attributes according to whether they are in the left or right hand side of an FD. The value of our approximate fixpoint within WORLD_LIMIT was set to 2 decimal places; this may be set empirically.

Number of FD sets	12 (6 BCNF / 6 non-BCNF)
Program Versions	Naive/Chase and hill-climbing
Single FD simulations	1 batch for each domain/tuple/cell-arity combination
Batch Range	500 runs in each
Domain Range	1 - 10
Tuple Range	5 - 50
Cell-Arity Range	2 - 6 (domain size $\geq$ cell-arity)

Table 4.3: Simulation details for the consistency problem

4.4.1 Use of our metric

The metric for sets of NDs, defined in Section 3.1, was used throughout the simulations to asses the proximity of an ND set to an FD set which it approximates, known to be the top of the lattice, N_{T}. Within a batch we formed the mean value of the metric, as shown in the graphs.

Algorithm 13 (CHASE_GEN(s, N, γ))
1. **begin**
2. Result := s;
3. Tmp := $\emptyset$;
4. Fail := 0;
5. **while** Tmp $\neq$ Result **do**
6. Tmp := Result;
7. **if** $\exists$ X $\rightarrow^k$ Y $\in$ N, A $\in$ Y$-$X and $u_1, u_2 \ldots, u_k, u_{k+1} \in$ Result such that
   u_1[X],u_2[X], . . ., u_k[X], u_{k+1}[X] are definite and
   u_1[X] = u_2[X] = . . . u_k[X] = u_{k+1}[X] **then**
8. **if** $\exists i, j \in \{1, 2, \ldots, k, k+1\}$ such that u_i[A] $\cap$ u_j[A] $\neq \emptyset$ **then**
9. u_i[A], u_j[A] := $\{v\}$, where i, j and $v \in u_i$[A] $\cap$ u_j[A] is randomly chosen;
10. **else**
11. **if** Fail $\leq \gamma$ **then**
12. Result := s;
13. Tmp := $\emptyset$;
14. Fail := Fail + 1;
15. **else**
16. $\forall i \in \{1, 2, \ldots, k, k+1\}$, u_i[A] = $\emptyset$;
17. **return** Result;
18. **end if**
19. **end if**
20. **end if**
21. **end while**
22. **return** Result;
23. **end.**

Figure 4.9: The CHASE_GEN algorithm for applying a chase method randomly

4.4.2 Results

We now present some results based upon our simulations. We show in Figure 4.10 results depicting the closest proximity within a batch for both the naive and the chase and hill-climbing approaches for the FD set $F_1 = \{A \rightarrow B, A \rightarrow C, A \rightarrow D\}$ having a domain of 7 and containing indefinite cells of a maximum arity 6, rather large for real purposes. This figure represents the result for just one run in a batch. We can see for this run that the best results for the use of our chase methodology are the same as for the naive procedure when the relation contains both 15 and 35 tuples. The similarity between the use of the chase and naive methods, leading to an uneven graph is expected given that the chase is only a heuristic to aid our hill-climbing procedure. We note, however, that across a batch or 500 runs that the mean results of the chase procedure are slightly better, shown in Figures A.9 and A.10 in Appendix A. Additionally, we discuss the superior efficiency of the chase and hill-climbing algorithm in section 4.4.3.

The limit, α, on the iteration size, as supplied by the Bootstrap was equalled in less than

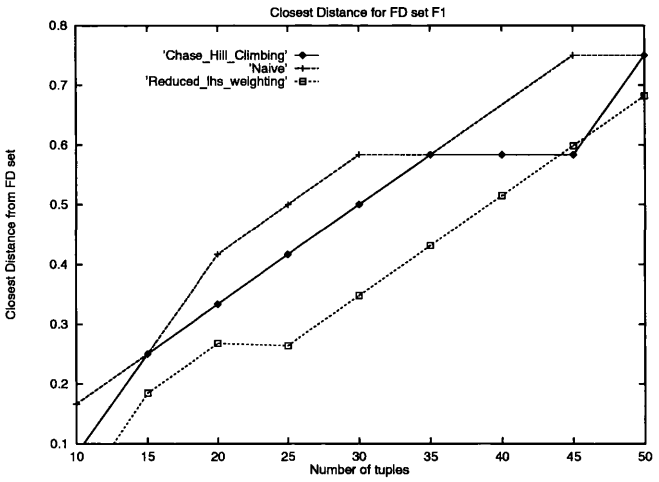

Figure 4.10: Closest Proximity for FD set F_1 across a number of different weighted relations

1% of the simulations, showing this to be a suitable upper bound. Indefinite relations with a large number of indefinite cells in relation to the domain size are apt to satisfy very many ND sets which are equivalent, leading to exhausting of the limit α provided by WORLD_LIMIT.

4.4.3 Analysis of the Chase results

Simulations showed that the chase procedure outperformed the naive approach, on average, by an increasing margin as the number of tuples within a randomly generated relation increased. This margin became slightly larger at higher domain sizes within relations. Obviously, as the tuple and domain size are increased, the chase procedure becomes more effective due to the increased probability of there being more redundant values to remove. We can see in Figure 4.10 that if there is a bias towards having more indefinite cells in attributes which are present in the right hand side of FDs (or fewer indefinite cells in the left hand side, by symmetry) then the closest proximity for both the naive and chase approaches are better than an even weighting of indefinacy in left and right hand sides. The chase procedure is also more effective at an earlier stage, evidenced by the *Reduced_lhs_weighting* line in Figure 4.10. An increased number of indefinite cells in attributes on the right hand side of FDs implies that there may be more values which may lead to unnecessarily low ND satisfaction (i.e. each ND will have a larger branching factor) which can now be removed by the chase heuristic. Our simulations show the increased efficacy of the chase in such cases. A larger indefinite cell arity also implies that the chase will have more values to remove and therefore perform even better. In the case of reducing the weighting of indefinite cells of the left hand side of FDs, a naive approach performs much worse than in an evenly weighted relation due to there being fewer indefinite cells from which it can select different values, thereby preventing much variation of the partitioning on the NDs in a relation which might otherwise oc-

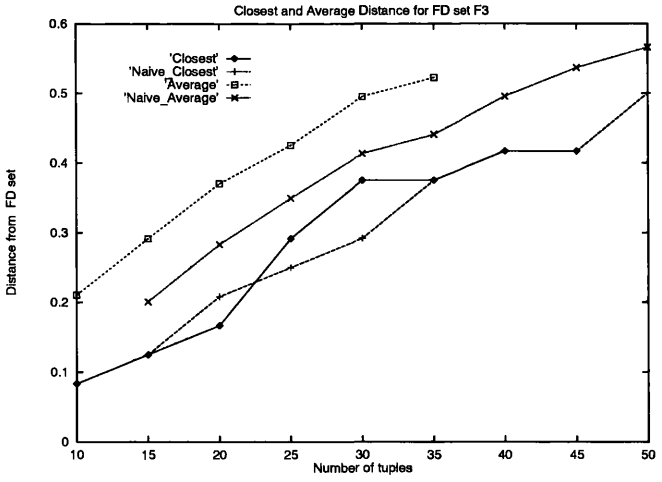

Figure 4.11: Closest and Average Proximity for FD set F_3

cur. Fewer possible worlds are present in such biased relations though there are still too many to consider applying a naive approach alone. Conversely, a reduced weighting of indefinite cells in the right hand side of FDs also produces better results than an even weighting of indefinite cells. This was simply due to the creation of more partitions. As expected, and in contrast to our work in Chapter 3 on evolving relations, we did not find a significant difference between using BCNF and non-BCNF FD sets in either case. The fact that an attribute is, or is not, part of a superkey did not affect the overall proximity to an FD holding within a randomly generated indefinite relation. This may not necessarily be the case for real-world data, where the presence of a key may suggest a closer proximity to dependency satisfaction.

Figure 4.11 shows results for $F_3 = F_1 \cup \{BD \rightarrow A\}$. For this relation the chase procedure performs poorly, on average, with respect to the naive technique. We believe this is due to the interference of the attributes within the FD set having attributes determining and being determined by each other which reduces the application of the chase heuristic. We note however that the best results within a batch obtained by both the naive and the chase and hill climbing are increasingly similar as relation size increases. We reiterate that the chase and hill climbing approach requires far fewer worlds.

In Figure 4.12 we see that, for both FD set F_1 and $F_2 = \{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$, as the number of tuples increases there is a slight peak, after which further increases in the number of tuples results in a fall in the average number of worlds required. This is based on every relation within a batch having a fixed domain size d and an indefinite cell maximum arity, reaching a point where it is likely that any further increases in the tuple size will lead to the satisfaction of the numerical dependency set with each ND left hand side determining d branches and so fewer worlds

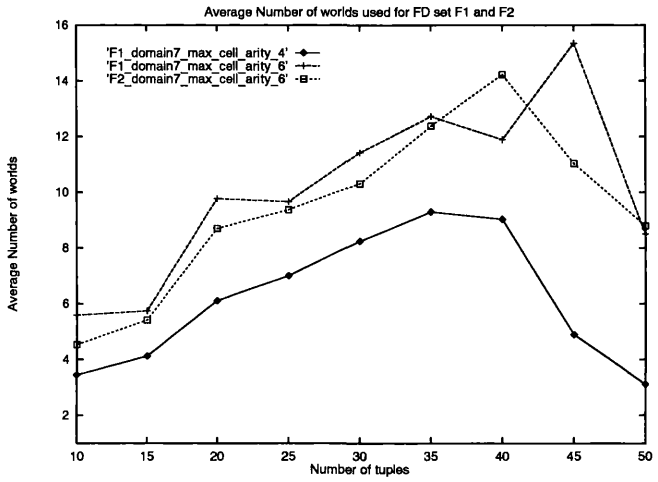

Figure 4.12: Average Number of Worlds required by the chase and hill-climbing approach

are required before any attempts to apply the chase returns an undefined relation implying that nothing better can be found. The peaks in Figure 4.12 were reflected in the values of α returned by our bootstrap technique, corroborated by Figure 4.15. In our application of the bootstrap, as the relation size of a random relation is increased and the domain size is held constant, the sampling will also reach a point where the variance in the samples amongst the randomly generated possible worlds is reduced due to most possible worlds satisfying the NDs each with a branching factor close to their domain size. This is likely to also be the case for very large real world indefinite relations.

Given that all of our test data was uniformly randomly generated, with a bias to or against indefinacy in specified attributes if desired, we remark that the results echoed the general behaviour presented here. The average number of worlds required in Figure 4.12 emphasises the efficiency of the chase and hill-climbing over naive procedures. The lack of indefinite information within databases in daily use prevent grander conclusions on the efficacy of the chase, where it may have wider use, particularly with respect to larger relations. For example, if a database required only indefinite information in an attribute on the right hand side of a given FD and the domain size was small with respect to the database size then the chase would be an effective heuristic.

4.4.4 Changing Bias of indefinite information

We now briefly discuss differences within resampling for relations with different bias of indefinite cells in the relation, following on from the discussion of bias in the previous section. Experiments exemplified the importance of how the definite cells satisfy ND sets; if the definite cells in an indefinite relation satisfy ND sets which are closer to FD sets then we found a larger overall variance in our possible worlds. This is explained due to the definite cells themselves being further from or

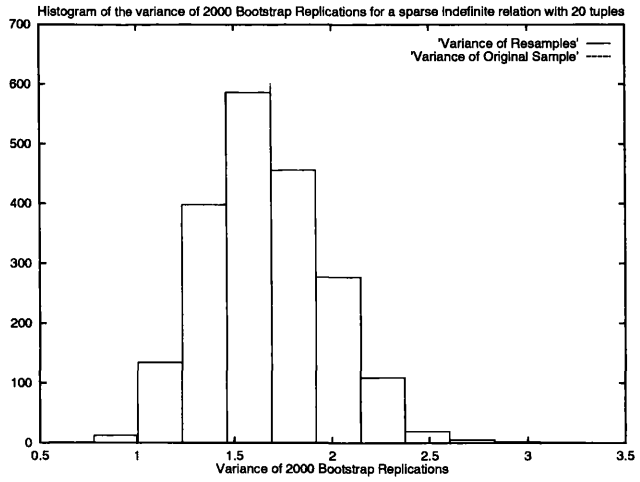

Figure 4.13: Histogram of 2000 bootstrap replications of sample size 25 for a 20 tuple relation and 10 FDs, with lhs attributes definite and rhs attributes sparse (in FD set) in indefinite cells

closer to FD set satisfaction which implies, respectively, a smaller or larger change in proximity within the lattice of NDs. This was more significant for bigger relations, with a larger domain size and hence a larger lattice.

In Figure 4.13 we provide a histogram of 2000 bootstrap replications for a relation with 20 tuples, 10 attributes and 10 FDs, each with the same singleton left hand side A and a different singleton right hand side $B_1, \ldots, B_9$. The relation has a single partition on A and for each B_i 50% of the tuples are indefinite. We emphasise that more indefinite cells on the left hand side of the FDs decrease the variance due to each left hand side indefinite value creating new partitions on attribute values whilst more indefinite cells on the right hand side of FDs increase the variance. Obviously, the arity of indefinite cells and intersections of values temper the change in variance.

4.4.5 Finding a suitable sample size

Our use of the bootstrap procedure was found to provide a suitable upper bound on the number of worlds required by our algorithms. We have explained how the dynamic resampling relies on the variance of ND set satisfaction amongst possible worlds in the sample to infer when a larger sample is not required. The fact that this provided an upper bound for our algorithms justifies its use. As the sample size grows, highlighted in Figure 4.14, there is a reduction in variance between successive iterations. The non-parametric nature of the resampling is shown to be useful in that the empirical confidence limits for the bootstrap process are shown to converge for the distance measure of an ND set. Determining confidence intervals with the bootstrap is discussed in Appendix B.

A problem with the bootstrap is also discussed in (Diaconis and Efron, 1983). It seems to

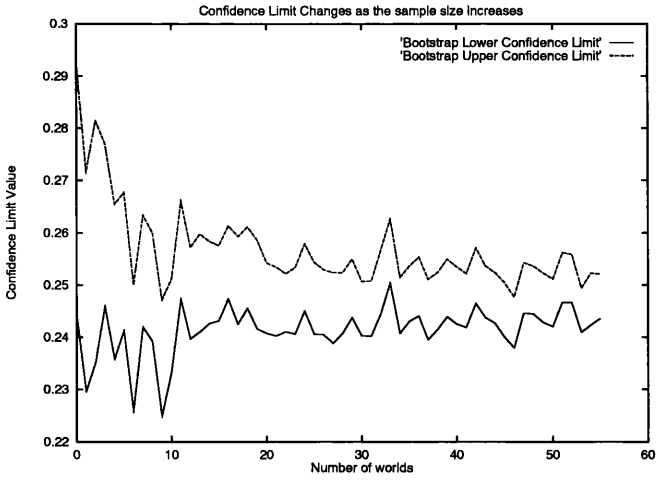

Figure 4.14: Empirical bootstrap percentile confidence limits shown to converge for the distance measure of ND sets

occur when there is very little variation in the range of values in the sample data. For instance, it may be the case that we have an indefinite relation which for a given FD set is such that nearly all possible worlds satisfy this FD set. Bootstrap sampling on this data set would be judged to have a very high accuracy, based on the empirical lack of variation found in the samples. (Diaconis and Efron, 1983) says this would be incorrect. The bootstrap will always perform badly when there is an indefinite relation with only one or few worlds which we consider to be *good* in the context of approximating FD sets. Indeed, we do not say that the bootstrap performs badly, it merely creates an average of the branching values based upon sampling with replacement from the original subsample. This will always be a good reflection of the average branching values in the ND set unless the original n samples are not a good reflection of the true values in the relation.

Note that the bootstrap procedure can not be used to provide any indication as to whether there is, or is not, present a very good approximation to an FD set, or an FD set itself within the relation. It will only provide an indication of this when there are a large number of very good FD sets. Therefore we can state, obviously:

1. If the values of the Bootstrap when a fixpoint is reached are functional or near functional then the majority of possible worlds will satisfy the dependency set functionally or nearly functionally.

2. If the values of the Bootstrap are not nearly functional in proportion to the size of the relation this indicates that most of the definite worlds are poor in terms of satisfying the specified FD set close to functionally.

We can see from this that the Bootstrap is an averaging mechanism. The question of why

the bootstrap provides an upper bound remains. The chase and hill-climbing algorithm exits if the chase heuristic returns an undefined relation for the current highest found ND set, N_T, in the lattice. This implies that the indefinite relation is unable to satisfy any ND sets above N_T. Given that this generally occurs before reaching the limit α (provided by the bootstrap) it seems reasonable to propose that the variance across the possible worlds of an indefinite relation, in terms of ND set satisfaction, is a naive statistic and our hill-climbing and chase heuristic method is sufficient to reach a *good* approximation before examing α initial points. The correspondence between the heuristic and the changing upper limit, due to changing variance of ND set satisfaction in indefinite relations, is to be expected and its usefulness is highlighted in this work.

4.4.6 A Comparison with Jackknife Resampling

The strategy of the jackknife is to remove a single data point from each resample. This allows the creation of n jackknife resamples from an original sample of size n. The bootstrap provides additional flexibility in that the sample is made up of any values uniformly and randomly selected with replacement from the original and, additionally, is not limited to n resamples. In our process the number of worlds required is increased until a fixpoint is reached. Using the jackknife as the worlds reach a large number q we are constrained to q resamples, each of size $q - 1$. Under the bootstrap application we have a fixed number of resamples which, in the majority of cases, will increase to a sample size that is smaller than the q required by the jackknife. We found that the results were very similar for both the bootstrap and jackknife, highlighted in Figure 4.15, despite our use of the bootstrap conducting fewer replications than the jackknife at large sample sizes. Based on the dynamic nature of our resampling often requiring large sample sizes it is therefore much more efficient to use bootstrap and not jackknife resampling. Figure 4.15 also presents the falling limit of the fixpoint as the domain size is held constant but the tuple size increases, due to a reduction in variance within possible worlds as the relation size grows, highlighted in Figure 4.14.

Additional results are given in appendix A for different FD sets; they parallel the results presented.

4.4.7 Real-World Applications

In (Imielinski et al., 1991) we are shown how indefinite information may be used to represent a possible schedule. Our approach allows us to discover an approximation to an *ideal* relation, *ideal* being a relation which satisfies a set of FDs. NDs are a useful tool in this context and indeed schedule representation within relational databases would be enhanced with their use. Any approximation provided by our system for a relation can be analysed by the system users. The schedule which is produced by this, or any other, system can be studied with respect to the result

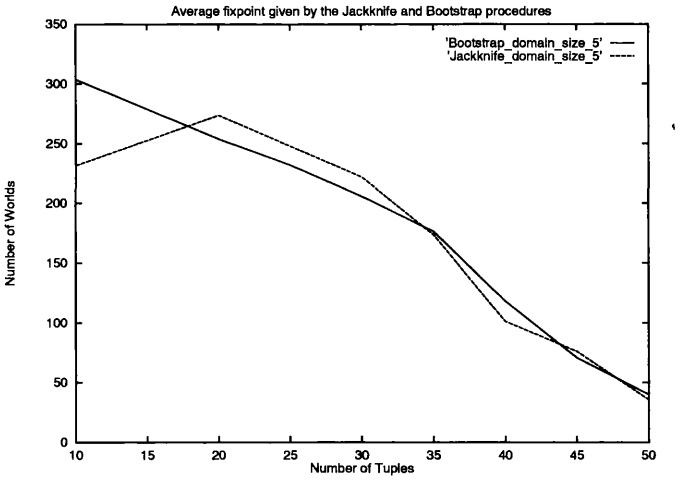

Figure 4.15: Average Number of Worlds given as upper bounds by the Bootstrap and Jackknife techniques for a fixed domain size 5

of our procedures. If there are FDs which are not satisfied within the relation and these are less functional than those provided as output by our chase and hill-climbing approach we can assume a superior schedule exists. The Bootstrap parameters will also tell the user valuable information on the variance and mean of the possible dependency sets which will enhance their knowledge of the indefinite data within the relation.

Section 4.1 briefly mentioned the relationship between the use of indefinite information and constraint logic programming. We can easily see that a domain constraint, stating for example $p = 4$ or $p = 5$ or $p = 6$, can be represented within a relation with an indefinite cell of the form $\{4, 5, 6\}$. A FD can then be used to constrain p to just one of these values. In such a way we note that various constraint problems can be encoded within a database, motivating the use of indefinite information.

NDs, together with the metric presented in Section 4.3.5, are applicable within any relational database for approximating and comparison of FD sets. In a data mining environment this could be used for contrasting approximations in relations over the same attributes which may be in use at different locations. The use of our dynamic resampling procedure, presented in algorithm 10, has applications wherever a non-naive sample size is required to be representative of a population. The approximate fixpoint can be refined empirically based upon the data set and the application to a point where dynamic resampling can be applied to numerous problem instances within the same domain.

4.5 Discussion

We have described how the representation of indefinite information is a valuable extension to relational databases, following on from the work in (Imielinski et al., 1991; Vadaparty and Naqvi, 1995). In addition to this we note that NDs suitably generalise FDs both in a database design context where they may be used in their own right when an FD is too strict (Grant and Minker, 1985a), or within the context of their usage in this chapter where we have used them to approximate FDs based on all possible NDs which may hold within a relation for a given set of FDs forming a complete lattice. The use of NDs extends the work of (Vadaparty and Naqvi, 1995) where relations which do not satisfy the constraint set functionally are said to be *unrealisable*. In many dependency data mining applications, which range from data summarisation to learning within decision trees (Piatetsky-Shapiro and Matheus, 1993), we may wish to obtain a numerical value, between 0 and 1, denoting how close a set of FDs are to being satisfied; the metric, presented in Chapter 3, and used in indefinite relations, achieves this. The consistency problem for relations with indefinite information is widely known to be NP-complete. Therefore we cannot expect to develop a polynomial time based solution unless $P = NP$ or the database is restricted as in (Vadaparty and Naqvi, 1995). Our approach does however introduce an interesting new technique based on sampling, extending the bootstrap to providing useful approximations for problems such as the consistency problem. Essentially, it is based on extracting a representative sample and inducing assumptions on the complete indefinite relation based on the variance within the samples, and subsequently the Bootstrap resamples. We have shown this to provide us with valid upper bounds. The dynamic resampling approach we have presented may be applicable to other NP-complete problems where an approximation to a solution may be useful, using a sample of the data.

The simulations have shown that our procedure can provide useful approximations to FD sets in the form of ND sets for any indefinite relation. We compared different weightings of indefinite information within a relation and showed that as a relation approaches what (Vadaparty and Naqvi, 1995) refer to as a *good database*, one without indefinite information in the left hand side of the dependencies, then the chase procedure for NDs becomes more effective. The efficacy of the chase heuristic, extended in this work to apply to NDs, over naive methods is shown in that best result achieved within a batch is generally found when using the chase and hill-climbing procedure, evidenced in Appendix A. Also, on average around 10% of the worlds used in a naive approach are required by a chase and hill-climbing approach. The bootstrap provides a suitable upper bound with, on average, less than 1% percent of relations generating the number of worlds it takes to reach a fixpoint when using chase and hill-climbing technique. No simulations or empirical discussion is provided in related work, (Vadaparty and Naqvi, 1995; Imielinski et al., 1991).

(Imielinski, 1991) presents a theorem (Theorem 3) which states that no chase-like procedure, namely an algorithm which examines a fixed number of tuples at a time, exists to completely remove all redundant values in relations with indefinite information. (Vadaparty and Naqvi, 1995) notes that other algorithms, however, may achieve this.

A greater understanding of the behaviour of NP-complete problems is provided in (Selman et al., 1992; Mitchell et al., 1992). (Cheeseman et al., 1991) introduces the details of *phase transitions* occurring where NP-complete problems become really hard. These areas are dense in local minima so that there are many near solutions which the search procedure follows. On either side of this critical boundary the problem distribution tends to be either over- or under- constrained. For both of these cases the search is cut off quickly and the probability of success tends to 1 and 0 respectively. Phase transitions occur from a region where most problems are easy and soluble to a region where most are easy but do not contain a solution. However, as (Smith and Grant, 1994) note, there are certain *exceptionally* hard problems on either side of the phase transition which are much harder than those occurring inside the phase transition. A study of these exceptionally hard problems shows that they are the ones most likely to encounter an insoluble subproblem at an early stage. (Cheeseman et al., 1991) points out that complex systems with many interacting values can often be understood at the macroscopic level which characterises the whole system. We should seek, similarly, to understand where such transitions occur for the consistency problem in further work, as they might provide a useful insight into the representation of indefinite information in relations such as a suitable frequency of indefinacy.

Though we have analysed our algorithms behaviour empirically, an approach advocated in (Hooker, 1994), we consider a theoretical analysis to be an interesting avenue for future research, in particular for algorithm 10, our dynamic bootstrap application. Other approximation techniques for the consistency problem, such as finding a suitable subset of an indefinite relation which satisfies an ND set and then adding tuples to this in a hill-climbing fashion would also be interesting to study, both empirically and theoretically. We would also like to see results of dynamic resampling gained from application in other domains.

CHAPTER 5

Temporal Data Mining for Temporal Property Detection

In this chapter we introduce a temporal logic based upon sequences with NDs, possibly representing time series functions, for temporal data mining purposes. We show how temporal properties may be formalised within this logic and used for temporal data mining.

In Section 5.1 we introduce and motivate this work, stating why we focus on NDs. Section 5.2 follows with a discussion of why properties are useful for temporal data mining, concentrating on the ability to succinctly characterise temporal behaviour. In Section 5.3 we briefly present NDs in a temporal database and follow this in Section 5.4 with a presentation of time series analysis. We provide this for two reasons. Principally because our logic uses some time series analysis functions and secondly as a comparison between our work and a standard time series analysis that may be performed on a temporal database, noting that the branching factors of an ND in a temporal database may be viewed as a time series. In the next chapter we shall see some results from applying our logic for temporal property discovery just to time series. Section 5.3.1 provides an introduction to temporal sequences upon which our logic is based. In Section 5.5 we formally define our temporal logic. Finally, in Section 5.6 we define some temporal properties and discuss the intuition behind attempting to discover these properties from a temporal database. (Jaeger et al., 1996) state that, "the task of data mining can be seen as the problem of extracting the interesting part of the logical theory of a model." We consider specific properties to represent interesting patterns within our logic. We conclude with a discussion of the open problems that remain in 5.7.

5.1 Introduction

In Temporal Databases we may view each state at time point t as a snapshot of the database. Over a series of time points, taken at fixed intervals, each snapshot may satisfy changing ND sets which

may model temporal relationships previously unknown to the database user. We assume the time intervals are fixed for clarity within the discovery process though it would be feasible to *unfold* time points over different size intervals into a fixed representation.

The sets of points satisfied by the ND sets across time form a time series. We introduce in Section 5.5 a temporal logic of sequences to model aspects of time series statistics and present them as "properties" of the temporal database. The modal operators are extended from the temporal logic operators of safety, implying at all future points, and guarantee, implying at some point in the future, to implying all subsequences of size n and some subsequence of size n, respectively. In this way we use these operators to characterise the temporal database with such statements as, for example, "all sequences of 100 days contain, at some point, a downward trend of 30 days." The size of the sequence may pertain to a relevant unit of time, such as a month or a week, or length relating to behaviour of the data in question. We also define the $\rightsquigarrow$ temporal operator which represents a non-strict temporal ordering in that overlap is allowed. The expression of temporal behaviour within a succinct logical form allows for both the discovery of new knowledge and the machine understandable form of well understood behaviour within the temporal database. This has applications both in knowledge discovery and decision support.

We show how our logic may be applied to study time series for property discovery. In Chapter 6 we give examples of properties found in temporal datasets which may be viewed as temporal relations. Loosely speaking, properties are formulae within our language which satisfy a template such that properties of a particular nature may be classified as, say, conditional or persistent properties. We motivate their use in Section 5.2. We also provide results showing interesting properties discovered on stocks within the FTSE 100 over different time periods. Additionally, we make use of the resampling technique known as the *moving blocks bootstrap*. From an input time series we randomly sample blocks, or in this case sequences of a size n, and append the sequences to the resampled series as they are selected until we have a resampled series of equivalent length to the original series. The resampling destroys long term relationships whilst preserving relationships of a size less than n, allowing us to look for short range properties which may hold in various time series. We apply our property discovery algorithms to these and the original sequences and provide examples of interesting, useful and previously unknown properties which hold, satisfying all of the criteria for successful knowledge discovery. We do however stress that properties discovered may require expert examination for validation as a contribution to knowledge. This is a key point for all knowledge discovery systems (Fayyad et al., 1996b; Mannila, 1997). We conclude in Section 5.7 with a discussion proposing the inclusion of these techniques into DBMS.

5.2 Why do we need properties for Temporal Data Mining?

There has been much work on properties holding in temporal logic, upon which the seeds of this work lie, most notably (Manna and Pnueli, 1992). Properties in temporal logic have arisen out of the application of temporal logic to computing. Transition rules in a program allow for properties to be specified. For example, the standard notation would use $\Box p \rightarrow \Diamond q$ to denote that at all future points p holds ($\Box p$) which implies that at some point in the future q holds ($\Diamond q$) and this is referred to as a *response to insistence* property. We redefine connectives and properties in our logic so that we may discover various forms of response and persistence rules for temporal sequences. We define a response rule as $\boxminus^n \diamondminus^m \sigma$ which implies that all subsequences of size n ($\boxminus^n$) contain a sequence of size m ($\diamondminus^m$) which satisfies σ, and a persistence rule as $\diamondminus^n \boxminus^m \sigma$ stating that for a sequence of size n all of its m length subsequences satisfy σ, where $m \leq n$. The contribution of this work is the use of property discovery in a temporal logic relating to subsequences for discovering relationships about NDs, the atoms of our logic, in temporal databases.

5.3 Numerical Dependencies in a Temporal Database

In a Temporal Database each snapshot at a particular time may satisfy a set of NDs. We assume that the ND set is specified via an attribute set template provided by the database user, though we note that it is possible to "mine" the relation for NDs blindly as detailed in Section 3.4.

5.3.1 Temporal Relation Sequences

Definition 5.3.1 (Temporal Relation Sequence) A *relation sequence* (temporal database) Δ over R is a finite set of relations over R with $\Delta = \{r_0, r_1, \ldots, r_n\}$, indexed chronologically $0, 1, \ldots, n$ from an initial point 0 and having a final point n, each state corresponding to a time point a fixed interval apart from its previous and next value. □

We assume that our relation sequence, equivalent to a temporal database, is a collection of relations which are linearly ordered. As such we infer within our logic that time itself is linearly-ordered. At each moment there is only one possible future moment. Our underlying sequence is finite. This is natural given that the input for the data mining procedures is a finite sequence of relations.

5.3.2 Time Series Analysis and Numerical Dependencies

We now briefly present the relationship between time series and NDs.

The simple example in tables 5.1 and 5.2 for a relation $COLLEGE(C, S, T)$ over two years where C, S, and T represent course, student and tutor, respectively, highlights possible transition in a temporal database. The change in ND set satisfaction for the ND set = $\{C \rightarrow^k S, C \rightarrow^k T\}$

from $\{C \rightarrow^3 S, C \rightarrow^2 T\}$ to $\{C \rightarrow^4 S, C \rightarrow^1 T\}$ may be an indicator of both increasing student numbers on courses whilst at the same time implying that tutors have more work to do on a course. This information could be represented in a single relation if timestamps were attached to each tuple.

C	S	T
b11a	Paul	Mark
b11a	Tina	Mark
b11a	Fred	Robin
b151	Paul	Robin

Table 5.1: 1997 student intake records

C	S	T
b11a	Tom	Mark
b11a	Dan	Mark
b11a	Louise	Mark
b11a	Jim	Mark
b151	Jim	Robin
b151	Jose	Robin

Table 5.2: 1998 student intake records

Clearly, the change in ND set satisfaction may be viewed as a time series. For example, $C \rightarrow^{16} S, C \rightarrow^{20} S, C \rightarrow^{27} S$ may be viewed as a time series of points 16, 20, 27 assuming a fixed time interval between insertion.

The requirement that for a template of NDs provided for a relation the ND set only changes on the branching factor may be seen as restricting. Schema evolution (Orlowska and Ewald, 1992; Roddick, 1994) may remove an attribute from the relation thereby making an ND in a given set null and void. We assume the following: (1) For the input provided the schema is fixed, and (2) changes in the schema can be assessed by separate mining processes on two separate relation sequences, one before and the other after any schema update.

5.4 Time Series Analysis

We now provide a brief overview of time series analysis. In Section 5.4.1 we discuss research on time series analysis and emphasise areas which our work may be considered as contributory to. Then in Section 5.4.2 we provide definitions of standard functions used within linear time series analysis which are embedded within our logic.

5.4.1 Time Series Analysis: Basics

The goal of time series analysis is to model an observed system so that its future behaviour may be predicted (Weigend and Gershenfeld, 1994). We discuss both traditional time series analysis and new techniques, such as the use of neural networks, and then relate this to our work. Having read this section the reader will fully appreciate the statistical functionality we incorporate into our logic, presented in Section 5.5. We assume familiarity with the statistical functions, such as variance, covariance, correlation, autocorrelation etc, defined in Section 5.4.2.

The standard methodology for analysing a time series is to decompose the series into trend, seasonal and irregular components, each of which may be expressed as individual functions of time. (Weigend and Gershenfeld, 1994) demarcates the difference between understanding and learning from a time series as that of applying explicit mathematical insight for model creation to that of using learning algorithms to emulate the behaviour of the time series. Our goal is closer in spirit to understanding the sequence, using properties to achieve this. For linear and stationary time series one of the most popular techniques is to create an autoregressive (AR) model, of the following form for the Mth order AR model, where the first M autocorrelations determine the coefficients (Enders, 1995):

$$x_t = \sum_{m=1}^{M} a_m x_{t-m} + e_t$$

where e_t represents noise and a_m the autoregressive coefficients for x_t on $x_{t-1}, x_{t-2}, \ldots, x_{t-M}$; e_t is assumed to have expectation 0 and is independent of previous values. Moving Average (MA) models can also be characterised by autocorrelation coefficients describing how values τ steps apart co-vary with each other. (Kendall and Ord, 1990) remark that autocorrelation coefficients for large lags are unreliable for model identification. We found this to be true within our logical representation and adopted their advice of restricting lags of a time series with n points to lags up to $\frac{n}{4}$. This seemed to be a sensible restriction across all time series sizes, given that the reliability of the lag values decrease for higher lags and that we are using sequences of a size chosen by the user which may be arbitrarily short.

AR and MA models may themselves be combined to form ARIMA models, denoting Autoregressive Integrated Moving Average models, integrated implying that we are dealing with a stationary time series, after *differencing*. We omit a full discussion of model selection, provided in (Kendall and Ord, 1990; Enders, 1995), suffice to say that ARIMA models have had the greatest impact on linear time series analysis. Other aspects of time series analysis are the Yule-Walker equations which allow the autocorrelation coefficients of a time series to be expressed by autoregressive coefficients. This is simply understood given their definitions; see (Kendall and Ord, 1990). The restriction of analysis methods to linear time series may cause problems. Two approaches to combat this are:

1. Approximating a system with more than one linear model, known as local linear modelling. (Weigend and Gershenfeld, 1994) state that many regions must be selected if the nonlinearity is of a quadratic degree or greater.

2. The use of differencing to remove trend. (Nazem, 1988; Enders, 1995) comment that most nonstationary time series, where nonstationary implies a trend, can be changed to stationary

time series by differencing once or twice. Given a series $y_1, y_2, \ldots, y_n$ we obtain the first and second order differenced series by $y_2 - y_1, y_3 - y_2, \ldots, y_n - y_{n-1}$ and $y_3 - 2y_2 + y_1, y_4 - 2y_3 + y_2, \ldots, y_n - 2y_{n-1} + y_{n-2}$, respectively. (Nazem, 1988) comments that most economic time series are stationary after at most second order differencing. (Rafiei, 1999) refers to differencing as *momentum*.

Our logic incorporates aspects of local linear modelling by breaking a time series into sequences which may then be linearly regressed within the sequence; we also allow differencing within our logic. (Nazem, 1988) states that "the best practical approach in examining a series is visual examination of the plot of the series." It is a key intention of this work to provide a definite contribution to any visual examination of a time series.

Nonlinear time series have most recently been the subject of analysis by neural networks. (Weigend and Gershenfeld, 1994) stresses the importance of differentiating between learning for model discovery and simple memorisation. The latter occurs when the data is overfitted and prediction relies too heavily on previous values (including noise) rather than looking for a model. The complexities of non-linear time series analysis are outside the remit of this work. We believe that the application of sequences to differenced, and/or moving averaged, time series implies that our procedures can still obtain meaningful properties from such non-linear series. This is due to the fact that though there may not be any global linear properties of the time series our use of sequences breaks the time series up and within these sequences there may be linear behaviour allowing for potentially interesting knowledge discovery. (Cleveland and Loader, 1996) notes some strengths of local regression stating it adapts well to high curvature, can be tailored for many distributional assumptions, and is easy to understand and implement.

5.4.2 Time Series Analysis: Definitions

We now present the standard statistical functions used within linear time series analysis (Kendall and Ord, 1990).

Definition 5.4.1 (Variance) Given a time series x of length n, its variance is written as $var(x)$ where

$$var(x) = \frac{1}{n-1}\sum_{i}^{n}(x_i - \mu)^2$$

We assume that the series is stationary having a mean value μ. □

Definition 5.4.2 (Standard Deviation) Given a time series x its standard deviation is σ_x where

$$\sigma_x = \sqrt{var(x)} \qquad \square$$

Definition 5.4.3 (Covariance) Given two time series x and y, both of length n, their covariance is written as $cov(x, y)$, where

$$cov(x, y) = \frac{1}{n}\sum_{i}^{n}(x_i - \mu_x)(y_i - \mu_y)$$

We assume that the series x and y are stationary with mean values μ_x and μ_y, respectively. □

Covariance is a measure of the linear association between two variables. The strength of the relationship unfortunately depends on the unit of measurement used and so to avoid this we introduce the correlation coefficient.

Definition 5.4.4 (Correlation Coefficient) Given two time series x and y the correlation coefficient $cor(x, y)$ is

$$cor(x, y) = \frac{cov(x, y)}{\sigma_x \sigma_y} \qquad \square$$

The regression coefficient determines the slope for a series of values where y is time when dealing with temporal sequences.

Definition 5.4.5 (Regression Coefficient) Given a time series x, the regression coefficient $reg(x)$ is

$$reg(x) = \frac{cov(x, y)}{\sigma_y}$$

where y represents time. □

We note that regression is equivalent to correlation but without the standard deviation of x in the denominator. Therefore, unlike regression, correlation does not make a distinction between the y-value and the value upon which it is regressed, in our case time. Another process for determining the trend of a sequence is to use discordance which sums the value comparisons over all possible pairs of values to determine trend, defined as:

Definition 5.4.6 (Discordance Test) Given a time series $y = \{ y_1, y_2, \ldots, y_n \}$, we let

$$\begin{aligned} q_{ij} &= 1, \quad \text{if} \quad y_i > y_j \quad \text{when} \quad j > i \\ &= 0, \quad \text{otherwise} \end{aligned}$$

We define Q as:

$$Q = \sum_{i<j}\sum q_{ij}$$

Now, this series is random under the null hypothesis and since there are n points in the time series then there are $\frac{1}{2}n(n-1)$ pairs and so the expected value of Q, E(Q) = $\frac{1}{4}n(n-1)$ Our

discordance function for a time series y is:

$$\begin{aligned} discord(y) &= 1, \quad Q < E(Q) \\ &= -1, \quad Q > E(Q) \\ &= 0, \quad \text{otherwise} \end{aligned} \quad \square$$

Autocovariance and autocorrelation are presented as we may wish to compare sequences of the same time series.

Definition 5.4.7 (Autocovariance) Given a time series x, of length n, its autocovariance of lag k (or lead $-k$) is written as $autocov(x,k)$ where

$$autocov(x,k) = \frac{1}{m}\sum_{i}(x_i - \mu_x)(x_{i-k} - \mu_x)$$

$$autocov(x,k) = \frac{1}{m}\sum_{i}(x_i - \mu_x)(x_{i-k} - \mu_x) \quad \text{where} \quad m = n - k$$

We assume that the series x is stationary with mean value μ_x. $\square$

Definition 5.4.8 (Autocorrelation Coefficient) Given a time series x the correlation coefficient $acor(x,k)$ is

$$acor(x,k) = \frac{autocov(x,k)}{\sqrt{var(x)var(x-k)}} \quad \square$$

Definition 5.4.9 (Cross Covariance) Given two time series x and y, both of length n, their cross covariance of lag k (or lead $-k$) is written as $ccov(x,y,k)$ where

$$ccov(x,y,k) = \frac{1}{n}\sum_{i}^{n}(x_i - \mu_x)(y_{i-k} - \mu_y)$$

We assume that the series x and y are stationary with mean values μ_x and μ_y, respectively. $\square$

Definition 5.4.10 (Cross Correlation Coefficient) Given two time series x and y the cross correlation coefficient $ccor(x,y,k)$ is

$$ccor(x,y,k) = \frac{ccov(x,y,k)}{\sigma_x \sigma_y} \quad \square$$

5.4.3 Catalytic Data Mining

Catalytic Data Mining is a term introduced by (Hale and Shenoi, 1995) for the data mining of two or more relations which agree on a common attribute or more so that the data mining process can be enhanced. We mention it here given that it applies to NDs in temporal relations. Our data mining process is such that for a company we can extract NDs from either an employee or product sales relation and then perform the mining on this together with a directly numerical value such as the stock price. Over time a fall in stock price combined with little change in an ND in the sales relation each October may suggest that a sale is held at this time of year to increase sales.

5.4.4 Advantages of a logical approach

Standard time series techniques allow us to apply time series functions to time series naively. This, in turn, may produce useful results such as a high cross-correlation between two time series. A symbolic representation of this provides similar information without recourse to numerical comparison. Therefore our logic is algorithmic and as such is amenable to symbolic manipulation. This implies that it is of use within decision support tools and perhaps general data mining systems.

The logic is also flexible so that many different kinds of relationships and patterns can be expressed in a very concise form. This is a result of using a high-level logic to represent desired concepts.

5.5 Numerical Dependency Linear Temporal Logic

We now formalise our temporal logic for sequences which we refer to henceforth as NDLTL. Much of the intuition behind sequences follows from Allen's temporal intervals which we advise reading for a clear understanding of the use of intervals and sequences in time (Allen, 1984).

5.5.1 Temporal Logic

Propositional Linear Temporal Logic is propositional logic augmented with the modalities $\mathcal{S}$ and $\mathcal{U}$, denoting since and until, respectively, defined in (Gabbay et al., 1994). For atoms A and B, A $\mathcal{S}$ B is true at time t_n, if for time t_0 where $t_0 < t_n$, if B is true at t_0 and for all points between t_0 and t_n, A is true. Similarly, A $\mathcal{U}$ B is true at t_p if for some t_q where $q > p$, B is true at t_q and for all points between t_p and t_q, A is true. From these modal primitives further temporal operators may be defined, of which the principal ones are $\Box$ and $\Diamond$, implying, respectively, at all future points and at some point in the future. $\Diamond$ A may be defined as *true* $\mathcal{U}$ A, $\Box A$ is the dual of $\Diamond$ A defined as $\neg\Diamond\neg$ A. $\bigcirc$ A, representing *nexttime* A, may be defined as *false* $\mathcal{U}$ A

A logic may be created due to concerns that there are inadequacies within previous logics to represent various kinds of informal argument (Haack, 1978). We wish to represent arguments representing aspects of time series analysis within a logical form that allows patterns in the temporal sequences to be represented; a logic is a system for obtaining answers from Δ (Gabbay et al., 1994). Our logic is with respect to sequences of temporal relations, equivalent to the interval representation of time (Allen, 1984). We modified our logic to contain $\rightsquigarrow$ and $\boxminus^n$ as primitives with respect to sequences. Formulae of the form $\sigma_1 \rightsquigarrow \sigma_2$ imply that a sequence s_1 starts before s_2 and ends before s_2 ends, with s_1 satisfying σ_1 and s_2 satisfying σ_2. If a sequence s_1 satisfies $\boxminus^n\sigma$ this implies that all sequences of size n in s_1 satisfy σ.

Another possible approach would have been to incorporate the time series operators at the atomic level and apply these within a standard temporal logic for knowledge discovery. We now show by example some potential problems with this. A sequence s may satisfy ccor(σ_1, σ_2, k_1) $\mathcal{U}$ ccor(σ_1, σ_2, k_2). Even if we allow the inclusion of such time series functions we are likely to discover that time series are unlikely to satisfy a formulae A $\mathcal{U}$ B without numerous conjuncts leading to formulae such as A $\mathcal{U}$ B $\mathcal{U}$ C $\mathcal{U}$ A $\mathcal{U}$ B. The problem with using standard *point based* temporal logic for the discovery of such formulae is that they are apt to *overfit* the data; in this brief example we have A $\mathcal{U}$ B holding twice. It would be of more value to be able to represent this fact, which our logic of sequences achieves in some respect. Standard temporal logic models atoms occurring at certain time points such as inferring A $\mathcal{U}$ B over a period of p time points. This, within a point based logic, may lead to the discovery of many hundreds of formula. However, we approach our discovery from the basis of creating sequences so that we control the granularity of property discovery given that we may have to deal with many hundreds of time points. Though this may result in valuable knowledge found we believe a sequence based logic results in finding *interesting* knowledge more easily. Similarly $\Box A$, denoting at all points in the future A holds in temporal logic, is unlikely to either be satisfied or, if it does, represent interesting information. Another potential problem is that the discovery of temporal logic formulae without restriction may often be too complex for efficient knowledge discovery; formulae such as (C $\mathcal{U}$ A) $\mathcal{S}$ (B $\mathcal{U}$ C) or $\Box$ (A $\rightarrow \Diamond$ D). These formulae themselves do not represent complex behaviour, for example, the former proposition may hold if C occurs between one or more occurrence of both B and A. Therefore we choose to restrict our discovery to search for what we believe are interesting formulae.

These deficiencies suggest that formulae be verified with respect to sequences. Therefore, we have modified our modal operators with respect to sequences. We also introduce $\rightsquigarrow$; this is a temporal ordering operator which allows overlap. We motivate its inclusion based on the fact that within time series and temporal sequences strict transitions of properties may not occur. The formalisation of $\rightsquigarrow$ is sufficiently flexible yet restrictive enough to discover interesting patterns within and across sequences. The requirements outlined in (Gabbay et al., 1994) for the components for specification of a temporal logic are all provided in our formal definition apart from allowing our units of time to vary across data sets, though we may generalise and say that time is the set of integers, satisfied in all data sets.

We shall demonstrate, informally, how sequence-based temporal operators are more appropriate for data mining applications. As we shall see in section 5.5.6 our discovery process is computationally efficient due to our restriction of fixed sequence sizes which allow for polynomial

time knowledge discovery. We assume that we have as input a finite temporal database; given this it is of minimal value to search for certain temporal logic formulae. The discovery of $\Diamond\sigma$ in state i, say, tells us little in a knowledge discovery sense. Additionally, the random discovery of useful formula is a challenging task given that there may be a significant number of different patterns within a temporal database. The division of an input sequence into all possible sequences of a size chosen by the user allows knowledge discovery over all possible different time periods within the input sequence. Sentences not containing any of the sequence or temporal operators reduce to sentences of classical propositional logic with NDs and ND time series functions as atoms.

Our logic has additional operators which incorporate a means of representation for time series analysis techniques within our logic. Therefore our logic allows information about the input data to be analysed with respect to a given time period specified by the user within which techniques such as regression and correlation are applied and then the rules for the time periods themselves are analysed for possible properties which may hold in a sequence. The complexity of temporal logic and time series implies that it is highly unlikely that we will discover a rule of the form $\Box\sigma$ at a particular point, unless the input is near trivial in which case it is uninteresting. Similarly, $\Diamond\sigma$ is uninteresting due to its general application.

The modal operator $\boxminus^n$ is not, in the strict sense, a temporal operator. A system is considered temporal if it has an (irreflexive transitive) ordering $<$ (Gabbay et al., 1994). Our operator $\boxminus$ makes use of inclusion (of sequences), see Definition 5.5.2, which refers to set containment as opposed to ordering.

We remark that our logic is non-monotonic in that properties discovered for a sequence may not hold if we apply the same discovery algorithms to an updated version of the same sequence, containing a longer sequence. As such fewer properties are likely to hold demonstrating the non-monotonicity of this approach. In temporal databases logics for integrity constraints are necessarily monotonic, unless the semantics of the database are altered; this is not the case for data mining. (Gabbay et al., 1994) notes that monotonicity in temporal logic requires further study.

5.5.2 Syntax

We refer to a particular relation at state j within a relation sequence Δ as (Δ, r_j). We refer to a subsequence s of Δ ($s \preceq \Delta$) as (Δ,s). r_j is relation at point j and $r_j \models N_j$, the set of NDs satisfied by r_j and N_j is an approximation to an FD set F which is given as input in our discovery model. Δ may be omitted if the sequence is understood from the context. We may use $\Delta \models \sigma$ to represent (Δ,Δ) $\models \sigma$.

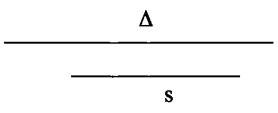

Figure 5.1: Sequence Inclusion, $s \preceq \Delta$

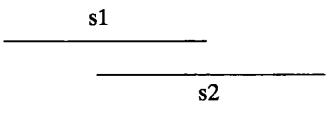

Figure 5.2: Sequence Ordering, $s1 \lessdot s2$

We define two operators for inclusion and ordering of sequences referred to in the semantics of our logic, after initially defining a sequence.

Definition 5.5.1 (Sequence) s is a sequence of relations within a temporal relation sequence Δ *iff* $\forall r_i \in s$ we have $\neg\exists r_l \in \Delta$ such that $r_j \leq r_l \leq r_k$ and $r_j, r_k \in s$ but $r_l \notin s$. □

Definition 5.5.1 enforces that all sequences are continuous.

Definition 5.5.2 (The inclusion operator, $\preceq$) $s \preceq \Delta$ *iff* $\forall r_i \in s$ we have $r_i \in \Delta$ and s is a sequence in Δ. □

$s \preceq \Delta$ implies that s is a subsequence of Δ and that s contains a series of consecutive states, as illustrated in 5.1.

Definition 5.5.3 (The Temporal Ordering operators, $\lessdot$ and $\gtrdot$) $s_1 \lessdot s_2$ *iff* $\exists r_j \in s_1$ such that $\forall r_k \in s_2$ we have $j < k$ and $\exists r_p \in s_2$ such that $\forall r_m \in s_1$ we have $p > m$ and s_1, s_2 are sequences. $\gtrdot$ is defined similarly. □

The intuition behind our temporal ordering operator is that a sequence comes before another sequence if at least one point in the sequence s_1 is before any in s_2 and s_2 has at least one point after s_1. As desired, a subsequence of another sequence does not satisfy this relation. Figure 5.2 shows an example of sequence ordering with an overlap between sequences.

The set of formulae of NDLTL is the least set generated by:

1. Each ND $X \rightarrow^{k^*} Y$ or $X \rightarrow^{\updownarrow k^*} Y$ is an atomic formula where $k^* \in \{k, \bar{k}, \ddot{k}\}$ and $\updownarrow \in \{\uparrow, \downarrow, \uparrow_r, \downarrow_r, \uparrow_d, \downarrow_d\}$.

2. If σ_1 and σ_2 are formulae then so are $\neg\sigma_1, \sigma_1 \wedge \sigma_2, \sigma_1 \wedge^k \sigma_2$ (k is a constant) and $\sigma_1 \rightsquigarrow \sigma_2$.

3. If σ is a formula then so are $\boxminus^m \sigma$ and $\diamondminus^m \sigma$.

We now define the semantics of our logic inductively.

5.5.3 Semantics

Relation state formulae are not concerned with the sequence size used. They are obtained by performing operations on the complete temporal relation sequence, Δ. These operations represented

the ND holding at a particular state, the moving average value for a window size w, or the differenced value for a particular state. We write $(\Delta, u) \models^w \sigma$ to mean that σ holds with respect to a window size w in sequence (or state) u of the temporal relation sequence Δ. We may omit w if it is not pertinent to σ; for example, σ may be an ND alone and u may be a single relation state. The discussion of Section 5.4 motivated the need for differencing. Second order differencing, not directly expressible in our logic, is occasionally required; an additional axiom could be added similar to our differencing formulae.

Relation State Formulae:

1. $(\Delta, r_j) \models^w (X \rightarrow^k Y)$ *iff* $r_j \models X \rightarrow^k Y$.

2. $(\Delta, r_j) \models^w (X \rightarrow^{\bar{k}} Y)$ *iff* $r_{j-m} \models X \rightarrow^{k_1} Y, r_{j-(m-1)} \models X \rightarrow^{k_2} Y, \ldots, r_j \models X \rightarrow^{k_{m+1}} Y, r_{j+1} \models X \rightarrow^{k_{m+2}} Y, \ldots, r_{j+m} \models X \rightarrow^{k_n} Y$ where $m = \frac{w-1}{2}$, w is odd and $\bar{k} = \frac{1}{w}\sum_{i=1}^{w} k_i$ and $j - m \geq 0$ and $j + m \leq 0$.

3. $(\Delta, r_j) \models^w (X \rightarrow^{\ddot{k}} Y)$ *iff* $j > 0$ and $r_{j-1} \models X \rightarrow^{k_1} Y$ and $r_j \models X \rightarrow^{k_2} Y$ with $\ddot{k} = k_1 - k_2$.

Definition 2 provides the moving average value for a *window* of size w for the branching factors of the NDs for relation state j. At times we omit w from the following definitions for clarity. Definition 3 above provides a difference ND values, from which a differenced sequence may be created.

All the following time series functions used are defined in Section 5.4.2.

Relation subsequence trend formulae are required to represent trends within our logic. These may be strict, linearly regressed, or discordant trends. Informally, a $\uparrow$ denotes an upward trend and $\downarrow$ a downarrow trend. We may subscript these with either a d or an r to represent discordance or regression, respectively. In the following formulae we abbreviate *such that* to *s.t.* and denote the first relation r_i in a sequence s by $fst(s)$. We also refer to a particular ND $X \rightarrow^k Y$ sequence in s as s_{XY} where necessary.

Relation Subsequence Trend Formulae:

1. $(\Delta, s) \models^w (X \rightarrow^{\uparrow \bar{k}_1} Y)$ *iff* $\mid s \mid \geq 2$ and $\forall r_j, r_{j+1} \in s\ (\Delta, r_j) \models^w X \rightarrow^{\bar{k}_1} Y$ and $(\Delta, r_{j+1}) \models^w X \rightarrow^{\bar{k}_2} Y$ for some $\bar{k}_2$, where $\bar{k}_1 \leq \bar{k}_2$.

2. $(\Delta, s) \models^w (X \rightarrow^{\downarrow \bar{k}_1} Y)$ *iff* $\mid s \mid \geq 2$ and $\forall r_j, r_{j+1} \in s\ (\Delta, r_j) \models^w X \rightarrow^{\bar{k}_1} Y$ and $(\Delta, r_{j+1}) \models^w X \rightarrow^{\bar{k}_2} Y$ for some $\bar{k}_2$, where $\bar{k}_1 \geq \bar{k}_2$.

3. $(\Delta, s) \models^w (X \rightarrow^{\uparrow_r k} Y)$ *iff* $\mid s \mid \geq 2$ *s.t.* $r_i = fst(s)$ and $(\Delta, r_i) \models^w X \rightarrow^k Y$ and $reg(s_{XY}) > 0$.

4. $(\Delta, s) \models^w (X \rightarrow^{\downarrow_r k} Y)$ *iff* $\mid s \mid \geq 2$ *s.t.* $r_i = fst(s)$ and $(\Delta, r_i) \models^w X \rightarrow^k Y$ and $reg(s_{XY}) < 0$.

In definitions 3 and 4 we can replace the trend operator $\uparrow_r$ with $\uparrow_d$ to represent that we have obtained the trend using the discordance method, Definition 5.4.6, in place of linear regression.

We now present two rules which allow for the representation of a trend without an explicit initial value; these rules are necessary when we may want to represent such a rule within our operator for all sequences of a size n ($\boxminus^n$) and the regression values may differ between sequences though the general trend may be the same.

1. $(\Delta, s) \models^w (X \rightarrow^{\uparrow_r} Y)$ *iff* $\mid s \mid \geq 2$ *s.t.* $reg(s_{XY}) > 0$.

2. $(\Delta, s) \models^w (X \rightarrow^{\downarrow_r} Y)$ *iff* $\mid s \mid \geq 2$ *s.t.* $reg(s_{XY}) < 0$.

Relation Subsequence Formulae are required to represent the more complex relationships within our temporal database. σ, possibly subscripted, is either a relation state formulae or a relation subsequence trend formulae and s may be a relation state or sequence. We note that $\wedge$ may be superscripted by k; this denotes a fixed value maximum lag cross-correlation for the sequence between σ_1 and σ_2. If σ_1 and σ_2 relate to the same sequence then this becomes auto-correlation; we do not consider this further due to autocorrelations for linear time series following a dampening oscillatory pattern towards 0 as the lag increases (Enders, 1995). The use of autocorrelations for non-linear time series in this logic is deserving of further study. $\boxminus^n$ is a universal operator for all sequences of size n and $\diamondminus^n$ is its existential dual. The final operator is $\rightsquigarrow$ which introduces a temporal ordering between two sequences allowing for change in a temporal database to be depicted; we shall see in Section 5.6 how these operators are used to form temporal properties. We require in the following definitions that all sequences are maximal.

Relation Subsequence Formulae:

1. $(\Delta, s) \models \neg\sigma_1$ *iff* $(\Delta, s) \not\models \sigma_1$.

2. $(\Delta, s) \models \sigma_1 \wedge \sigma_2$ *iff* $(\Delta,s) \models \sigma_1$ and $(\Delta,s) \models \sigma_2$.

3. $(\Delta, s) \models \sigma_1 \wedge^k \sigma_2$ *iff* $(\Delta,s) \models \sigma_1$ and $(\Delta,s) \models \sigma_2$ and k is the lag value for which the cross-correlation is maximum.

4. $(\Delta, s) \models \sigma_1 \rightsquigarrow \sigma_2$ *iff* $(s,s_1) \models \sigma_1$ and $(s,s_2) \models \sigma_2$ for some $s_1, s_2 \preceq s$ where $s_1 \lessdot s_2$.

5. $(\Delta, s) \models \boxminus^n \sigma_1$ *iff* $\forall s_i \preceq s$ where $\mid s_i \mid = n$ and $(s, s_i) \models \sigma_1$.

6. $(\Delta, s) \models \diamondminus^n \sigma_1$ *iff* $\exists s_i \preceq s$ where $\mid s_i \mid = n$ such that $(s, s_i) \models \sigma_1$.

We note in rule (4) that $s_1, s_2 \preceq s$ do not have to cover every point in s. Within this logic it is possible to express formulae such as $(\Delta, s) \models \boxminus^n (\sigma_1 \rightarrow \sigma_2)$, stating that for all n size subsequences of s either σ_1 does not hold or σ_2 holds. We restrict our knowledge discovery process to search for positive information though a user might ask such queries.

5.5.4 Examples

We now provide some examples of the logic. From Section 5.3.2 we introduced an example of NDs in a relation of student records. We assume that for the ND $C \rightarrow^k S$ we have 7 years of records and the relation sequence of ND branching factors is Δ_{cs} = { 3,4,5,6,5,4,7 } where each refers to a relation starting from position 0. We now highlight some rules satisfied by this sequence:

1. $(\Delta_{cs}, r_3) \models^3 (\mathrm{C} \rightarrow^{5.\bar{3}3} \mathrm{S})$
2. $(\Delta_{cs}, r_1) \models^3 (\mathrm{C} \rightarrow^{\ddot{\imath}} \mathrm{S})$
3. $(\Delta_{cs}, s_3) \models^3 (\mathrm{C} \rightarrow^{\downarrow 6} \mathrm{S})$ where $s_3 = \{6, 5, 4\}$.
4. $(\Delta_{cs}) \models^3 (\mathrm{C} \rightarrow^{\uparrow r^3} \mathrm{S})$
5. $(\Delta_{cs}) \models^3 \boxminus^4 \diamondminus^3 (\mathrm{C} \rightarrow^{\uparrow r} \mathrm{S})$

Examples 1 and 2 show moving average and differenced values for NDs. In the data mining process we generally do not need to represent these values directly though occasionally it may be required. Example 3 provides a strict downward trend rule within a given subsequence of Δ_{cs} and Example 4 provides similar for a linear regression trend on the sequence. Example 5 states that all sequences of size 4 contain, at some point, a sequence of size 3 for which there exists an upward linear regression. We do not represent the exact value as these differ within sequences but our logic allows for this, highlighted in Section 5.5.3. This is needed given that general properties might exist, as in 5, but the exact values will differ.

5.5.5 Axioms of the logic

We now highlight some additional axioms of our logic, related to sequences, with A and B as formulae of our logic. We use $\Rightarrow$ to denote *implies*.

1. $\diamondminus^n \sigma \equiv \neg \boxminus^n \neg \sigma$
2. $\boxminus^n \mathrm{A} \Rightarrow \diamondminus^n \mathrm{A}$
3. $\boxminus^n (\mathrm{A} \wedge \mathrm{B}) \equiv \boxminus^n \mathrm{A} \wedge \boxminus^n \mathrm{B}$

4. $\Diamond^n (A \vee B) \equiv \Diamond^n A \vee \Diamond^n B$

5. $\boxminus^n (A \rightsquigarrow B) \Rightarrow \boxminus^n (\Diamond^{n_1} A \wedge \Diamond^{n_2} B)$ where $n_1, n_2 \leq n$

6. $\boxminus^m \boxminus^n$ implies that $m \geq n$

7. $\Diamond^m \Diamond^n$ implies that $m \geq n$

8. $\boxminus^m \Diamond^n$ implies that $m \geq n$

9. $\Diamond^m \boxminus^n$ implies that $m \geq n$

Axiom 1 implies that we merely have to define either $\boxminus^n$ or $\Diamond^n$ as primitive. Axioms 1, 2, 3, and 4 are all properties of standard temporal logic. Axiom 5 expresses behaviour concerning the $\rightsquigarrow$ operator. Its proof is clear from the definitions of $\rightsquigarrow$ and $\lessdot$. Finally, axioms 6, 7, 8 and 9 present some properties related to the sequence size requirements of our modal operators.

Some standard axioms of temporal logic do not hold within our logic. These include $\boxminus^m \sigma \Rightarrow \sigma$. All subsequences of size m satisfying σ does not imply that σ holds. This is due to the incorporation of statistical functions which may hold in all sequences of a particular size but not for the sequence as a whole. If we consider a time series there may be a general upward trend within it though this may consist of many local peaks and troughs; our sequence logic is capable of detecting these. $\boxminus^m A \Rightarrow \Diamond^n A$, where $m \neq n$, does not generally hold for the same reason.

5.5.6 Querying our Logic

We now show by induction that any formulae σ in the logic can be tested in polynomial time. Firstly, we present some lemmas which shall be of use in the proof.

Lemma 5.5.1 Given a sequence s of n relation states then s has $\frac{1}{2}n(n+1)$ subsequences

Proof. There are n subsequences of size 1, $n-1$ of size 2, ..., 1 of size n. $n + (n-1) + (n-2) + \ldots + 1 = \frac{n(n+1)}{2}$. □

We note that a sequence of size 1 contains two relations, temporally ordered.

Lemma 5.5.2 The number of subsequences which start in position k in a sequence of size n is $n - k$. We assume the first position is denoted by 0.

Proof. Trivial. □

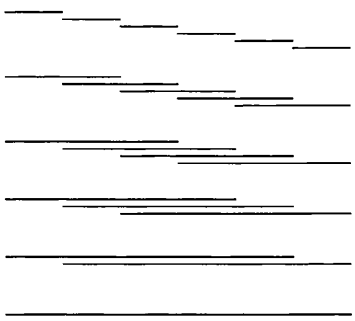

Figure 5.3: All possible subsequences in a sequence containing 7 relations

Lemma 5.5.3 Assuming a total sequence length m and a sequence s which begins in position k of the complete sequence. The number of sequences which start after and end after a sequence s of size n is:

$$\sum_{i=k+1}^{m} m - i - \sum_{i=1}^{n-1} n - 1$$

Proof. Clear, using the facts that we wish to count all sequences which start at position $k+1$ onwards to the end of the sequence which sums lemma 5.5.2 and we wish to exclude those subsequences from position $k+1$ which do not end after s, therefore we remove all subsequences contained at each position from $k+1$ to $k+n$ that are contained in a sequence of size $n-1$. □

Theorem 5.5.4 Given a subsequence s of a relation sequence Δ and a formulae σ in NDLTL we prove that $(\Delta, s) \models \sigma$ can be shown in polynomial time.

Proof. We show inductively that we can test if $(\Delta, s) \models \sigma$ in polynomial time. We base our proof on the structure of σ.

(*Basis*): If σ is an atomic formula of the form $X \rightarrow^{\uparrow k} Y$ or $X \rightarrow^{\downarrow k} Y$ we examine each consecutive pair of points (relations) in s to determine if the trend is satisfied. This can be achieved in linear time. Similarly we can test branching factor values of regression and discordance in linear and polynomial time respectively.

(*Induction*): We now consider all possible structures of σ. If σ is of the form:

1. $\neg\sigma$ we need to determine if $(\Delta, s) \not\models \sigma$. We assume, by the inductive hypothesis, that σ can be checked in polynomial time. Therefore it is easy to see that $\neg\sigma$ can also be checked by polynomial time.

2. $\sigma_1 \wedge^k \sigma_2$ - we assume that σ_1 and σ_2 can be determined in polynomial time and it is clear that the result of the cross-correlation function ccor(br(σ_1),br(σ_2),k) can be computed in polynomial time. Similarly $\sigma_1 \wedge \sigma_2$ holds.

3. $\sigma_1 \rightsquigarrow \sigma_2$ can be tested in time polynomial from lemma 5.5.3.

4. $\boxminus^n \sigma$ can be determined if for all subsequences of s we can show that σ holds. Given $\boxminus^n$ there are $\frac{n(n+1)}{2}$ subsequences of s where n is at most the size of the sequence. We assume that each subsequence can be checked in time polynomial, implying that $\boxminus^n \sigma$ can be checked in polynomial time.

5. $\diamondminus^n \sigma$ holds if there is at least one subsequence s_2, of size n, of s such that $(s, s_2) \models \sigma_1$ and it is clear that we can examine each subsequence of this individually in polynomial time.

We have shown by induction on the structure of σ_1 that we can check if this is satisfied in polynomial time. □

We now show why we restrict ourselves to sequences of a fixed length. We define a *cover* to be a set of sequences of different sizes whose union contain the complete sequence. In a time series there may be properties which hold across different sequence sizes, however we now show that there is an exponential number of such *covers*.

Lemma 5.5.5 Given a sequence s_n containing n relation states then s_n has $\frac{(n+1)!}{2}$ covers, where $n!$ denotes n factorial.

Proof. If there is 1 relation in a sequence there is only 1 cover. If $n > 1$ then we state that there are m_n covers. We note that an $n-1$ size sequence has m_{n-1} covers. A sequence s_n will have one more relation in the sequence than s_{n-1}. The additional relation state in s_n provides an additional $n+1$ ways of combining with m_{n-1} so we have the recurrence relation

$$\begin{aligned} m_1 &= 1 \\ m_n &= (n+1)m_{n-1} \end{aligned}$$

We can easily see that $m_n = (n+1)(n)(n-1)(n-2)\ldots 3.1$ which is equivalent to $\frac{(n+1)!}{2}$ □

We therefore restrict ourselves to fixed length sequences in the logic for safety properties ($\boxminus^n$) defined in Section 5.6.

5.5.7 Expressiveness of NDLTL

We wish to know how expressive are the temporal connectives of our logic? In propositional temporal logic we know that the connectives *since*, $\mathcal{S}$, and *until*, $\mathcal{U}$, are defined as fully expressive by (Gabbay et al., 1980) due to their satisfaction of the separation property, where any formulae can be rewritten as a combination of past, present, and future, over integer time. The only temporal operator our logic permits is $\rightsquigarrow$, due to $\boxminus^n$ and $\diamondminus^n$ relating to sequences. We are only interested

in certain temporal orderings within sequences and we know that use of $\mathcal{U}$, or other temporal operators, might frequently be too restrictive; the changing nature of ND values in a temporal relation sequence and in a temporal database may require many instances of $\mathcal{U}$. We consider the expressiveness of our logic with respect to time series as the capability of expressing patterns.

We consider the following for pattern expression with respect to linear regression, without loss of generality. Within any single time series and for a sufficient sequence size we can express patterns in a time series obtained using linear regression and using the connective $\rightsquigarrow$. At the finest granularity these sequences may contain only two points. If we consider multiple time series the sequence size must be fine enough for representation of change in any of the patterns in any of the time series. Our temporal operator $\rightsquigarrow$ is not expressively complete as we can not state sentences of the form 'A before B'. It is reasonable to view the restriction to fixed length sequences as necessary for uniform property discovery, given the result of lemma 5.5.5. Any sequence of size n can be separated into subsequences of any fixed size m, where $m \leq n$, which enhances the expressive nature of the logic. Due to lemma 5.5.5 it is not feasible to consider otherwise within an applied scenario.

When we move into time series issues of expressive completeness become vague as we do not know what is complete when denoting the relationship between two or more time series. The temporal and modal connectives of our logic are clearly incomplete and require further study; we have formulated them such they are sufficient for the data mining task at hand.

5.6 Temporal Logic Properties

We now show how we can use our logic for the expression of properties which may hold for a temporal sequence. Initially, we informally discuss the intuition behind the properties holding in our logic and provide a brief survey of the field.

(Pnueli, 1977) introduced the application of temporal logic for program verification. (Gabbay et al., 1980) presented numerous *properties* for the application of temporal logic to reactive and concurrent (non-terminating) programs. Such programs could not utilise previously developed correctness methods given that these methods were intended for finite (terminating) programs. Much of the literature refers to the two general forms of properties, *safety* and *liveness*. Safety properties refer to the intuition that "nothing bad ever happens" whilst liveness properties imply that "something good eventually happens" (Prasad-Sistla, 1994). We can see in program verification how we might want to prevent particular conditions from ever being satisfied (safety) whilst at the same time ensuring that specific condition are satisfied intermittently (liveness). From these definitions we can infer the dual universal and existential natures of safety and

liveness. These are also referred to as the two most general classes of invariances and eventualities.

We now provide two brief examples of safety and liveness properties from program verification (Emerson, 1990). If we assume that s and h are the initial and final labels of a program then $s \wedge \phi \Rightarrow \Box(h \Rightarrow \psi)$, where $\Rightarrow$ denotes implication, is a safety property. It infers that if a program in state s satisfies ϕ then at all points in the future the final state implies ψ. We view ϕ and ψ as pre- and post-conditions. Mutual exclusion and deadlock prevention principles are also examples of safety properties. Liveness properties consist of intermittent assertion, total correctness and guaranteed accessibility conditions. To illustrate, $s \wedge \phi \Rightarrow \Diamond(h \wedge \psi)$ is the total correctness property implying that a program starting in s satisfying ϕ will, at some point in the future, halt in h satisfying ψ. We can see from these simple examples how temporal logic is directly applicable for program verification. Proof theoretic methods for program verification, not considered here, have been developed and the most appropriate proof method depends on the property being verified.

Properties may take numerous forms which we now outline. Safety properties have the canonical form $\boxminus^n \sigma$. We may refer to invariant NDs as safety properties. The "informal" description of safety, stating that nothing bad ever happens implies that it is a bad thing if σ does not hold in a relation. (Prasad-Sistla, 1994) describes *strong safety* properties which remain safety properties after the exclusion of a state. In our logic all possible sequences satisfying a property imply that it is a safety property. For example $(\Delta,s) \models \boxminus^n (X \rightarrow^{\uparrow k} Y)$ implies that all subsequences of s satisfy an upward trend from value k. Conditional safety properties have the form $\sigma_1 \rightsquigarrow \boxminus^n \sigma_2$. We can refer to this as a *trigger*, denoting that once σ_1 occurs then at some point after σ_2 will hold in all sequences of size n. It is apt within a finite temporal sequence wherein a certain value may imply other values for all subsequent states.

Guarantee properties have the canonical form $\diamondminus^n \sigma$. A conditional guarantee property implies that all subsequences of a fixed size n always contain a sequence where $s_1 \lessdot s_2$ and $(\Delta,s_1) \models \diamondminus^m \sigma_0$ which leads to $(\Delta,s_2) \models \diamondminus^m \sigma_1$. This is written as $(\Delta,s) \models \boxminus^n (\sigma_0 \rightsquigarrow \diamondminus^m \sigma_1)$ where $s_1, s_2 \preceq s$.

Obligation properties are the disjunction of canonical safety and guarantee properties $\diamondminus^n \sigma_1 \vee \boxminus^m \sigma_2$. A canonical obligation formula is a conjunction of these properties. Note that the sequence sizes need not be the same, though the semantics of this in a knowledge discovery context is not clear.

Response properties are of the general form $\boxminus^n \diamondminus^m \sigma$. A Canonical response property can

also be viewed as a seasonal occurrence property and is represented by $(\Delta,s) \models \boxminus^n \diamondminus^m \sigma$. Within sequence s all sequences of size n contain an occurrence of σ which will hold at some point in the s. The temporal logic definition states that at a particular point all points imply that σ will hold at some point in the future. Our definition is that all sequences of size n will, at some point, contain a sequence of size m which satisfies σ. It is clear that we can use this property to present seasonal behaviour to the system user. Alternatives for standard temporal logic, listed in (Manna and Pnueli, 1992), are $p \rightarrow \Diamond q$, known as response to an impulse, and $\Box(p \rightarrow \Diamond q)$ where $\Diamond$ implies at some point in the future and $\Box$ implies at all points in the future. Within our logic a response property may occur only finitely many times. A seasonal response property implies a regular chain of two, or more, events written as $(\Delta,s) \models \boxminus^n \diamondminus^m (\sigma_0 \rightsquigarrow \sigma_1)$. An immediate response property in temporal logic states that at a particular point all points imply that σ will hold at the next point, written as $\Box \bigcirc \sigma$. We do not incorporate this within our logic of sequences given that the representation of the *next* sequence will often, depending on sequence size, contain a significant overlap which would frequently result in the discovery of uninteresting properties.

Persistence properties in standard temporal logic, written as $\Diamond\Box p$, may be triggered by a preceding event where we infer that all positions from a certain time on satisfy p. This directly translates to our logic where we write persistence as $(\Delta,s) \models \diamondminus^n \boxminus^m \sigma$. In our sequence logic a persistence property implies that at some point in a sequence s of size n all sequences within s of size m satisfy σ. This allows us to depict properties which may hold in nonlinear time series, for example, a continuous downward trend in an otherwise rising microchip stock due to an unforeseeable influence, such as fires in the chip factory destroying stock. It is likely that this would otherwise represent nonlinear behaviour. Ordered persistence properties, written as $(\Delta,s) \models \sigma_0 \rightsquigarrow \diamondminus^n \boxminus^m \sigma_1$ are not directly relevant in our work due to a single occurrence of σ_0 possibly having no relation to $\diamondminus^n \boxminus^m \sigma_1$.

Reactive properties, shown in (Manna and Pnueli, 1992) to be the maximal class of properties which needs to be considered, are written as $\boxminus^n \diamondminus^m \sigma_1 \vee \diamondminus^p \boxminus^q \sigma_2$. Within the finite sequence of our logic and the knowledge discovery process such a property found would represent complex behaviour. Such a reactive property might denote the oscillation between σ_1 occurring sporadically in all n size sequences and σ_2 holding continually within all q size sequences within a sequence of size p. This may represent a relationship between σ_1 and σ_2. The reactive formulae could also be restricted just to contain the same atom in each disjunct, which may be more interesting.

5.6.1 Application of Properties

We now show how the properties are related to each other and extend the discussion to include data mining applications. We note that the classification provided in Figure 5.4, given in (Manna and Pnueli, 1992), for temporal logic properties holds within our logic, exemplified by the axioms of Section 5.5.5. We make use of this *hierarchy* to discover rules in an incremental fashion.

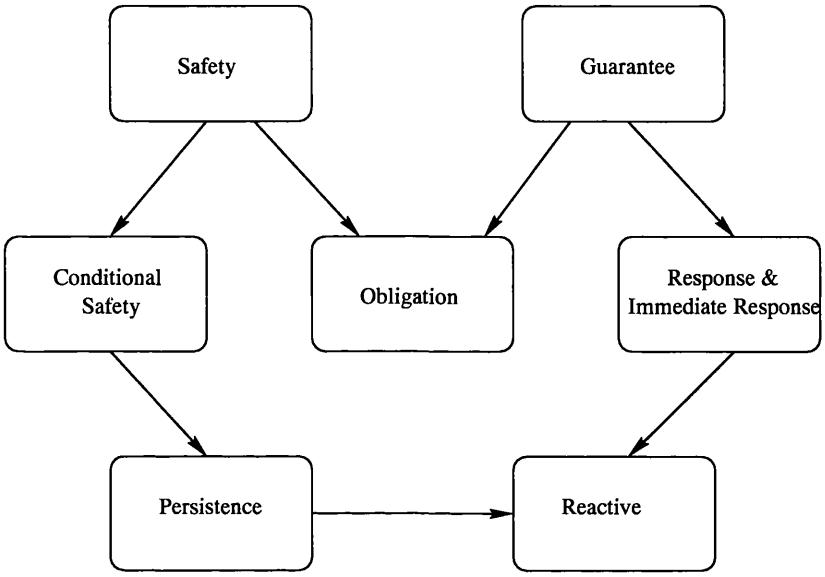

Figure 5.4: A Classification of Temporal Properties

The classification of temporal properties aids the process of knowledge discovery as we move concurrently from smaller to larger sequences and from obtaining different properties according to their classification in the hierarchy. The structure of properties, which we believe to contain a good classification of interesting patterns, therefore simplifies the data mining process. The value of these properties in program verification translates directly to knowledge discovery. We discuss this further in Chapter 6.

5.7 Discussion

Our logic conveys information within time series without the need to compare or analyse specific values, unlike a standard statistical analysis. Moreover, on top of this analysis we seek to discover properties, derived from those used within program verification, which the time series satisfies. Representation in our symbolic logic itself enhances the overall knowledge discovery process. (Halpern and Rabin, 1983) present a belief logic for reasoning about the likelihood of events with a modal operator $\mathcal{L}$ to state that an event is likely to happen. The application of this logic is as a decision support tool where, the authors claim, a statement "it is likely that s" is more useful than saying "s will occur with probability 0.63." Numbers may be attached to the modal operator $\mathcal{L}$ to

provide a degree of likelihood. Similarly, our logic may be viewed as a decision support tool when we state that, for example, in all monthly periods the trends in a particular bank lead the trends in its insurance subsidiary by one day. This formula expressed succinctly will directly aid any decision support process. Also, symbolic rules provide more information support than a graphical analysis as well as expressing data in a machine understandable form which is transferable to other knowledge discovery tools.

Further work is required to study the expressive nature of our logic which may lead to enhancements within the data mining process for knowledge discovery. Two mechanisms suggest themselves: (1) to include additional temporal operators within sequences, and (2) to extend the time series functionality of the logic at the atomic (and possibly at the connective) level. The latter could easily be extended to use techniques from related research, such as the discrete Fourier transform (Agrawal et al., 1995; Das et al., 1997; Rafiei, 1999), upon which we then search for properties.

We examine how our logic for NDs can be applied for inferring additional dependencies within a temporal relation sequence. If this logic were restricted further to FDs alone then the axioms of reflexivity, augmentation and transitivity are invariant within this language. For example, using $\Rightarrow$ to denote *implies*, we have, with respect to a sequence, $\boxminus^n X \rightarrow Y \Rightarrow \boxminus^n XZ \rightarrow YZ$ where $X, Y, Z \subseteq R$. Inference within the language is also similar to standard FD inference:

$$\boxminus^n(A \rightarrow B) \wedge \diamondminus^n(B \rightarrow C) \Rightarrow \diamondminus^n(A \rightarrow C)$$

or

$$\boxminus^n(A \rightarrow B) \wedge \boxminus^n((A \rightarrow C) \vee (B \rightarrow C)) \Rightarrow \boxminus^n((A \rightarrow C))$$

A logic with FDs as atomic formulae provides much more flexibility that incorporating Boolean dependencies into the model (Demetrovics et al., 1993) given that we have the enhanced expressiveness of temporal operators. This is a possible avenue for further research.

Within a temporal query language we propose that it would be valuable to be able to form queries asking if certain properties are satisfied over a given time period. These properties could be defined with respect to our, or another, logic. They also need not exactly conform to ND sets. Their use would be widespread both in data mining and for integrity analysis. For example, in a financial data mining system a user might want to ask if a persistence property holds in some yearly period for all monthly segments. A positive result might imply some seasonal behaviour that would not have been directly deducible from a visual analysis.

Schema evolution (Roddick, 1994) is a research area on the fringes of temporal data mining. Schema evolution is defined as the ability for a database schema to evolve without the loss of existing information. Mining the changes in schemas for patterns is a valid temporal data mining research area, not yet well developed. (van Bommel, 1993) presents an interesting discussion of evolutionary schema mutation and the converse of these ideas can be adopted for data mining. The methodology presented herein could mine different segments of temporal databases for properties; a comparison for properties found could then be conducted on relation sequences having different schemas. For example, an attribute *sterling_value* may be updated to *ecu_value*; this may or may not affect the general form of knowledge discovered.

CHAPTER 6

Temporal Property Detection with Numerical Dependencies and Resampling

We now present results of our temporal logic for knowledge discovery from NDs in temporal sequences applied to real world data. In Section 6.1 we discuss the context of our experiments and present the model for the discovery of properties in Section 6.2. The model we provide, given the flexibility of our logic, is not rigid and numerous algorithms may be created to extend or diverge from this model. We present one algorithm in this context, noting that other algorithms are direct implementations of the semantics provided by the logic. The results of our experiments are presented in two sections. Firstly, in 6.3 we present results gained from temporal relation sequences satisfying NDs. Then, in Section 6.4 we discuss results from experiments on standard time series data obtained from financial stocks. We present an analysis of our work in the context of this research area in Sections 6.5 and 6.6, as two case studies, relating these results to behaviour in the real-world in Section 6.6.1. In section 6.7 we introduce the moving blocks bootstrap for large relations and provide a critical study of our methodology in 6.8. We compare our work with other work conducted on time series similarity in Section 6.9. We conclude in Section 6.10.

6.1 Introduction

The flexibility of the logic implies that the knowledge discovery process requires restriction of the types of rules found to prevent trivial rules being discovered; we therefore focus on the discovery of properties, defined in Section 5.6, for pairs of temporal datasets. The discovery of properties used within program verification has not previously been applied to knowledge discovery. Obviously our work is closely related to other work on rule discovery though our logic allows for temporal relationships to be discovered. As we have seen in 2.3.2, (Berger and Tuzhilin, 1998) is a recent work which uses temporal logic for rule discovery; the logic used is a standard temporal logic and as such requires restriction for interesting patterns to be found. The use of properties

places such a restriction on patterns to be discovered whilst at the same time ensures the discovery of interesting properties.

Our property discovery model incorporates aspects of the property classification hierarchy thereby simplifying the knowledge discovery process. We move from obtaining values of statistical functions at the sequence level to the creation of safety and guarantee rules and then to a larger sequence size for the discovery of more complex properties, such as response and persistence. Within the knowledge discovery process we employ moving blocks resampling to discover short range properties. The moving blocks bootstrap considers all possible blocks of a given size n within an input time series. A resampled time series is then formed by randomly selecting blocks from the original series and appending each block to the resampled series until the resample is equal to or greater than the length of the original series. Different time series are sampled from simultaneously so that relationships between series are preserved within blocks. Property discovery may then be applied to this resampled sequence knowing that relationships have only been preserved within blocks. We show that useful conclusions can be found from this process, particularly in conjunction with property discovery from the original process. Our data mining model is no different from typical data mining systems which, as (Mannila, 1996) states, have modest aims in terms of the complexity of the knowledge obtained.

We applied our property discovery model to a number of different data sets including National Football League (NFL) data over 3 seasons and data sets of US National Notifiable disease data, both of which we could mine for ND set satisfaction. Restricting our logic solely for time series we then applied our methods to stocks from the FTSE 100. We found rules which complement the graphical depiction of a time series. Because we are referring explicitly to time series and not NDs in a temporal relation sequence we do not have specific attributes within which to refer to trends, or similar, so we use placeholders, in this case bp and sh. To illustrate, we found the following property in two oil stocks, BP (bp) and SHELL (sh), represented as $\Delta_{oil} \models^{3} \Diamond^{30}\boxminus^{15} (bp \downarrow_r v_1 \wedge^{0} sh \downarrow_r v_2)$ where $\downarrow_r$ implies a downward regressive trend and v_1, v_2 are the initial values of the stock when the rule holds (found at 7 locations over 242 days). Graphical analysis of these stocks (in Figure 6.9) would suggest a definite relationship but any general trends are obscure. This result for these sequence sizes suggests downward trends have lasted longer than upward from 1 December 1997 to 1 November 1998 for these stocks. Many additional results showed that interesting and unexpected properties were discovered, which we detail and analyse, that both complement and extend a graphical depiction.

For the sake of clarity we often present rules without specific values which we believe would

not aid in the presentation or understanding of the rule. The analysis of data mining methods is both an empirical and theoretical science. Measures are used for the latter whilst expert analysis is incorporated into the former. The results we find here, given that they are expressed in a logical form, are assessed empirically. The use of standard statistical functions within our logic implies that all results are statistically sound.

6.2 Property Discovery Model

We may validate our property discovery model of Figure 6.1 as follows. Our model is a natural generalisation of the upward formation of property discovery based on the formalisation of our logic. The goal is the discovery of properties within the framework of the temporal logic of sequences described in Section 5.5. A standard time series analysis would examine series for potential correlations, cross-correlations and similar functions (Kendall and Ord, 1990). The functions would take as input either the original time series, or a moving average time series to allow smoothing or a differenced time series for trend removal or a combination of these. The first part of our model incorporates this behaviour linearly by creating moving averages before searching for correlations. We also note that we may also create resampled sequences upon which we apply moving average and differencing techniques. After this initial step we seek to obtain reliable trends for sequence description.

At this stage we then have expressions representing the temporal relation sequence. These expressions of our logic do not contain any of the modal operators. Firstly, we may obtain a complete sequence description by applying $\rightsquigarrow$ to the non-modal expressions for a specific sequence size n. This may optionally include the correlations between NDs in the given input template of FDs F.

The final section of the property discovery model seeks to discover properties of our logic containing the $\boxminus^n$ and $\diamondminus^n$ modal operators using the classification hierarchy of Figure 5.4. Indicated in Figure 6.1 by the upward arrows from response to guarantee properties and from persistence to safety properties is the potential for recursive property discovery; such as a safety property containing persistence rules. We could not attempt to discover properties without first having expressions, similarly we do not wish to discover expressions without first applying moving average, differencing, and/or resampling techniques; it would not make sense to, say, create the moving average of a trend expression after we had broken it up into sequences due to extra repeated computation and, perhaps, different results due to increased *end effects*.

We now step through the property discovery model. Input is a sequence of n relation states. Each relation in this sequence satisfies a set of NDs. We wish to provide details of *properties*

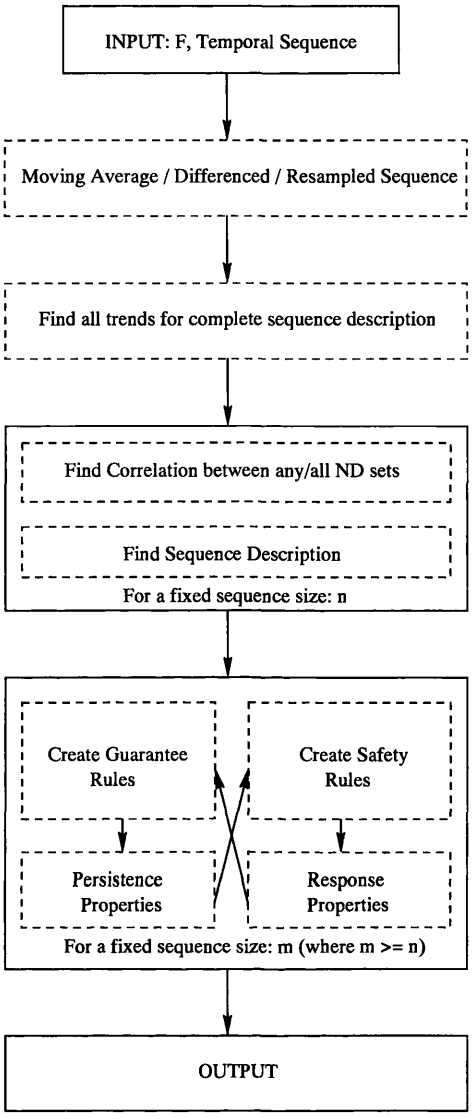

Figure 6.1: A description of our Temporal Property Discovery System

which may hold in the sequence. From the initial relation sequence we may form series of moving averages of windows, each of size w, so that the sequences we in effect deal with are moving average sequences, each of size $n - (w - 1)$, given the original relation sequence is of size n, or we can simply use the original relation sequence for trend detection. Differenced lists can also be created for seasonal property detection. We also have the option of employing *jackknife* resampling (Efron, 1982), for smoothing, at this point so that the sequences are robust i.e. noisy outliers are weakened by the use of resampling. We consider examples both with and without jackknife resampling. We can also apply the moving blocks bootstrap to recreate time series for short range property detection.

Using this information we then can gain trend, cross- and auto-correlation, and sequence description information. We examine the sequence for correlation and sequence description pur-

poses all sequences of a fixed size n. This allows us to find safety and guarantee properties with regard to n. We also obtain an input for a larger sequence size m so that complex properties, such as response properties, are detected with respect to m and n. Additional flexibility is therefore achieved by looking for patterns of n time points within larger sequences m time points. Property discovery occurs in a bottom-up fashion whilst querying, if enabled, would occur top-down.

If all sequences of a fixed size satisfy a rule we refer to this as a *safety property*, whereas there may exist a set of sequences such that a property holds throughout the complete sequence but not for subsequences of a fixed size, which we denote as a *cover*, which may imply irregular behaviour. As we have shown there are an exponential number of such covers and we do not attempt to discover these. If any of these properties occur not for the complete sequence but for a complete subsequence we denote this by creating persistent properties. Figure 6.1 also shows that properties may themselves contain properties, such as a safety property for persistence rules. This would then require three sequence sizes to be given by the user or for incremental steps in sequence size to be performed within the discovery process. We limit ourselves to two sequence sizes.

6.2.1 The Generic Property Discovery Algorithm

In the data mining literature there has been much discussion of working towards a common framework for data mining, presenting comparisons of data mining now to database research in the 60s before the adoption of the relational model (Fayyad et al., 1996c; Mannila, 1996). Generic algorithms for data mining have been proposed, most notably by (Mannila, 1996), extended in (Mannila, 1997). We now outline this generic procedure. A candidate set of initial patterns is provided by the user. The database (or data set) is then examined to see if these patterns occur a sufficiently frequent number of times, in which case they are classified as interesting. A new candidate set is generated from the interesting patterns and the previous candidate set and the process is repeated. This is continued until there are no new candidate elements and the interesting set is returned as knowledge discovered. We can see that our algorithm 14 has a similar skeleton to this generic procedure. Our procedure is general in that we consider the satisfaction of a property to be interesting and the natural classification of properties allows properties to be discovered using the input relation sequence and the properties previously discovered. Using Figure 5.4 as a basis, property set p_1 is higher than set p_2 if there does not exist a property in p_2 which is formed from a property in p_1. We also assume that within sets p_1 or p_2 no properties are formed from any other properties in the set.

Algorithm 14 (Property_Mine(Δ, F))
1. **begin**
2. Rule_set := $\emptyset$;
3. **while** $\exists$ a new classification of properties **do**
4. **for each** property p at same classification c **do**
5. Rule_set := $\{q \mid p$ property rule q discovered from Δ, F, and Rule_set $\} \cup$ Rule_set;
6. **end for**
7. c := Next classification of temporal properties
8. **end while**
9. **return** Rule_set;
10. **end.**

Figure 6.2: The Generic Property Data Mining Algorithm

6.2.2 The Response Persistence Algorithm

In Algorithm 15, detailed in Figure 6.3, we present a simple algorithm for detecting response and persistence properties with respect to two sequence sizes given by the user. This may be considered as a direct specialisation of Algorithm 14. For this algorithm the classification is {{ Safety, Guarantee }, { Response, Persistence }}. Algorithm 15 accepts a temporal relation sequence Δ and a set of FDs F, which we assume are satisfied as NDs, together with lower and upper sequence sizes. The algorithm works in a bottom up fashion such that all formulae which may hold are classified into sets of formulae for each subsequence. Membership of formulae in any or all of these sets then determines if a rule in a higher classification is satisfied.

6.3 Relational Sequence Data Sets

We now discuss the experiments carried out and the results achieved using NDLTL. Given the flexibility of our logic it is easy to extend the results presented here by:

- Allowing the user to query a given input. He may want to know, using sales data obtained daily over 2 years, if there is a peak of sales in every quarter, and express this using our logic. This example shows a possible seasonality query which would take the form $\boxminus^{730}\diamondsuit^{90}$ (X $\rightarrow^{\uparrow_r K}$ Y $\rightsquigarrow$ X $\rightarrow^{\downarrow_r K}$ Y)

- Modifying the time series functions within the logic. Different functions can be incorporated, for example, that are specifically known to handle nonlinear time series better than linear regression or discordance.

We now present the results, initially focusing on ND temporal relation sequences and then moving on to time series results alone. We focus on the latter due to the lack of significant real-world data available for temporal data (it is easier to obtain public data, such as share closing

Algorithm 15 (Response_Persistence(Δ, F, n, m))
1. **begin**
2. Main_Rule_set := $\emptyset$;
3. Final_Rule_set := $\emptyset$;
4. **for each** subsequence s of Δ of size m **do**
5. Rule_set := $\emptyset$;
6. **for each** subsequence s_n of s of size n **do**
7. Rule_set$_{s_n}$:= Rule set discovered for s_n wrt F;
8. Rule_set := { Rule_set$_{s_n}$ } $\cup$ Rule_set;
9. **end for**
10. M_rule := $\emptyset$;
11. **if** $\forall\, r \in$ Rule_set $\exists \sigma$ such that $\sigma \in r$ **then**
12. M_rule := $\{\boxminus^n \sigma\} \cup$ M_rule;
13. **end if**;
14. **if** $\exists\, r \in$ Rule_set and $\exists\, r_2 \in$ Rule_set with $r \neq r_2$ such that $\sigma \in r$ and $\sigma \notin r_2$ **then**
15. M_rule := $\{ \diamondminus^n \sigma\} \cup$ M_rule;
16. **end if**;
17. Main_Rule_set := { M_rule } $\cup$ Main_Rule_set;
18. **end for**
19. **if** $\forall\, r \in$ Main_Rule_set $\exists \sigma$ such that $\sigma \in r$ **then**
20. Final_Rule_set := $\{\boxminus^m \sigma\} \cup$ Final_Rule_set;
21. **end if**;
22. **if** $\exists\, r \in$ Main_Rule_set and $\exists\, r_2 \in$ Main_Rule_set with $r \neq r_2$ such that $\sigma \in r$ and $\sigma \notin r_2$ **then**
23. Final_Rule_set := $\{ \diamondminus^m \sigma\} \cup$ Final_Rule_set;
24. **end if**;
25. **return** Final_Rule_set;
26. **end.**

Figure 6.3: The Response Persistence Algorithm

prices, compared to a database of employee data over the last 20 years). We also concentrate on time series due to the availability of data with a significant number of points, whereas a temporal database may only be updated monthly/yearly, though this is changing for many automated knowledge discovery and data warehousing applications.

6.3.1 Results

We present two datasets used to obtain ND values, both publicly available at Statlib, a data set resource (`http://lib.stat.cmu.edu`). The experimental methodology used in these simulations is fully discussed in Appendix B. The first we discuss are (blind) records of disease data concerning occurrences of mumps in the US from 1957 to 1989. These records contain the number of patients for cases of mumps reported on a state-by-state basis. Though we obtained from the dataset the number of patients per year suffering from mumps we note that this may have been expressed in a database from where a relation was used for storing patient data in the form

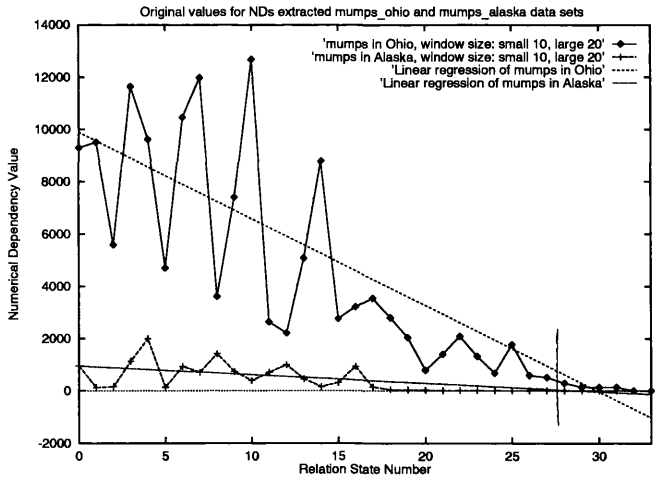

Figure 6.4: Original data values of mumps cases in Ohio and Alaska from 1957 - 1989

of $\emptyset \rightarrow^{k} PATIENT\_ID$. We note that when the left hand side of an ND is empty the branching factor of the ND is a cardinality constraint on the domain size of the right hand side attribute set. This is a small data set, referred to as Δ_{mp}, and we can see that it has a clear downward trend in Figure 6.4. We use it for illustration before moving on to more complex data sets. For the sake of clarity we express both ND values such as $\emptyset \rightarrow^{\uparrow k} PATIENT\_ID$ simply as a marker of trend preceded by an identifier if the trends are for different NDs or time series, e.g. $ohio \uparrow$. The specific ND values are not important in this context particularly as they are most probably related to population size which would need to be normalised.

The results obtained for a small sequence size of 5 years and a large size of 10 years were $\Delta_{mp} \models \boxminus^{10} \diamondminus^{5} (ohio \downarrow_r \wedge^{0} alaska \downarrow_r)$ together with persistence results of the following abbreviated form $\diamondminus^{10} \boxminus^{5} (ohio \downarrow_r \wedge^{0} alaska \downarrow_r)$. Clearly, these results tell us that the number of cases in mumps is falling, continuously, without significant fluctuation. Applying a simple pattern matching algorithm for comparing exact trend within a sequence we find that $\boxminus^{12} (ohio \downarrow \rightsquigarrow ohio \uparrow \rightsquigarrow ohio \downarrow)$ holds for Ohio. Therefore although we have found the general trends to be downward there are peaks within larger sequences. For a comparison between the number of cases in Alaska and Ohio we increased the sequence sizes to 12 and 20 and found, continuously throughout the sequence that the following persistence rule holds: $\diamondminus^{20} \boxminus^{12} (ohio \downarrow_r \wedge^{2} alaska \downarrow_r)$. We see that the lag in the downward trends (ohio lags alaska by 2 years) may provide an indication that the number of cases falling is correlated with geographical regions. Obviously expert knowledge is required to confirm this; (Fayyad, 1998b) discusses issues of correlation versus causality noting that it is generally not clear in which situations correlation can lead to causality. It may be foolish to infer a causal relationship solely on the basis of the discovery of a lagged correlation. This is

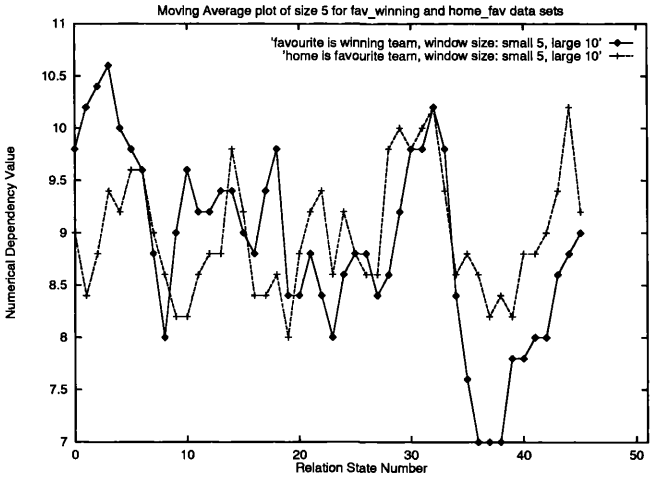

Figure 6.5: Moving Average data set values of two NDs from NFL season data 1989-1991

the subject of much study with further references provided in (Glymour et al., 1997).

A complete description of the series is provided by $(ohio \downarrow_r \wedge^1 alaska \downarrow_r) \rightsquigarrow (ohio \downarrow_r \wedge^2 alaska \downarrow_r)$. We find the same rule from applying discordance instead of linear regression. This rule concisely presents the behaviour of the sequence. It is obtained from three 12 year sequences with some overlap (there is 33 years of data) and compressed, for clarity, so that $\sigma \rightsquigarrow \sigma$ becomes σ. We omit presentation of such description rules for longer time series. We can see from these simple results how properties of NDs over time can be succinctly characterised within our logic. We now move on to a slightly more interesting example.

In Figure 6.5 we present the changing ND values for two NDs obtained from relations containing Football Data. Each week of the season, for three seasons from 1989 to 1991, details of team results were stored in a database together with details of the favourite team for each match. We obtained two NDs from the relation $YEARWEEK \rightarrow^k FAV\_TEAM\_WIN$ and $YEARWEEK \rightarrow^k HOME\_FAV$. These NDs correspond to the number of favoured teams winning and the number of home favourites, respectively, within a particular week. We note that each relation contained $YEARWEEK\ DAY$ representing the day the match was played on. Figure 6.5 shows the two lines relating to changes in each ND. We can see no clear trend in this figure for moving averaged data.

For the two NDs we found the following property from the moving average record of points with each moving average of size 5, $\Delta_{NFL} \models \Diamond^{10}\Box^5 (HOME\_FAV \uparrow_r \wedge^0 FAV\_TEAM\_WIN \uparrow_r)$ suggesting that an increase in the home team winning is correlated with an increase in the favourite team winning. This persistence property expresses the fact concisely and we can perhaps infer from this that the home teams are most often the favourite

team. Additionally, we examined the original and differenced data set for patterns in sequences of 5 weeks and found none. It is clear that the nature of the data contains no underlying trend expressing only the strong correlation between NDs.

Before presenting the results obtained from time series data we formally define the moving blocks bootstrap.

6.3.2 The Moving Blocks Bootstrap

We introduce the Moving Blocks Bootstrap for verification of short range event rules and provide details of its efficacy discussing the results we found from its application in later sections.

Definition 6.3.1 (Moving Blocks Bootstrap for Relation Sequences) Given a relation sequence $\{r_1, r_2, \ldots, r_N\}$ we construct blocks of relations where MB_t is a block containing b relations such that $MB_t = \{r_t, r_{t+1}, \ldots, r_{t+b-1}\}$ and there are $N - b + 1$ blocks where $t = 1, 2, \ldots, N - b + 1$. We then resample k moving blocks uniformly with replacement from $\{MB_1, MB_2, \ldots, MB_{N-b+1}\}$ where $N \sim bk$. This may be repeated any number of times. □

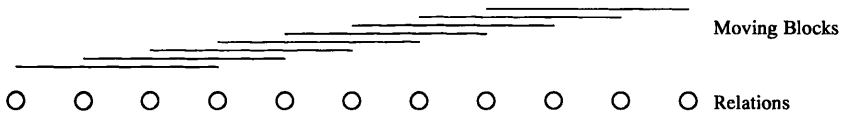

Figure 6.6: All possible blocks of size 4 for a relation sequence

The moving blocks bootstrap forms an empirical distribution and this distribution is the proposed bootstrap approximation. For a block length b all possible contiguous blocks of length b within the time series are available for selection. In (Efron and Tibshirani, 1993) each moving blocks bootstrap sample has an AR(1) model fitted to it to estimate the parameter β. Results decreased upon an increase in the block size, perhaps allowing us to infer that dependency was based on the previous few points only (to any significant degree). The potential to sample all possible contiguous blocks of a given size b allows all possible relationships of length less than or equal to b to be sampled.

6.3.3 The Moving Blocks Bootstrap for Large Data Sets

We now propose a new Moving Blocks Bootstrap for creating resamples when either the temporal relation sequence or the time series contain too many points for a resample to obtain meaningful results. For example this may be a stock over 10 years. We may form a series of values, which are fixed in ordering, so that this 10 year sequence may be compressed into resamples of, say, 2 years each. This will then create a manageable sequence size for property discovery across multiple resampled sequences.

Definition 6.3.2 (Moving Blocks Bootstrap for Large Relation Sequences) Given a relation sequence $\{r_1, r_2, \ldots, r_N\}$ we construct a resampled relation sequence of size n, where $N \gg n$, with the order of the resampled relation sequence preserved from the original data set. We construct blocks of relations where MB_t is a block containing b relations such that $MB_t = \{r_t, r_{t+1}, \ldots, r_{t+b-1}\}$ and there are $N - b + 1$ blocks where $t = 1, 2, \ldots, N - b + 1$. We then divide the sequence into R regions where $R = \frac{N}{n}$. From each region $R_i, 0 \leq i \leq R-1$ we resample 1 moving block uniformly from $\{MB_{(i*n)+1}, MB_{(i*n)+2}, \ldots, MB_{(i*n)+n-b+1}\}$ and append the block to our resampled sequence. Order is therefore preserved for n, a given size. This may be repeated any number of times. □

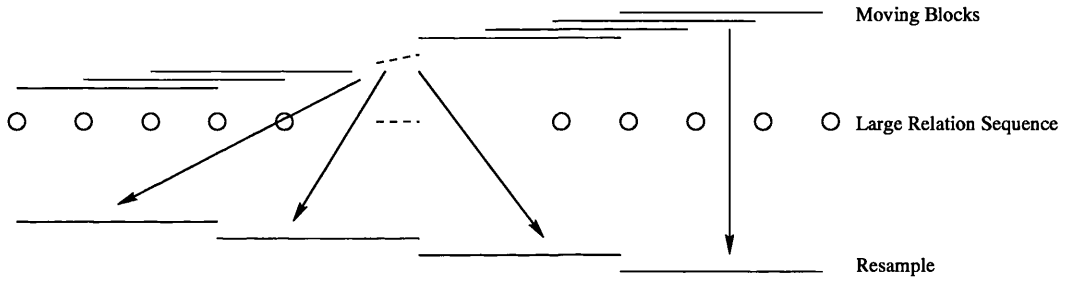

Figure 6.7: A large relation sequence and a resample

Given that the random block selection for the moving blocks bootstrap for large relations is order preserving we must be careful in our random block selection to ensure that we do not initially select a block at the end of the sequence to which no blocks can be appended. To remedy this we propose that the sequence is divided into a number of regions n, from Definition 6.3.2. From these regions we select one block randomly. For example, a 10 year sequence may be divided into 24 regions, of 5 months each, from which a block of one month is selected from each. This would then allow reasonable knowledge discovery on a manageable sequence with relationships between either NDs or time series preserved within blocks and the ordering of the sequence is preserved in the resamples. Alternatively, we could use standard moving blocks resampling with smaller sample sizes and sorting each resample into the correct temporal order; this would create more randomness in the resamples.

6.4 Time Series Data Results

We now examine more complex temporal data sets. We assume that the data is restricted to numerical data alone, although if the data were stored in a suitable manner then NDs could easily apply. When analysing time series we are not seeking to discover what a complex statistical analysis could not; indeed much of our logic is based on statistical functions. Instead we are looking

for a concise representation that might convey specific properties which hold at a certain time or throughout time and we express these with our logic. The logic details properties in a machine understandable form.

For the following study we focused on financial stocks from the FTSE 100. We analysed stocks in similar sectors seeing if there are general properties from which we can infer information, concentrating on financial, oil and retail sectors. We highlight some of the results found and remark that all applications discovered possibly useful properties. In the following we initially present results based on each category of data, discussing the results with respect to the sequence size selected; we follow this with a discussion of how these properties might relate to the data they represent in the real world. A summary of results is presented for each data set in Tables 6.1, 6.2, and 6.3. As a precursor to this discussion we remark that the absence of a property with respect to sequence sizes chosen by the user, when evaluated with results for other sequence sizes, may itself tell us much about the data set.

6.5 Case Study I

We present results for Debenhams (db) and the Arcadia Group (ag), summarised in Table 6.1, in an extended form which highlight characterisation of the discovery of property. Subsequent studies are abbreviated to avoid repetition and we refer to this sequence as Δ_{ad}. Due to the similarity between regression and discordance results, we concentrate on results obtained using linear regression. We note that we are now dealing exclusively with time series and not ND values; this is primarily due to the availability of time series data and, as we have seen, the limited availability of data satisfying ND sets. We note that all results are equivalent to those that may be found for ND sets in a temporal relation sequence and could feasibly be expressed with NDs, though this may often be impractical.

6.5.1 Original Data Analysis

Properties extracted from the original data set are based solely on the original data values upon which our property discovery algorithms are applied. We first discuss looking for trends as properties. We see in Table 6.1 that we obtained safety trends, for a sequence size 80, $\boxminus^{80}\diamondsuit^{1}$ $(ag \downarrow)$ for the Arcadia Group and $\boxminus^{80}\diamondsuit^{1}$ $(db \uparrow)$ for Debenhams. This states that in every 80 day sequence there is at least one day when the stock goes up or down! This may seem obvious but these two results together suggest that Debenhams might be performing better than Arcadia. The procedure to obtain these naive trend rules uses a simple pattern matching algorithm. We also found persistence rules of the form $\diamondsuit^{20}\boxminus^{10}$ $(ag \uparrow_r \wedge^0 db \uparrow_r)$ and $\diamondsuit^{20}\boxminus^{10}$ $(ag \downarrow_r \wedge^0 db \downarrow_r)$. The former persistence rule was discovered at five points in the complete sequence and the latter was

Debenhams and Arcadia Group	
Description of data set	199 days of closing prices
In all formulae: Arcadia Group (ag) and Debenhams (db)	
Trend Discovery	
Original Data Set	$\Delta_{ad} \models \boxminus^{80}\Diamond^{1}\,(ag\downarrow)$ $\Delta_{ad} \models \boxminus^{80}\Diamond^{1}\,(db\uparrow)$
Property Discovery	
Original Data Set	$\Delta_{ad} \models \boxminus^{160}\,(ag\downarrow_r \wedge^0 db\downarrow_r) \rightsquigarrow (ag\downarrow_r \wedge^0 db\downarrow_r)$ $\Delta_{ad} \models \Diamond^{10}\boxminus^{5}\,(ag\uparrow_r \wedge^0 db\uparrow_r)$ $\Delta_{ad} \models \Diamond^{20}\boxminus^{10}\,(ag\uparrow_r \wedge^0 db\uparrow_r)$ $\Delta_{ad} \models \Diamond^{20}\boxminus^{10}\,(ag\downarrow_r \wedge^0 db\downarrow_r)$ $\Delta_{ad} \models \Diamond^{30}\boxminus^{15}\,(ag\downarrow_r \wedge^0 db\downarrow_r)$ $\Delta_{ad} \models \Diamond^{40}\boxminus^{20}\,(ag\downarrow_r \wedge^0 db\downarrow_r)$ $\Delta_{ad} \models \Diamond^{80}\boxminus^{40}\,(ag\downarrow_r \wedge^0 db\downarrow_r)$ $\Delta_{ad} \models \boxminus^{160}\Diamond^{80}\,(ag\downarrow_r \wedge^0 \downarrow_r)$ $\Delta_{ad} \models \boxminus^{160}\Diamond^{80}\,(ag\uparrow_r \wedge^1 db\downarrow_r)$
Moving Average Window size:3 Window size:8	 $\Delta_{ad} \models^3 \Diamond^{20}\boxminus^{10}\,(ag\downarrow_r \wedge^0 db\downarrow_r)$, found 12 times $\Delta_{ad} \models^8 \Diamond^{80}\boxminus^{40}\,(ag\downarrow_r \wedge^0 db\downarrow_r)$
Moving Block Bootstrap Block Size: 5 Block Size: 10 All for MA Block Size:3	 $\Delta_{ad} \models^3 \Diamond^{20}\boxminus^{10}\,(ag\downarrow_r \wedge^0 db\downarrow_r)$ $\Delta_{ad} \models^3 \boxminus^{40}\Diamond^{20}\,(ag\downarrow_r \wedge^0 db\downarrow_r)$ $\Delta_{ad} \models^3 \boxminus^{80}\Diamond^{20}\,(ag\downarrow_r \wedge^0 \downarrow_r)$ $\Delta_{ad} \models^3 \boxminus^{80}\Diamond^{20}\,(ag\uparrow_r \wedge^0 db\uparrow_r)$
Differenced Series	 $\Delta_{ad} \models \boxminus^{80}\Diamond^{20}\,(ag\downarrow_r \wedge^0 db\downarrow_r)$ $\Delta_{ad} \models \boxminus^{80}\Diamond^{20}\,(ag\uparrow_r \wedge^0 db\uparrow_r)$
2nd Order Differenced Series	 $\Delta_{ad} \models \boxminus^{80}\Diamond^{20}\,(ag\uparrow_r \wedge^0 db\downarrow_r)$ $\Delta_{ad} \models \boxminus^{80}\Diamond^{20}\,(ag\downarrow_r \wedge^0 db\uparrow_r)$

Table 6.1: Results for 199 days of Arcadia and Debenhams Group

found to hold at the very beginning of the series. This corresponds with the actual values which are generally within a downward trend apart from an initial upward trend depicted by their moving averages in Figure 6.10. Increasing the sequence size shows that persistence rules for upward trends do not hold at all for larger sequence sizes and so we conclude that upward trends within the series are short and that the general trend is down, exemplified by $\Diamond^{80}\boxminus^{40}\,(ag\downarrow_r \wedge^0 db\downarrow_r)$. Similarly, the safety trend descriptions indicate that Debenhams might be performing better that Arcadia.

6.5.2 Moving Average Analysis

The goal of creating a moving average of a time series is to smooth the series so that outliers, potentially caused by noise or outside effects, have a weakened influence on the data discovery process. They are widely used in stock data analysis (Rafiei and Mendelzon, 1997). We con-

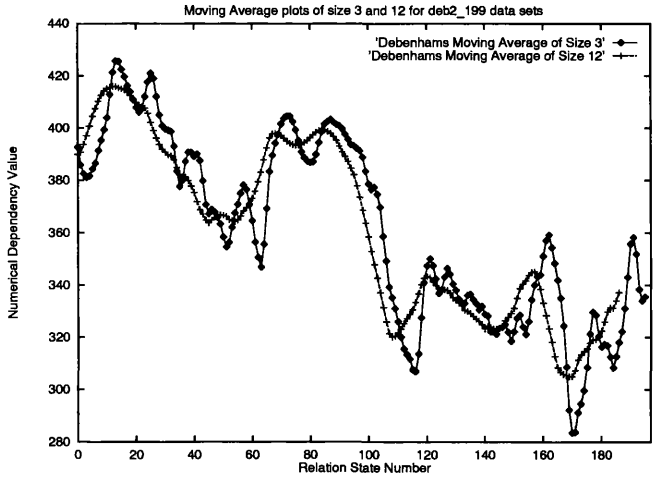

Figure 6.8: Moving Average Data values for two window sizes, 3 and 12

ducted experiments with a jackknife procedure for moving average smoothing, whereby the moving averages for each sequence are calculated successively with one data point removed and then averaged. We obtained a slight increase in smoothing over the standard moving average but not enough to justify the additional computational cost in its use and so this was not employed.

Moving average results in general back up the results provided by the original data set. We note that smoothing tends to obscure short term trends so that persistence or guarantee rules which hold in the original data set may not hold for a moving average series. Additionally, the size of the window for moving averages increases the *spreading* effect of a single data point. To illustrate we found far fewer properties for larger moving average windows. Figure 6.8 shows how a larger window size increases the spread for the moving averages, reducing the amplitude of both peaks and troughs.

6.5.3 Differenced List Analysis

In Time Series Analysis differencing is used by statisticians to remove trend from a series, as we outlined in Section 5.4. Similarly we may obtain the differenced values for a data set upon which we run our property detection algorithms. This may lead to the discovery of properties which then represent seasonal and not trend behaviour. We refer to Table 6.1 which present some results for differenced lists. Due to the complex nature of stock behaviour we can not be sure if the properties for differenced lists detail seasonal or just noisy behaviour. The result of a first differencing provides very similar results to the original and moving average properties. Namely, that the behaviour of the two stocks is closely related with response and persistence properties detecting either joint upward or downward trends. Extending this to a second order differencing we find that this shared behaviour is no longer discovered. What can we infer from this? For

a conclusive answer we would have to ask a fund manager, however, we note that perhaps their seasonal behaviour is not related or that the stocks are affected by different events outside of their relationship. In such a way we can use our algorithms for the discovery of seasonal properties.

6.5.4 Moving Blocks Bootstrap Analysis

The moving blocks bootstrap, defined in Section 6.3.2, is used within time series analysis for model creation based on the assumption that temporal relationships do not occur for a time longer than the size of the blocks used to create the moving blocks resamples. The moving blocks resamples may then be recreated many times to obtain a model based on these resamples. We created moving blocks resamples of our time series. Interesting properties were found and though we believe some properties for some resamples to be spurious we could have removed these via repeated application of the moving blocks bootstrap and intersection of the results. Applying the moving blocks bootstrap also allows for properties to be discovered which may be violated in sequences without such a rearrangement, perhaps caused by noise.

The order of the moving blocks resamples is random. It is therefore highly likely that spurious trends may be found for the moving blocks. $\boxminus^{80}\diamondminus^{20}$ $(ag \downarrow \wedge^0 db \downarrow)$ and $\boxminus^{80}\diamondminus^{20}$ $(ag \uparrow \wedge^0 db \uparrow)$ were both found with a block size of 10 days. This implies, we believe, two things. Firstly, that the behaviour of the two stocks is closely related both sharing either upward or downward trends. Secondly, the difference between small and large sequence sizes is quite significant implying that it is likely that blocks will occur to create an upward trend shown in the second response rule. The data miner needs to choose sequence sizes carefully in such cases. For a block size 5 we found $\diamondminus^{20}\boxminus^{10}$ $(ag \downarrow \wedge^0 db \downarrow)$, which backs up both the original and moving average results. The variation of block size allows us to make conclusions about the nature of the trends. For 5 and 10 days we found properties with upward or downward trends suggesting that these stocks possess trend behaviour longer, in general, than these block sizes. This is, however, a feature of the financial market in general.

6.6 Case Study II

We present results for two oil stocks BP (bp) and Shell Oil (sh), summarised in Table 6.2, in an abbreviated form.

The original values again emphasise a strong relationship between the two data sets at smaller sequence sizes. We experimented with large sequence sizes and found $\boxminus^{180}\diamondminus^{90}$ $(bp \uparrow_r \wedge^0 sh \downarrow_r)$. Two points about this are worth noting. Firstly, the disparity in trend in not immediately clear from a graph, cf. Figure 6.9, which exhibits much similarity. The initial upward trend of BP stocks may be viewed as *masked* by a number of short term downward

BP and Shell	
Description of data set	242 days of closing prices
In all formulae: British Petroleum (bp) and Shell (sh)	
Trend Discovery	
Techniques	
BP	$\Delta_{oil} \models^{w} \boxminus^{15} \diamondminus^{1} (bp\downarrow)$
	$\Delta_{oil} \models \boxminus^{50} \diamondminus^{4} (bp\downarrow \rightsquigarrow bp\uparrow \rightsquigarrow bp\downarrow \rightsquigarrow bp\uparrow)$
Shell	$\Delta_{oil} \models \boxminus^{15} \diamondminus^{1} (sh\uparrow)$
	$\Delta_{oil} \models \boxminus^{50} \diamondminus^{3} (sh\uparrow \rightsquigarrow sh\downarrow \rightsquigarrow sh\uparrow)$
Property Discovery	
Original Data Set	$\Delta_{oil} \models \diamondminus^{30}\boxminus^{15} (bp\downarrow_r \wedge^{0} sh\downarrow_r)$, found 7 times
	$\Delta_{oil} \models \boxminus^{45}\diamondminus^{15} (bp\downarrow_r \wedge^{0} sh\downarrow_r)$
	$\Delta_{oil} \models \diamondminus^{60}\boxminus^{30} (bp\downarrow_r \wedge^{0} sh\downarrow_r)$
	$\Delta_{oil} \models \boxminus^{90}\diamondminus^{45} (bp\uparrow_r \wedge^{0} sh\uparrow_r)$
	$\Delta_{oil} \models \boxminus^{180}\diamondminus^{90} (bp\uparrow_r \wedge^{0} sh\downarrow_r)$
	$\Delta_{oil} \models \boxminus^{180}\diamondminus^{90} (bp\uparrow_r \wedge^{0} sh\uparrow_r)$
Moving Average	$\Delta_{oil} \models^{3} \boxminus^{180} (bp\downarrow \wedge^{0} sh\downarrow) \rightsquigarrow (bp\downarrow_r \wedge^{0} sh\downarrow_r)$
Block Size: 3	$\Delta_{oil} \models^{3} \diamondminus^{60}\boxminus^{30} (bp\uparrow_r \wedge^{0} sh\uparrow_r)$
	$\Delta_{oil} \models^{3} \diamondminus^{60}\boxminus^{30} (bp\downarrow_r \wedge^{0} sh\downarrow_r)$
Block Size: 8	$\Delta_{oil} \models^{8} \diamondminus^{30}\boxminus^{15} (bp\downarrow_r \wedge^{0} sh\downarrow_r)$, found 7 times
Block Size: 10	$\Delta_{oil} \models^{10} \diamondminus^{60}\boxminus^{30} (bp\uparrow_r \wedge^{1} sh\uparrow_r)$
Moving Block Bootstrap	
Block Size: 15	$\Delta_{oil} \models^{8} \boxminus^{60}\diamondminus^{30} (bp\downarrow_r \wedge^{0} sh\downarrow_r)$
Block Size: 25	$\Delta_{oil} \models^{8} \boxminus^{100}\diamondminus^{50} (bp\downarrow_r \wedge^{0} sh\downarrow_r)$
All for MA Block Size: 8	
Differenced	$\Delta_{oil} \models \boxminus^{60}\diamondminus^{30} (bp\downarrow_r \wedge^{0} sh\downarrow_r)$
2nd Order Differenced	$\Delta_{oil} \models \boxminus^{60}\diamondminus^{30} (bp\downarrow_r \wedge^{0} sh\uparrow_r)$
	$\Delta_{oil} \models \boxminus^{60}\diamondminus^{30} (bp\uparrow_r \wedge^{0} sh\downarrow_r)$
	$\Delta_{oil} \models \boxminus^{60}\diamondminus^{30} (bp\downarrow_r \wedge^{0} sh\downarrow_r)$

Table 6.2: Results for 242 days of BP and Shell from Dec 1997 to Oct 1998

trends which the properties suggest. Secondly, the data set consists of only 242 points (days). Yet we are looking for sequences of 180 days which implies that there is no sequence which does not overlap with another. This response rule is therefore perhaps not quite so strong though still interesting.

Finally, we briefly refer to Table 6.3. This consists of stock prices for two newly converted building societies in their first 100 days on the market. Though we found properties they do not present themselves as showing much similarity. We can infer that the behaviour of the market at the time of launch is itself more important that what the company is. Tests showed that different stocks over different time periods tend towards not discovering properties as opposed to discovering properties which represent disparate behaviour. Additionally, of the properties discovered we see a number of spurious lags which do not suggest strong related behaviour. This is a validation of our discovery process.

Halifax and A & L Banks	
Description of data set	100 days of closing prices
In all formulae: Alliance & Leicester (al) and Halifax (hfx)	
Trend Discovery	
A & L Halifax	$\Delta_{ha} \models \boxminus^{10} \diamondminus^{1} (al \uparrow)$ $\Delta_{ha} \models \boxminus^{10} \diamondminus^{1} (hfx \downarrow)$ $\Delta_{ha} \boxminus^{20} \diamondminus^{1} (hfx \uparrow)$
Property Discovery	
Original Data Set	$\Delta_{ha} \models \boxminus^{60} \diamondminus^{30} (al \downarrow_r \wedge^{6} hfx \downarrow_r)$ $\Delta_{ha} \models \boxminus^{60} \diamondminus^{30} (al \uparrow_r \wedge^{-3} hfx \downarrow_r)$
Moving Average Block size: 5	$\Delta_{ha} \models^{5} \diamondminus^{20} \boxminus^{10} (al \uparrow_r \wedge^{0} hfx \uparrow_r)$ $\Delta_{ha} \models^{5} \boxminus^{60} \diamondminus^{30} (al \uparrow_r \wedge^{-3} hfx \downarrow_r)$ $\Delta_{ha} \models^{5} \boxminus^{60} (al \downarrow_r \wedge^{6} hfx \downarrow_r)$
Moving Block Bootstrap Block size: 5 MA Block size:5	$\Delta_{ha} \models^{5} \diamondminus^{20} \boxminus^{10} (al \downarrow_r \wedge^{0} hfx \uparrow_r)$

Table 6.3: Results for first 100 days trading of Halifax and Alliance & Leicester Banks

6.6.1 Real-World Analysis

Our first analysis focuses on BP and Shell. Under a story entitled "bad times for the oil industry" in the Lex column of the Financial Times, November 4 1998, it was discussed how the oil industry has suffered in recent months though some recent results (third quarter) posted by BP show that a 35% drop in profits is good news in comparison with a more than 50% drop by Shell prices. We can see from Figure 6.9 that BP (bp) has been outperforming Shell (sh) in terms of recent performance. We discovered however that the stocks are related in short term performance, as we would expect. We found that in Jan and Feb the following persistence property held $\diamondminus^{90} \boxminus^{60} (bp \uparrow_r \wedge^{0} sh \uparrow_r)$, a period of gradual rise in both stocks. We also found that $\diamondminus^{60} \boxminus^{30} (bp \downarrow_r \wedge^{0} sh \downarrow_r)$ holds from day 160 in Figure 6.9, relating to the downward trend that begins in May. With a smaller sequence similar results were obtained though we also discovered that $\diamondminus^{10} \boxminus^{5} (bp \downarrow_r \wedge^{0} sh \uparrow_r)$ held in September 1998; this opposite behaviour may be due to external influences.

In Figure 6.10 we show the moving averaged sequence for two companies, Debenhams (db) and the Arcadia Group (ag) since January 28 1998. On January 28 1998 Debenhams demerged from its former owner the Arcadia Group. We can see from Figure 6.10 that recently Debenhams has performed better than Arcadia due to, based on expert opinion, the fact that Debenhams sells many different goods whereas Arcadia concentrates more on fashion and is expected to perform poorly in the light of a recession. In August 1998, corresponding with a downturn in the economy, we found $\diamondminus^{30} \boxminus^{15} (db \downarrow_r \wedge^{0} ag \downarrow_r)$ and for the recent good performance of Debenhams we found

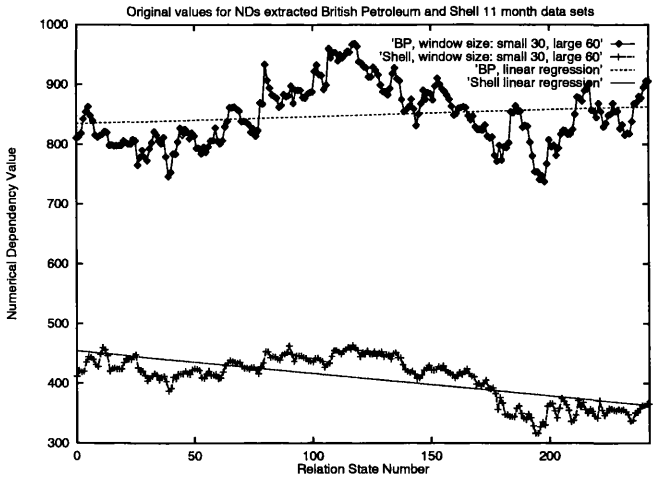

Figure 6.9: Time series of BP and Shell from 1 Dec. 1997 to 1 Nov. 1998

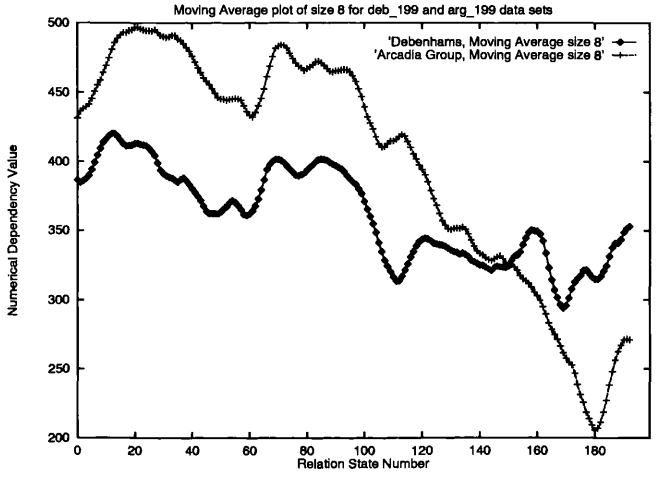

Figure 6.10: Moving Average values for Debenhams and Arcadia Group since demerger on Jan 28 1998

$\Diamond^{10}\boxminus^{5}\,(db\uparrow_r \wedge^0 ag\downarrow_r)$, amongst other rules. The regression coefficient used to determine trend may be significantly small. However the trend still exists and we can subscript trends by their regression value or even extend this to a *fuzzy* value to denote the significance of the trend.

It is imperative that the two sequence sizes are well chosen by the user. It would help if the user had expert knowledge of any kind of seasonality duration. If $n > \frac{m}{2}$ where n is the smaller sequence size and m the larger then there will be an overlap of at least one point in all n size subsequences of m. This is to be avoided and is advisable as a lower bound on the sequence size relationship.

For BP and Shell we found no properties for sequence sizes of less than 15 days on a moving blocks resampled sequence. This may imply that trends relate to longer term behaviour. We found within all results that we tested the moving blocks results confirmed previously found per-

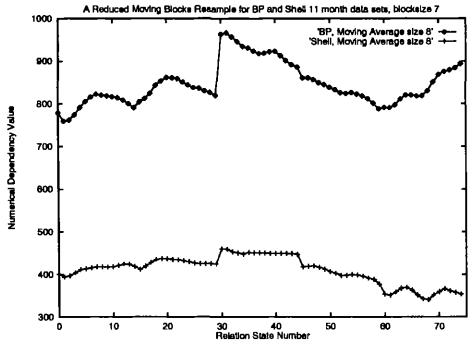

Figure 6.11: Reduced moving blocks samples for BP and Shell moving average data, 78 points from 11 regions and blocksize of 7 points

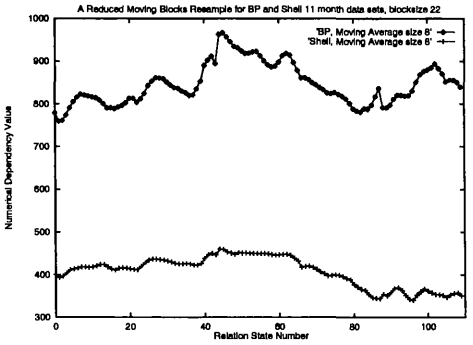

Figure 6.12: Reduced moving blocks samples for BP and Shell moving average data, 110 points from 5 regions and blocksize of 22 points

sistence results and occasionally presented spurious response properties. Repeated application to numerous moving block sequences and intersecting the results removed these spurious properties. Similarly differencing provided similar results in the data we tested; this may be due to forcing sequences and studying local linearity removes the need for longer term trend removal.

6.7 Moving Blocks Bootstrap for Large Relations

The use of the moving blocks bootstrap for large relations allows smaller resampled relation sequences to be created from the original data set. We can see in Figure 6.11, for example, a moving block resample of only 78 points from the original data set of 242 points, which closely resembles the original data set. The need for repeated iterations of the moving blocks samples is shown in the results found. We found $\Delta_{oil} \models (\Diamond^{30}\Box^{15}\ (bp \uparrow_r \wedge^0 sh \uparrow_r))$, and, in abbreviated form, $\Box^{30}\Diamond^{15}\ (bp \uparrow \wedge^0 sh \uparrow)$, $\Box^{30}\Diamond^{15}\ (bp \downarrow \wedge^0 sh \downarrow)$, and $\Box^{30}\Diamond^{15}\ (bp \uparrow \wedge^0 sh \downarrow)$ in one resample, which was not found in the original data set. We also found $\Box^{90}\Diamond^{45}\ (bp \uparrow \wedge^0 sh \uparrow)$ and $\Box^{90}\Diamond^{45}$ $(bp \downarrow \wedge^0 sh \downarrow)$, the latter of which was not found for the original data set.

We draw the following conclusions from using the moving blocks bootstrap for very large relations:

- A visual analysis shows that Figures 6.11 and 6.12 in comparison with Figure 6.9 of the original data set show the similarities for the two smaller resampled sequences. Such similarity can be exploited for knowledge discovery when a series contains a significant number of points to obtain a valuable *synopsis* of the sequence.

- That the resampled original, and possibly even the resampled moving average, data sets of reduced size, contains too many fluctuations, and as such allows the generation of proper-

ties which may be *generally* false of the data. For example, two resampled blocks may be concatenated and they may violate, or satisfy, a trend which holds, or does not.

- The use of the moving blocks bootstrap to cut down the number of points needed to examine for the discovery of properties, is, like the property discovery process itself, highly dependent on the choice of both *block size* and the *region size* from which the blocks are selected. If the block size is too small with respect to the region size it will not reflect trends sufficiently well. It it is too large then it will closely resemble the original sequence, which we might as well use in this situation. There is of course the additional problem of selecting a suitable blocksize in relation to the sequence sizes for property discovery. A blocksize smaller than a sequence size is more likely to result in fewer properties discovered, particularly when the resampled series is much smaller than the original.

- Even for a small number of points the reduced moving blocks bootstrap is able to detect relationships, and properties, across series reasonably well.

Resampling to create reduced size sequences is valuable when the data set is too large to mine in full for property satisfaction.

6.8 Critical Analysis

Our methodology for the discovery of properties has a number of problems. Particular properties are more likely to be discovered for particular sequence size choices. A response rule $\boxminus^m \diamondminus^n$ is much more likely to hold when $m \gg n$ wherein the guarantee property $\diamondminus^n$ is given more time in which to occur. Clearly, the data miner has the choice of setting these parameters.

Another questionable area is that whether all properties discovered are interesting. This is certainly not true. For example, we must be very careful with guarantee properties to ensure that they are not presented as knowledge discovery without good reason. This begs the question, what makes a property interesting? As properties become more complex they are more likely to represent an interesting feature of the dataset. We must be careful with properties that are not *boxed*, i.e., not safety properties. For example, an *ordered persistence property* of the form $\sigma_1 \rightsquigarrow \diamondminus^m \boxminus^n \sigma_2$ can be found for any persistence property apart from the very start of the time sequence, given that σ_1 can be an arbitrary formula which is true in some sequence before $\diamondminus^m \boxminus^n \sigma_2$ holds. Therefore it is of little value in knowledge discovery terms. However, if this property occurs similarly at regular points(within $\boxminus^p$) then we have discovered something potentially very interesting about the data.

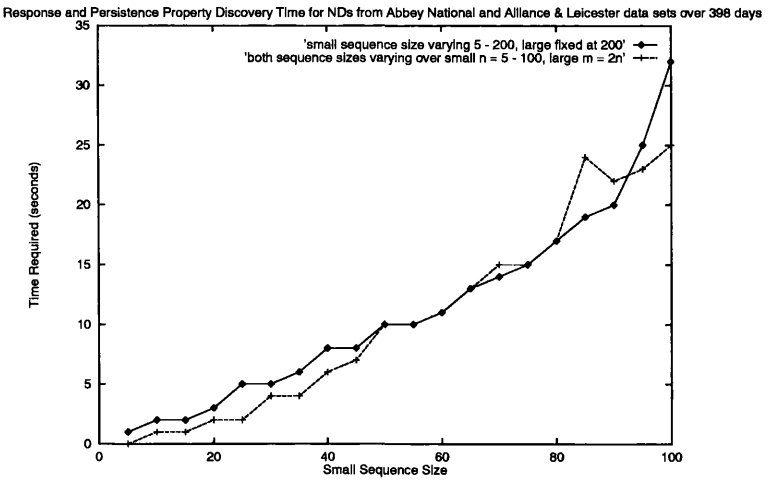

Figure 6.13: Time for discovery of response and persistence properties for varying small and large sequence sizes and small varying only (for a large sequence size) within a 398 point data set

The use of time series statistics has been shown to be both efficient and useful. We have found our representation of lags in the logic to be equivalent, though developed independently by what we considered to be a requirement, to the representation in time series of *lag operators* (Enders, 1995), where the value is ignored and only the lag itself is important. We conducted some tests to examine the efficacy of the lag. This was particularly important as most properties discovered found 0 lag. We overlapped our time series by a number of points n and then removed the extraneous n points at the beginning and end of the respective series. We found similar properties to hold but with the lag to be the same value as the overlap. For example, we found $\Diamond^{30}\boxminus^{15}$ $(ag \downarrow \wedge^0 db \downarrow)$ became $\Diamond^{30}\boxminus^{15}$ $(ag \downarrow \wedge^3 db \downarrow)$ when the overlap was 3 points for the retail data set. As we extended this the number of properties discovered decreased due to the lower likelihood of behaviour reoccurring at regular intervals. Additionally, for small sequences, overlaps also resulted in fewer properties. Upon the advice of (Kendall and Ord, 1990) we restricted lags to $\frac{n}{4}$ given that otherwise stronger lags are found at the highest lag length where there are far fewer points to correlate. (Enders, 1995) suggests beginning with the longest plausible lag length over which there may be a possible relationship.

In Figure 6.13 we provide details of the times required for discovery of response, $\boxminus^m \Diamond^n$, and persistence, $\Diamond^m \boxminus^n$, properties for different sequence sizes. We found that sequence size increasing for both m and n at a fixed rate (with $m = 2n$) was similar to increasing only the sequence size n with respect to property discovery for large sequence size $m = 200$. This is due to most of the computation time working on the discovery of safety and guarantee properties with respect to n. Also there are small fluctuations in the time required. We found this was due to some sequences satisfying fewer initial safety and guarantee properties leading to faster checking time

for response and persistence properties. Figure 6.13 shows that properties can be discovered very efficiently.

In Section 2.3.2 we discussed a number of alternative approaches to temporal data mining. We now compare our approach to that of (Berger and Tuzhilin, 1998) which uses a restricted temporal logic in conjunction with probabilities and an interestingness measure for rule discovery from input strings. In the example we now discuss (Berger and Tuzhilin, 1998) obtain the probability of an event e from dividing its frequency in a string by the total length of the string such that each single event has an interestingness of exactly 1. Within the domain of a `sendmail` program, having 31 commands, rules were discovered, such as $(\textit{sigblock}\,\mathcal{N}\,\textit{setpgrp})\,\mathcal{N}\,\textit{vtrace}$ with an attached interestingness value of 43.16, where *sigblock*, *setpgrp*, and *vtrace* are commands within the program. Additionally $(\textit{sigblock}\,\mathcal{B}_k\,\textit{stepgrp})\,\mathcal{B}_k\,\textit{vtrace}$ is also discovered with the same interestingness value suggesting that $\mathcal{N}$ would be redundant if the subscript k were given explicitly. Our logic may, if applied to a similar domain, represent the latter rule as $(\textit{sigblock} \rightsquigarrow \textit{setpgrp}) \rightsquigarrow \textit{vtrace}$, assuming that we allow program commands as atoms. We can also use properties to denote how often these rules occur within a given period of time. For example, a response rule within every two minute period is expressed by $\boxminus^{120}\diamondsuit^{n}\ (\textit{sigblock} \rightsquigarrow \textit{setpgrp}) \rightsquigarrow \textit{vtrace}$, where n is the time to execute these commands. An extension to our work could be the creation of our own *interestingness* measure based on a property being more interesting if the ratio of the smaller to larger sequence size is closer to 1. This could also be applied from the larger sequence size to complete sequence size. (Berger and Tuzhilin, 1998) also restrict the size of maximum string length for discovery such that the rules found cover only a small size of the original input string. Rules within sequences allow for discovery over longer time periods as we have seen; this is aided by our use of regression.

6.9 Similarity Assessment

Much recent work looking at assessing the similarity of two time series (Agrawal et al., 1995; Faloutsos et al., 1994; Das et al., 1997; Rafiei and Mendelzon, 1997), as discussed in Chapter 2, concentrates on transformations applied to Fourier sequence representation of a time series. As a novel contribution to this field we add our use of property discovery for similarity assessment. From suitable sequence sizes we discover properties which may represent related behaviour across sequences telling us about trends, lags, and seasonal events.

(Rafiei and Mendelzon, 1997) apply Euclidean distance to moving average time series to see if two time series are similar. (Das et al., 1997) applies transformation functions so that the scaling of the two time series being compared need not necessarily be the same over the same

number of points. This is a useful feature which would also be of value for property discovery. Time series are considered similar if there exists an approximate transformation function which maps one series to the other. (Das et al., 1997) considers linear functions only. We now propose another definition of similarity based on sequence sizes such that two sequences are similar if all properties discovered for a particular sequence size depict equivalent behaviour. For example, both moving average trends or seasonal behaviour after differencing would always be equivalent for both time series within all properties found. The work of (Agrawal et al., 1995) does not allow outliers and requires sequences to be of the same length whereas property discovery from moving averages would have the effect of already weakening any outliers. Our form of knowledge discovery will also have the advantage in that properties may be discovered which correspond to a particular range of the sequence within which they may exhibit similar behaviour before diverging. Most studies of similarity would not provide a decent result in this instance, particular if we are looking for a linear transformation function. Though we do not explicitly allows translation across time points of our time series this, as we have shown, is represented by the presentation of lags or lead values within series.

Finally, we remark that if a querying system were implemented for our logic the similarity would be able to be enumerated via a set of queries which may or may not hold.

6.10 Discussion

If a database query language were to incorporate the ability to search for properties within a temporal database then any DB user would be able to ask questions concerning possible properties that he suspects might hold in the data. We have shown that this can be achieved using our logic in polynomial time. The current range of statistical functions available in DBMS need only minimal extension to include time series functions and then it would be entirely feasible to express relationships in a readily understandable form such as that of our logic.

We have presented our logic for NDs in temporal sequences. Results applied to temporal relation sequences and time series have shown our logic capable of providing succinct characterisation of the data to a *system user*. The response and persistence properties that we discovered are both useful and valid and may be applicable in a decision support environment. Properties discovered within a DBMS might be desired to hold for all future points in which case they could be elevated to the status of integrity constraints. Extensions to this work include the implementation of a querying system and additional algorithms for property discovery.

We presented a generic algorithm for the discovery of knowledge using the temporal classification of properties and then refined this to a specialised algorithm for response and persistence

rule discovery. The algorithm is generic in that it applies to all discovery which uses the classification hierarchy, of which our model in Figure 6.1 is one particular instance. The similarities between these and the generic algorithms given in (Mannila, 1996; Mannila, 1997) point to similarities within the data mining model. Work on a unified theory of data mining will require a set of generic mining algorithms which can be specialised for many different approaches (Jaeger et al., 1996).

The goals of (Berger and Tuzhilin, 1998) were to generate unexpected predicates, expressed in a restricted temporal logic, from sequential databases or strings. This has application in rule discovery from categorical data whilst our logic relies on numerical data alone, though in the case of NDs this may be based on categorical data. The restriction of a maximum string length is similar to our requirement of a given sequence size. A sequence of size n satisfying $\sigma_1 \rightsquigarrow \sigma_2$ differs only from $\sigma_1 \mathcal{B}_k \sigma_2$ found in a string of size n in that we allow overlap, assuming that σ_1, σ_2 occur in sequences. The need for restriction is that the interestingness measure is always higher for longer sequences implying that a rule representing the complete input string is always the most interesting whereas our motivation is to enable property discovery, possibly relating to, say, seasonal behaviour. Both properties and measures, such as interestingness, are of value within data mining.

Much recent knowledge discovery research has been concerned with finding out if two time series are in some sense similar (Faloutsos et al., 1994; Agrawal et al., 1995; Das et al., 1997). Our logic has the expressive power to represent similarities as properties or standard sentences of the logic from which we can easily deduce similarities between two time series. As an avenue for further work it would be interesting to expand this using, perhaps, intersections of properties found for similarity assessment.

CHAPTER 7

Summary and Conclusion

In this thesis we have presented a novel methodology for data mining in indefinite and temporal databases. We have demonstrated throughout this thesis how NDs are useful within the data mining process. In the thesis we have provided empirical evidence that our dynamic use of resampling is effective for determination of a sample size; this may have applications for other NP-complete problems. Also, we have shown that our temporal logic for time sequences (of NDs) is easily applicable and a viable addition to the data mining toolkit.

7.1 Contribution of this work

We have outlined a general framework for data mining in non-standard databases, not previously considered. In the most informal sense, we take sets of approximations to FDs, in this thesis we consider only NDs, and use statistical functions and tools to infer conclusions on patterns within the data; in the domain of indefinite information we use resampling in a dynamic fashion based upon mean, variance or standard errors satisfied by sets of NDs. In the temporal domain we can use resampling or moving averages to form new sequences from the original values of ND set satisfaction, which change over time, to determine specific properties which may hold within the temporal relation sequences. Our use, in a general sense, of statistical functions upon large sets or sequences of NDs to discover information can be viewed as a *second order data mining*. We evidence this general approach in two domains though we speculate that there may be many more applications in other domains, ranging from spatial to active databases.

We now describe in more detail the specific contributions made by this work. Chapter 3 has shown that NDs are viable dependencies within the relational model, extending the intentions of Grant and Minker (Grant and Minker, 1985a; Grant and Minker, 1985b) when they introduced NDs as extensions of FDs for greater flexibility in schema specification. Principally we use the chase for NDs, proven to be sound and complete herein, for the inference of NDs; we show this to

be decidable. In this way the chase allows for ND inference to be tested in database applications, although this may be intractable. We also show how NDs themselves may be used within data mining or database design algorithms to approximate FD sets, demonstrated via an evolutionary database design algorithm. NDs were shown to effectively extend the class of methods approximating FD sets in a relation. ND mining may be limited in the sense that for an ND $X \rightarrow^k A$, if A is a category of exactly k elements, then the ND only tells us that all elements occur in A; it may be considered more informative if this were not the case, perhaps in continuous domains. Also, the mean ND combats this problem by providing more information within a data mining context. A metric for ND sets is also provided which we employed within our work on indefinite information in relations.

We studied indefinite information in relations, concentrating solely on the consistency problem, known to be NP-complete. We created a general randomised procedure which made use of a chase developed using NDs for indefinite relations and a dynamic resampling technique. We chose to employ resampling to be able to make statistically valid inferences from a sample of possible worlds taken from an indefinite relation. Each possible world satisfies an ND set. Resampling from a sample of possible worlds allows us to determine approximate values of variance and standard deviation. Our randomised algorithms require a sufficient sample size upon which to apply their selection functions so as to obtain decent approximations to FD set satisfaction. We found that as the variance and standard deviation change with the degree of indefinite cells in a relation it is possible to apply resampling iteratively on increasing sample sizes until an approximate fixpoint is reached. Independent of our work, (John and Langley, 1996) argue, in a position paper, for dynamic sampling to be adopted within data mining instead of naive sampling techniques in use. Our work does just this. Extensive simulations on these methods showed that the chase is of use in a larger relations with correspondingly larger domain sizes and that the resampling is useful for providing an upper bound on the number of possible worlds required.

In Chapters 5 and 6 we demonstrate the practicality of NDs in temporal databases. Given that changing ND sets, from a user supplied template, may only vary on their branching factor we can view the sequence of changes as a time series. Considering specific time series analysis techniques as a basis we developed a logic using modal operators to discover rules which a sequence may satisfy. Necessary restriction of the formulae of our temporal logic to properties, used within program verification, proved to be highly useful for knowledge discovery. We make no grand claims on the formalisation of our logic with respect to it being a panacea for time series data mining though we note that it allows for knowledge to be represented succinctly and has an

easily understandable semantics; an important yet understated factor of many knowledge discovery systems. Further theoretical analysis of the logic, outside the scope of this thesis, is definitely required. It is most likely that logics for time series analysis could be developed in many different ways.

Properties of temporal logic which were defined for program verification have been extended for data mining purposes. The specifications required in programs for correctness analysis lends itself well to knowledge discovery where changing inputs over time may frequently satisfy similar conditions. Properties of temporal logic have not, in the limits of our experience, been considered for data mining.

As we have shown this thesis is a contribution to the arena of data mining in both techniques and tools. We show that NDs are valuable within data mining and believe that the techniques of our randomised algorithms, dynamic resampling, and temporal logic have clear application. We feel that our hypothesis of NDs for data mining in non-standard relations has been vindicated via the work demonstrated herein.

7.2 Applications

There are a number of applications within which this work can be used, which we now detail:

- Our general framework can be transferred to other domains. For example, in a spatial database we can, after input of a FD set as a template, mine for ND set satisfaction of this template and then employ (or develop) statistics which are pertinent to spatial data sets; (Koperski et al., 1996) presents k-predicates for spatial data representation of the form, for example, $close\_to(x, lake) \wedge close\_to(x, road)$ implying that x is *close to* both a lake and a road which could also be summarised as an ND object $\rightarrow^k$ site in a relation CLOSE_TO(object,site). We believe that we could discover and use patterns represented by such NDs in spatial databases.

- We can employ dynamic resampling to generate a *representative* sample size in a number of NP-complete problems.

- Our logic can be applied to any time series for property detection.

- We can mine any database for ND set satisfaction. The metric presented in Chapter 3 can be applied to any set of NDs, assuming a finite domain.

7.3 Directions for future research

There are many directions for possible future research posed by this work, in domains of dependency theory (for data mining), temporal/time series data mining, and indefinite data mining. We begin by considering a direct extrapolation of this research.

7.3.1 Open Problems

This thesis has the following *important* open problems:

- The implementation of efficient mining procedures for NDs in standard relations. Extensions for NDs to the dynamic dependencies presented in (Vianu, 1987; Vianu, 1988) as outlined in Section 2.2.8 warrant further analysis, with regard to both database theory and data mining research
- A study of algorithms to create weak Armstrong Relations, as defined in Section 3.3.3, for Database design purposes.
- A theoretical analysis of our dynamic resampling algorithm, WORLD_LIMIT, is required.
- We conjecture that implication for ND sets with the chase is an NP-complete problem. It would be interesting to search for special classes of NDs or relations, possibly incomplete, within which the chase procedure is polynomial in execution time. This work would be similar in spirit to that of (Levene and Loizou, 1997).
- Implementation of a query system based on our temporal logic, NDLTL.
- An in-depth study of expressiveness of non-standard logics, such as NDLTL, is required. This would be particularly useful with a view to data mining applications.

We elaborate on some of these issues in the next section.

7.3.2 Further work

In the arena of NDs we could further extend their applicability by the creation of scaling and translation functions, as used for transformation functions in time series similarity (Agrawal et al., 1995). These functions have direct application when we are dealing with relations that are of vastly different sizes. As noted in Section 3.6 we could also investigate more sophisticated algorithms for the mining of NDs in standard relations. This work could make use of many heuristics including hypergraph transversals. One example using ND semantics may be that we do not have to consider mining the remainder of a relation if we have found a partition for an ND whose branching factor is greater than over half the number of tuples in the relation. Dynamic Dependencies introduced for FDs by (Vianu, 1987) would have a highly useful semantics if extended to

NDs, as motivated by the example in Section 2.2.8. It would be of value to mine corporate, and other, databases for the presence of these relationships whereby the branching factor of an ND may determine subsequent branching factors of itself and other NDs later in the timeline.

The work on searching for a satisfying possible world within an indefinite relation provides numerous avenues for further study. Clearly, it would be highly interesting to use real-world scheduling representations to see how useful our ND approximation sets are. We could also analyse rates of convergence for our resampling process with respect to the nature of an indefinite relation and the FD set used. Further study of this within such dynamic algorithms as our procedure would be very useful, both in terms of data mining and of relevance to a multi-disciplinary research field. Additionally, phase transitions in indefinite relations, referred to in Section 4.5, would be a most interesting further study, complementing previous phase transition work with dependencies and relations that have a real information content.

Finally, our work on temporal data mining requires a thorough study of the logic we have created. The inclusion of time series functionality makes the expressive nature of the logic unclear. The flexibility of the logic means that it is easily extended. Further research into time series behaviour may provoke the need for additional operators. We believe that this would include functions designed specifically for the analysis of non-linear relationships. We would also like to be able to spend time developing sophisticated algorithms which use this logic for temporal data mining. One such example would be to discover a suitable sequence size upon which to conduct the data mining process. Error functions from regression analysis could also be incorporated into the logic. Such would be desirable from a systems point of view.

7.4 The Evolution of Data Mining

Data Mining is a rapidly expanding field, not least due to a concentrated global effort into the extraction of information from data. The state of the art applications are still led by recent theoretical developments. There will be a significant increase in the use of statistical developments within data mining products. Our use of resampling in both the temporal and indefinite domains shows how such novel processes can be applied easily and effectively. More data mining tools will incorporate sampling and resampling in the quest for information which may *characterise* a data set.

There have been recent criticisms that data mining, as yet, is not fully integrated with the database interface (Mannila, 1997; John, 1997; Chaudhuri, 1998). It is only a matter of time before the next relational database upgrade includes a data mining toolkit. For clarity and ease of use, there is potential for the inclusion of such items as NDs and temporal logic. This, and other,

logics would make use of statistical functions within the database query language.

The process of data mining will mesh with databases so that predictors and forecasting can be assessed at any time, which may be NDs or other dependencies. These predictors themselves may be mined and the technique of building our logic upon dependencies as atoms is perhaps a first step in this direction.

7.5 Conclusions

The field of knowledge discovery is rapidly expanding due to the ever-increasing amounts of data being stored. The user-centric processes of data mining are extending the fields of statistics, artificial intelligence and machine learning into a new science (Fayyad and Uthurusamy, 1996). Our work has made significant use of database, statistical, and logical theory to develop a new general framework for data mining in temporal and indefinite relations.

BIBLIOGRAPHY

Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases*. Addison–Wesley, Massachusetts.

Agrawal, R., Imielinski, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. In Buneman, P. and Jajobia, S., editors, *Proceedings of the 1993 ACM SIGMOD*, pages 207–216, Washington, D.C. ACM Press.

Agrawal, R., Lin, K.-I., Sawhney, H., and Shim, K. (1995). Fast similarity search in the presence of noise, scaling and translation in time-series databases. In Dayal, U., Gray, P., and Nishio, S., editors, *Proceedings of the 21st international conference on very large databases (VLDB '95)*, pages 490 – 501.

Akutsu, T. and Takasu, A. (1994). On PAC learnability of functional dependencies. *New Generation Computing*, 12:359 – 374.

Alagar, V., Bergler, S., and Dong, F. Q., editors (1993). *Incompleteness and Uncertainty in Information Systems*, Montreal. Springer–Verlag.

Allen, J. F. (1984). Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154.

Anthony, M. and Biggs, N. (1992). *Computational Learning Theory*. Cambridge Tracts in Theoretical Computer Science – 30. Cambridge University Press, Cambridge, England.

Armstrong, W. (1974). Dependency structures of data base relationships. In *Proceedings of the IFIP Congress*, pages 580 – 583, Stockholm. Elsevier - North-Holland.

Atzeni, P. and De Antonellis, V. (1993). *Relational Database Theory*. Benjamin/Cummings, Redwood City, California.

Bäck, T. and Schwefel, H. (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1– 23.

Beaubouef, T., Petry, F. E., and Buckles, B. P. (1995). Extension of the relation database and its algebra with rough set techniques. *Computational Intelligence*, 11(2):233 – 245.

Beeri, C. and Bernstein, P. A. (1979). Computational problems related to the design of normal form relational schemas. *ACM Transactions on Database Systems*, 4(1):30 – 59.

Beeri, C., Dowd, M., Fagin, R., and Statman, R. (1984). On the structure of Armstrong relations for functional dependencies. *Journal of the ACM*, 31(1):30 – 46.

Beeri, C. and Vardi, M. (1984). A proof procedure for data dependencies. *Journal of the ACM*, 31(4):718 – 741.

Bell, S. (1995). Discovery and maintenance of functional dependencies by independencies. In Fayyad, U. M. and Uthurusamy, R., editors, *Proceedings First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, pages 27–32, Montreal, Canada. AAAI Press.

Bell, S. and Brockhausen, P. (1995). Discovery of data dependencies in relational databases. Technical Report LS-8 Report 14, Informatik VIII - University Dortmund.

Berger, G. and Tuzhilin, A. (1998). Discovering unexpected patterns in temporal data using temporal logic. In *Temporal Databases - Research and Practice*, volume 1399 of *Lecture Notes in Computer Science*, pages 281–309. Springer–Verlag.

Berndt, D. J. and Clifford, J. (1996). Finding patterns in time series: A dynamic programming approach. In Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., editors, *Advances in Knowledge Discovery and Data Mining*.

Bettini, C., Sean Wang, X., and Jajodia, S. (1996). Testing complex temporal relationships involving multiple granularities and its application to data mining. In *Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium of principles of database systems*, pages 68 – 78, Montreal, Canada. ACM Press.

Bitton, D., Millman, J., and Torgersen, S. (1989). A feasibility and performance study of dependency inference. In *Proceedings of the 5th International Conference on Data Engineering, February 6-10, 1989, Los Angeles, California, USA*, pages 635–641. IEEE Computer Society.

Blake, C., Keogh, E., and Merz, C. J. (1998). UCI repository of machine learning databases. http://www.ics.uci.edu/~mlearn/MLRepository.html, University of California, Irvine, Dept. of Information and Computer Sciences.

Bosc, P., Dubois, D., and Prade, H. (1994). Functional dependencies and quotient operators in fuzzy databases. Technical Report 41, Institut de recherche en informatique de Toulouse.

Brachman, R. J. and Anand, T. (1996). *The process of knowledge discovery in databases*, pages 37– 58. In (Fayyad et al., 1996d).

Chaudhuri, S. (1998). Data mining and database systems: Where is the intersection? *Bulletin of the Technical Committee on Data Engineering*, 21(1):4 – 8.

Cheeseman, P., Kanefsky, B., and Taylor, W. M. (1991). Where the *really* hard problems are. In Mylopoulos, J. and Reiter, R., editors, *Proceedings of IJCAI-91*, pages 331–337, San Mateo, CA. Morgan Kaufmann.

Chomicki, J. (1994). Temporal integrity constraints in relational databases. *Bulletin of the Technical Committee on Data Engineering*, 17(2):33–37.

Chomicki, J. and Toman, D. (1998). Temporal logic in information systems. In Chomicki, J. and Saake, G., editors, *Logics for Databases and Information Systems*. Kluwer Academic Publishers.

Cleveland, W. S. and Loader, C. L. (1996). Smoothing by local regression: Principles and methods. In Haumlrdle, W. and Schimek, M. G., editors, *Statistical Theory and Computational Aspects of Smoothing*, pages 10 – 49. Springer–Verlag, New York.

Clifford, J. and Tuzhilin, A., editors (1995). *Recent Advances in Temporal Databases: Proceedings of the International Workshop on Temporal Databases, 17-18 September, 1995*, Workshops in Computing, Zurich, Switzerland. Springer - Verlag.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387.

Codd, E. F. (1972). Further normalization of the data base relational model. In Rustin, R., editor, *Data Base Systems*, volume 6 of *Courant Computer Science Symposium*, pages 33–64. Prentice Hall, Englewood Cliffs, NJ.

Collopy, E. and Levene, M. (1996). Evolving example relations to satisfy functional dependencies. Technical Report RN/96/124, University College London, U.K.

Collopy, E. and Levene, M. (1998a). Evolving example relations to satisfy functional dependencies. In Tamer Özsu, M., Dogac, A., and Ulusoy, O., editors, *Proceedings of the Third Biennial World Conference on Integrated Design and Process Technology - Issues and Applications of Database Technology, (IADT '98)*, volume 2, pages 440 – 448, Berlin, Germany. Society for Design and Process Science.

Collopy, E. and Levene, M. (1998b). Knowledge discovery from numerical dependencies in temporal sequences. Technical Report RN/98/98, University College London, U.K. Submitted to Data Mining and Knowledge Discovery.

Collopy, E. and Levene, M. (1998c). Resampling in an indefinite database to approximate functional dependencies. In Żytkow, J. M. and Quafafou, M., editors, *Principles of Data Mining and Knowledge Discovery, Second European Symposium, (PKDD '98), Nantes, France, September 23-26 1998, Proceedings*, volume 1510 of *Lecture Notes in Artificial Intelligence*, pages 291 – 299. Springer–Verlag.

Collopy, E. and Levene, M. (1998d). Using numerical dependencies and the bootstrap for the consistency problem. In Lee, J. and Thuraisingham, B., editors, *Proceedings IEEE Knowledge and Data Engineering Exchange Workshop, (KDEX '98), Taipei, Taiwan, November 9, 1998*, pages 160–167. IEEE Press.

Das, G., Gunopulos, D., and Mannila, H. (1997). Finding similar time series. In Komorowski, H. J. and Żytkow, J. M., editors, *Principles of Data Mining and Knowledge Discovery, First European Symposium, (PKDD '97), Trondheim, Norway, June 24-27, 1997, Proceedings*, volume 1263 of *Lecture Notes in Artificial Intelligence*, pages 88–100. Springer–Verlag.

Das, G., Lin, K.-I., Mannila, H., Renganathan, G., and Smyth, P. (1998). Rule discovery from time series. In *Proc. of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*, pages 16 – 22, New York City, New York, USA. AAAI Press.

Date, C. J. (1995). *An Introduction to Database Systems*. Addison–Wesley, Reading, Massachusetts, sixth edition.

Davey, B. and Priestly, H. (1990). *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, U.K.

Demetrovics, J., Katona, G., and Sali, A. (1992). The characterization of branching dependencies. *Discrete Applied Mathematics*, 40:139–153.

Demetrovics, J., Ronyai, L., and Son, H. N. (1993). Functional dependencies among boolean dependencies. *Annals of Mathematics and Artificial Intelligence*, 7:83 – 106.

Demetrovics, J. and Thi, V. D. (1995). Some observations on the minimal Armstrong relations for normalised relation schemes. *Computers and Artificial Intelligence*, 14(5):455 – 467.

Diaconis, P. and Efron, B. (1983). Computer-intensive methods in statistics. *Scientific American*, 248(5):116–130.

Dopazo, J. (1994). Estimating errors and confidence intervals for branch lengths in phylogenetic trees by a bootstrap approach. *Journal of Molecular Evolution*, 38:300–304.

Efron, B. (1979). Computers and the theory of statistics: thinking the unthinkable. *SIAM Review*, 21(4):460 – 480.

Efron, B. (1982). *The jackknife, the bootstrap and other resampling plans*. Number 38 in CBMS-National Science Foundation Monograph. Society for Industrial and Applied Mathematics.

Efron, B. and Tibshirani, R. (1986). Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical Science*, 1(1):54 – 77.

Efron, B. and Tibshirani, R. (1993). *An Introduction to the bootstrap*. Number 57 in Monographs on Statistics and Applied Probability. Chapman and Hall.

Eiter, T. and Gottlob, G. (1995). Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278 – 1304.

Emerson, E. A. (1990). Temporal and modal logic. In van Leeuwen, J., editor, *Formal Models and Semantics*, volume B of *Handbook of Theoretical Computer Science*, pages 995 – 1072. Elsevier Science.

Enders, W. (1995). *Applied Econometric Time Series*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, inc.

Fagin, R. (1977). Functional dependencies in a relational database and propositional logic. *IBM Journal of Research and Development*, 21:534 – 544.

Fagin, R. (1982). Armstrong databases. Technical Report RJ3440 (40926), IBM Research Report.

Fagin, R. and Vardi, M. (1983). Armstrong relations for functional and inclusion dependencies. *Information Processing Letters*, 16(1):13 – 19.

Faloutsos, C., Ranganathan, M., and Manolopoulos, Y. (1994). Fast subsequence matching in time-series databases. In *Proceedings of the 1994 ACM SIGMOD*, pages 419 – 429.

Faraway, J. and Chatfield, C. (1995). Time series forecasting with neural networks: A case study. Technical Report 95-06, University of Bath.

Fayyad, U. (1998a). Editorial. *Data Mining and Knowledge Discovery*, 2(1):5–7.

Fayyad, U. (1998b). Mining databases: Towards algorithms for knowledge discovery. *Bulletin of the Technical Committee on Data Engineering*, 21(1):39 – 48.

Fayyad, U., Haussler, D., and Stolorz, P. (1996a). Mining scientific data. *Communications of the ACM*, 39(11):51–57.

Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996b). The KDD process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11):27–34.

Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996c). Knowledge discovery and data mining: Towards a unifying framework. In Simoudis, E., Han, J., and Fayyad, U., editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 82 – 87, Portland, Oregon. AAAI Press.

Fayyad, U. and Uthurusamy, R. (1996). Editorial. *Communications of the ACM*, 39(11):25–26.

Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., editors (1996d). *Advances in Knowledge Discovery and Data Mining*. MIT Press.

Gabbay, D., Pnueli, A., Shelah, S., and Stavi, J. (1980). On the temporal analysis of fairness. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 163 – 173.

Gabbay, D. M., Hodkinson, I., and Reynolds, M. (1994). *Temporal Logic - Mathematical Foundations and Computational Aspects*. Number 28 in Oxford Logic Guides. Oxford University Press.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A guide to the theory of NP-Completeness*. W. H. Freeman, New York.

Gertz, M. and Lipeck, U. W. (1995). "Temporal" integrity constraints in temporal databases. In (Clifford and Tuzhilin, 1995), pages 77 – 92.

Glymour, C., Madigan, D., Pregibon, D., and Smyth, P. (1997). Statistical themes and lessons for data mining. *Data Mining and Knowledge Discovery*, 1(1):11–28.

Gottlob, G. and Libkin, L. (1990). Investigations on Armstrong relations, dependency inference and excluded functional dependencies. *Acta Cybernetica*, 9(4):385 – 402.

Grant, J. and Minker, J. (1985a). Inferences for numerical dependencies. *Theoretical Computer Science*, 41:271–287.

Grant, J. and Minker, J. (1985b). Normalisation and axiomatisation for numerical dependencies. *Information and Control*, 65:1–17.

Gunopulos, D., Mannila, H., and Saluja, S. (1997). Discovering all most specific sentences by randomized algorithms. In Afrati, F. and Kolaitis, P., editors, *Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings*, volume 1186 of *Lecture Notes in Computer Science*, pages 215–229.

Haack, S. (1978). *Philosophy of Logics*. Cambridge University Press.

Hale, J., Finnerty, S., and Shenoi, S. (1994). Analysing inference in fuzzy database systems. In *Proceedings of 3rd IEEE International Conference on Fuzzy Systems*, pages 325 – 330. IEEE Computer Press.

Hale, J. and Shenoi, S. (1995). Imprecise database inference using functional dependencies. In Wang, P. P., editor, *Advances in Fuzzy Theory and Technology*, volume 3, pages 1 – 15, Durham, NC. Bookwright.

Halpern, J. Y. and Rabin, M. O. (1983). A logic to reason about likelihood. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 310–319, Boston, Massachusetts.

Hammer, J., Garcia-Molina, H., Widom, J., Labio, W., and Zhuge, Y. (1995). The Stanford data warehousing project. *Bulletin of the Technical Committee on Data Engineering*, 18(2):41 – 48.

Holsheimer, M., Kersten, M., Mannila, H., and Toivonen, H. (1995). A perspective on databases and data mining. In Fayyad, U. M. and Uthurusamy, R., editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, pages 150 – 155. AAAI Press.

Holsheimer, M. and Siebes, A. (1994). Data mining: the search for knowledge in databases. Technical Report 9406, Centrum voor Wiskunde en Informatica.

Hooker, J. N. (1994). Needed: An empirical science of algorithms. *Operations Research*, 42(2):201 – 212.

Hu, T. X. (1995). *Knowledge Discovery in Databases: An Attribute-Oriented Rough Set Approach*. PhD thesis, University of Regina, Canada.

Huhtala, Y., Kärkkäinen, J., Porkka, P., and Toivonen, H. (1998). Efficient discovery of functional and approximate dependencies using partitions. In *14th International Conference on Data Engineering (ICDE'98)*, pages 392 – 401, Orlando, Florida. IEEE Computer Society Press.

Imielinski, T. (1991). Abstraction in query processing. *Journal of the ACM*, 38(3):534 – 558.

Imielinski, T. and Lipski, W. (1984). Incomplete information in relational databases. *Journal of the ACM*, 31(4):761 – 791.

Imielinski, T., Naqvi, S., and Vadaparty, K. (1991). Incomplete objects - a data model for design and planning applications. In *Proceedings of the ACM SIGMOD*, pages 288 – 297. ACM Press.

Imielinski, T. and Vadaparty, K. (1989). Complexity of query processing in databases with OR-objects. In *Proceedings of the Eighth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS '89)*, pages 51 – 65.

Imielinski, T., Van Der Meyden, R., and Vadaparty, K. (1995). Complexity tailored design: A new design methodology for databases with incomplete information. *Journal of Computer and System Sciences*, 51:405–432.

Inmon, W. H. (1996). *Building the Data Warehouse*. Wiley, New York.

Jaeger, M., Mannila, H., and Weydert, E. (1996). Data mining as selective theory extraction in probabilistic logic. In *SIGMOD '96 Workshop on Knowledge Discovery*.

Jensen, C. S., Dyreson, C. E., Boehlen, M., and Clifford, J. (1998). The consensus glossary of temporal database concepts — February 1998 version. *Lecture Notes in Computer Science*, 1399:367–405.

Jensen, C. S., Snodgrass, R. T., and Soo, M. S. (1992). Extending normal forms to temporal relations. Technical Report TR 92-17, University of Arizona.

Jensen, C. S., Snodgrass, R. T., and Soo, M. S. (1996). Extending existing dependency theory to temporal databases. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):563–582.

John, G. (1997). *Enhancements to the Data Mining Process*. PhD thesis, Stanford University.

John, G. H. and Langley, P. (1996). Static versus dynamic sampling for data mining. In Simoudis, E., Han, J., and Fayyad, U., editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 367–370, Portland, Oregon. AAAI Press.

Kanellakis, P. C. (1980). On the computational complexity of cardinality constraints in relational databases. *Information Processing Letters*, 11(2):98–101.

Kendall, M. and Ord, J.-K. (1990). *Time Series*. Edward Arnold, third edition.

Kent, W. (1983). A simple guide to the five normal in relational database theory. *Communications of the ACM*, 26(2):120 – 125.

Keogh, E. and Smyth, P. (1997). A probabilistic approach to fast pattern matching in time series databases. In Heckerman, D., Mannila, H., Pregibon, D., and Uthurusamy, R., editors, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97)*, pages 24–30. AAAI Press.

Kim, W. (1990). *Introduction to Object-Oriented Databases*. MIT Press, Cambridge, Massachusetts.

Kivinen, J. and Mannila, H. (1994). The power of sampling in knowledge discovery. In *Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '94)*, pages 77–85. ACM Press.

Kivinen, J. and Mannila, H. (1995). Approximate inference of functional dependencies from relations. *Theoretical Computer Science*, 149(1):129 – 149.

Klemettinen, M., Mannila, H., Ronkainen, P., Toivonen, H., and Inkeri-Verkamo, A. (1994). Finding interesting rules from large sets of discovered association rules. In Adam, N., Bhargava, B., and Yesha, Y., editors, *Third International Conference on Information and Knowledge Management*, pages 401–407, Gaithersburg, Maryland. ACM Press.

Koperski, K., Adhikary, J., and Han, J. (1996). Spatial data mining: Progress and challenges. In *SIGMOD'96 Workshop. on Research Issues on Data Mining and Knowledge Discovery (DMKD'96)*, Montreal, Canada.

Laird, P. (1993). Identifying and using patterns in sequential data. In *Algorithmic Learning Theory: 4th International Workshop*, pages 1–18.

Lerat, N. (1986). Query processing in incomplete logical databases. In Ausiello, G. and Atzeni, P., editors, *International Conference on Database Theory (ICDT '86), Rome, Italy, September 8-10, 1986, Proceedings*, volume 243 of *Lecture notes in computer science*, pages 260 – 277. Springer-Verlag.

Levene, M. (1995). A lattice view of functional dependencies in incomplete relations. *Acta Cybernetica*, 12:181 – 207.

Levene, M. and Loizou, G. (1997). Null inclusion dependencies in relational databases. *Information and Computation*, 136(2):67 – 108.

Levene, M. and Loizou, G. (1998). Axiomatisation of functional dependencies in incomplete relations. *Theoretical Computer Science*, 206(1-2):283–300.

Levene, M. and Vincent, M. W. (1997). Recovery from inconsistency in incomplete relations. Technical Report RN/97/81, Department of Computer Science, University College London.

Liddle, S. W., Embley, D. W., and Woodfield, S. N. (1993). Cardinality constraints in semantic data models. *Data & Knowledge Engineering*, 11:235 – 270.

Lipski, W. (1979). On semantic issues connected with incomplete information databases. *ACM Transactions on Database Systems*, 4(3):261 – 296.

Lloyd, J. (1987). *Foundations of Logic Programming*. Springer–Verlag, Berlin, second,extended edition.

Lucchesi, C. L. and Osborn, S. L. (1978). Candidate keys for relations. *Journal of Computer and System Sciences*, 17(2):270–279.

Maier, D. (1983). *The Theory of Relational Databases*. Computer Science Press, New York.

Maier, D., Mendelzon, A. O., and Sagiv, Y. (1979). Testing implications of data dependencies. *ACM Transactions on Database Systems*, 4(4):455–469.

Makowsky, J. A. (1987). Why Horn formulas matter in computer science: Initial structures and generic examples. *Journal of Computer and System Sciences*, 34:266–292.

Manna, Z. and Pnueli, A. (1992). *The Temporal Logic of Reactive and Concurrent Systems - Specification*. Springer–Verlag.

Mannila, H. (1996). Data mining: machine learning, statistics, and databases. In *Proceedings of the Eight International Conference on Scientific and Statistical Database Management*, Stockholm.

Mannila, H. (1997). Methods and problems in data mining. In Afrati, F. and Kolaitis, P., editors, *Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings*, volume 1186 of *Lecture Notes in Computer Science*, pages 41–55.

Mannila, H. and Räihä, K.-J. (1986). Design by example: An application of Armstrong relations. *Journal of Computer and System Sciences*, 33:126 – 141.

Mannila, H. and Räihä, K.-J. (1992a). *The Design of Relational Databases*. Addison–Wesley.

Mannila, H. and Räihä, K.-J. (1992b). On the complexity of dependency inference. *Discrete Applied Mathematics*, 40:237 – 243.

Mannila, H. and Räihä, K.-J. (1994). Algorithms for inferring functional dependencies from relations. *Data & Knowledge Engineering*, 12:83 – 99.

Mannila, H. and Toivonen, H. (1996a). Discovering generalized episodes using minimal occurrences. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 146–151, Portland, Oregon. AAAI Press.

Mannila, H. and Toivonen, H. (1996b). Multiple uses of frequent sets and condensed representations. In Simoudis, E., Han, J., and Fayyad, U. M., editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 189 – 194, Portland, Oregon. AAAI Press.

Mannila, H., Toivonen, H., and Verkamo, A. I. (1995). Discovering frequent episodes in sequences. In Fayyad, U. M. and Uthurusamy, R., editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, pages 210 – 215, Montreal, Canada. AAAI Press.

Mitchell, D., Levesque, H., and Selman, B. (1992). A new method for solving hard satisfiability problems. In *Proceedings of the tenth national conference on artificial intelligence (AAAI-92)*, pages 440 – 446, San Jose.

Nazem, S. M. (1988). *Applied Time Series Analysis for Business and Economic Forecasting*. Marcel Dekker, New York.

Orlowska, M. and Ewald, C. (1992). Schema evolution - the design and integration of fact based schemata. In Srinivasan, B. and Zeleznikow, J., editors, *Research and Practical Issues in Databases, Proceedings of the 3rd Australian Database Conference*, pages 306 – 320, La Trobe University. World Scientific.

Padmanabhan, B. and Tuzhilin, A. (1996). Pattern discovery in temporal databases: A temporal logic approach. In Simoudis, E., Han, J., and Fayyad, U., editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 351 – 354, Portland, Oregon. AAAI Press.

Park, S. and Miller, K. (1988). Random number generators: Good ones are hard to find. *Communications of the ACM*, 31(10):1192 – 1201.

Pfahringer, B. and Kramer, S. (1995). Compression-based evaluation of partial determinations. In Fayyad, U. M. and Uthurusamy, R., editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, Montreal, Canada. AAAI Press.

Piatetsky-Shapiro, G. and Frawley, W., editors (1991). *Knowledge Discovery in Databases*. MIT Press.

Piatetsky-Shapiro, G. and Matheus, C. J. (1993). Measuring data dependencies in large databases. In *Knowledge Discovery in Databases: Papers from the 1993 Workshop*, pages 162–173, Washington DC. AAAI Press.

Pnueli, A. (1977). The temporal logic of programs. In *Proceedings 18th Annual IEEE Symposium on Foundations of Computer Science*, pages 46–57.

Prasad-Sistla, A. (1994). Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6(5):495 – 511.

Rafiei, D. (1999). On similarity-based queries for time series data. In *Proc. IEEE Intl. Conf. on Data Eng. (ICDE '99) (to appear)*, Sydney, Australia.

Rafiei, D. and Mendelzon, A. (1997). Similarity-based queries for time series data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 13–24, Tucson, Arizona.

Ramakrishnan, R., Srivastava, D., and Sudarshan, S. (1992). CORAL - control, relations and logic. In Yuan, L.-Y., editor, *Proceedings of the 18th VLDB Conference*, pages 238 – 250, Vancouver, British Columbia, Canada. Morgan Kaufmann.

Roddick, J. F. (1994). A survey of schema versioning issues for database systems. Technical Report CIS-94-010, University of South Australia.

Sagiv, Y., Delobel, C., Stott Parker, C., and Fagin, R. (1981). An equivalence between relational database dependencies and a fragment of propositional logic. *Journal of the ACM*, 28(3):435 – 453.

Savnik, I. and Flach, P. (1993). Bottom-up induction of functional dependencies from relations. In Piatetsky-Shapiro, G., editor, *Proceedings of the AAAI-93 Workshop on Knowledge Discovery in Databases*, pages 174–185.

Schlimmer, J. C. (1993). Efficiently inducing determinations: A complete and systematic search algorithm that uses optimal pruning. In *Machine Learning: Proceedings of the tenth international conference*, pages 284 – 290, Amherst, Massachusetts. Morgan Kaufmann.

Schmidt, D., Marti, R., Dittrich, A. K., and Dreyer, W. (1995). Time series, a neglected issue in temporal database research? In (Clifford and Tuzhilin, 1995), pages 214 – 234.

Segev, A. and Shoshani, A. (1993). A temporal data model based on time sequences. In (Tansel et al., 1993), pages 248 – 270.

Selman, B., Levesque, H., and Mitchell, D. (1992). Hard and easy distribution of SAT problems. In *Proceedings of the tenth national conference on artificial intelligence (AAAI-92)*, pages 459– 465, San Jose.

Shen, W. M. (1991). Discovering regularities from large knowledge bases. In *Machine Learning: Proceedings of the eigth international conference*, pages 539 – 543. Morgan Kaufmann.

Silva, A. M. and Melkanoff, M. A. (1981). A method for helping discover the dependencies of a relation. In Gallaire, H., Minker, J., and Nicolas, J. M., editors, *Advances in Data Base Theory*, volume 1, pages 115–133, New York, U.S.A. Plenum Press.

Smith, B. M. and Grant, S. A. (1994). Sparse constraint graphs and exceptionally hard problems. Technical Report 94.36, School of Computer Studies, University of Leeds.

Soutou, C. (1998). Relational database reverse engineering: Algorithms to extract cardinality constraints. *Data & Knowledge Engineering*, 28:161–207.

Srikant, R. and Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. In Apers, P., Bouzeghoub, M., and Gardarin, G., editors, *Advances in Database Technology - EDBT '96, 5th International Conference on Extending Database Technology, Avignon, France, March 25-29, 1996*, volume 1057 of *Lecture Notes in Computer Science*, pages 3 – 17. Springer - Verlag.

Tansel, A. U., Clifford, J., Gadia, S., Jajodia, S., Segev, A., and Snodgrass, R., editors (1993). *Temporal Databases: Theory, Design and Implementation.* Benjamin / Cummings.

Toivonen, H. (1996). Sampling large databases for association rules. In *22nd International Conference on Very Large Databases (VLDB'96)*, pages 134 – 145, Mumbay, India. Morgan Kaufmann.

Toivonen, H., Klemettinen, M., Ronkainen, P., Hätönen, and Mannila, H. (1995). Pruning and grouping of discovered association rules. In *MLnet Workshop on Statistics, Machine Learning, and Discovery in Databases*, pages 47 – 52, Heraklion, Crete, Greece.

Tuomela, R. (1978). Theory-distance and verisimilitude. *Synthese*, 38:213 – 246.

Ullman, J. D. (1988). *Principles of Database and Knowledge-Base Systems*, volume 1. Computer Science Press, Rockville, MD.

Vadaparty, K. and Naqvi, S. (1995). Using constraints for efficient query processing in non-deterministic databases. *IEEE Transactions on Knowledge and Data Engineering*, 7(6):850 – 864.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, 27(11):1134 – 1142.

van Bommel, P. (1993). A randomised schema mutator for evolutionary database optimisation. *Australian Computer Journal*, 25(2):61 – 69.

Van Hentenryck, P. (1989). *Constraint Satisfaction in Logic Programming*. MIT Press, Massachusetts.

Vianu, V. (1987). Dynamic functional dependencies and database aging. *Journal of the ACM*, 34(1):28–59.

Vianu, V. (1988). Database survivability under dynamic constraints. *Acta Informatica*, 25:55–84.

Weigend, A. S. and Gershenfeld, N. A., editors (1994). *Time Series Prediction: Forecasting the Future and Understanding the Past*, volume XV of *SFI Studies in the Sciences of Complexity*. Addison-Wesley.

Wijsen, J. (1995). Design of temporal relational databases based on dynamic and temporal functional dependencies. In (Clifford and Tuzhilin, 1995), pages 61 – 76.

Wong, S. K. M. and Ziarko, W. (1986). On learning and evaluation of decision rules in the context of rough sets. In *Proceedings of the International Symposium on Methodologies for Intelligent Systems*, pages 308 – 324, New York. ACM Press.

Ziarko, W. (1991). The discovery, analysis, and representation of data dependencies in databases. In (Piatetsky-Shapiro and Frawley, 1991), pages 195–211.

APPENDIX A

The Consistency Problem: Supplemental Results

We now provide additional results for the consistency problem. In Table A.1 we detail the FD sets referred to in the following figures. All of the mean values referred to for the average number of worlds required were obtained within batchs, each of 500 runs. The results in this appendix reinforce those presented in Chapter 4. All relations used, with respect to the FD sets in Table A.1, contain exactly those attributes within the respective FD set as the schema and no more.

Set 1	Set 2	Set 4	Set 6	Set 7	Set 11	Set 15	Set 17
$A \rightarrow B$ $D \rightarrow C$	$A \rightarrow BC$ $D \rightarrow C$	$C \rightarrow AB$ $B \rightarrow AC$	$D \rightarrow ABC$ $AB \rightarrow D$ $A \rightarrow B$ $B \rightarrow A$	$AB \rightarrow D$ $D \rightarrow ABC$	$A \rightarrow B$ $D \rightarrow C$ $BC \rightarrow A$	$A \rightarrow BCD$	$A \rightarrow B$ $B \rightarrow C$ $C \rightarrow D$

Table A.1: FD sets used in Figures A.1 to A.21

For an FD set $X \rightarrow Y$, where $|\ Y\ | > 1$, we split this into a set of FDs such that for all $A \in Y$ we have $X \rightarrow A$ for expression as NDs in simulations. This is justified given that from $X \rightarrow^k Y$ we can infer, for all $A \in Y$, $X \rightarrow^k A$.

A.1 Average Number of Worlds Required

We present examples showing the average number of worlds required in batches by our chase and hill-climbing algorithm in figures A.1 to A.6. We can see immediately that the average number of worlds required is very small. We hypothesise that within our random relations it is relatively easy to generate a definite world using algorithms 12 and 13. The average number of worlds is reduced for larger relations with respect to a fixed domain size. Investigation has shown this to be due to ND sets being satisfied closer to the domain size, whilst the presence of additional tuples, with respect to a fixed domain size, increases the number of redundant values within indefinite cells which the chase procedure can remove. This is particularly true of relations with larger ar-

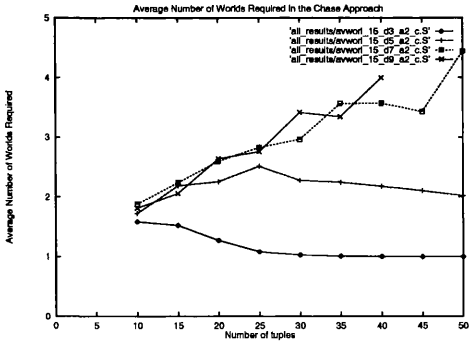

Figure A.1: Average Number of Worlds Required by the chase and hill-climbing approach for FD set 15, domain sizes 3 - 9, maximum indefinite cell arity 2

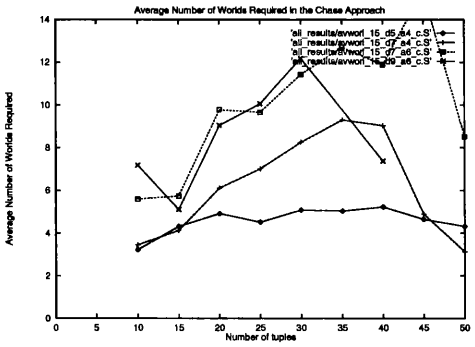

Figure A.2: Average Number of Worlds Required for FD set 15, domain sizes 5 - 9, max indefinite arity 4 - 6

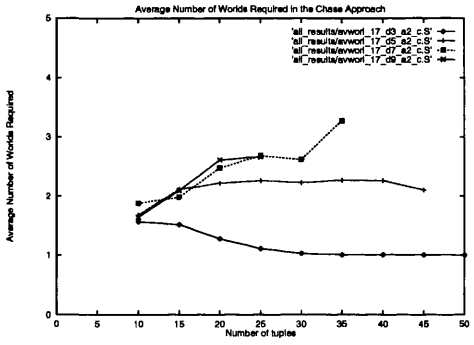

Figure A.3: Average Number of Worlds Required for FD set 17, domain sizes 3 - 9, max indefinite arity 2

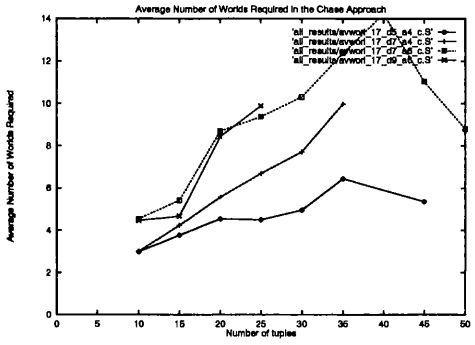

Figure A.4: Average Number of Worlds Required for FD set 17, domain sizes 5 - 9, indefinite cell arity 4 - 6

ity indefinite cells, see figures A.2, and A.8. Figures A.3 and A.7 show this is less likely when relations have smaller arity indefinite cells.

We also note that the small number of average worlds is in sharp contrast to the number of worlds required by the naive *generate and test* algorithms to obtain similar results. We are vague as to an exact relationship due to the varying nature of both the indefinite relations and FD sets.

A.2 Average Proximity to FD sets

We now discuss the proximity of our results to that of an FD set. The increasing proximity to an FD set as relation size increases is due to the domain size remaining fixed. Normalisation of our measure for ND sets, a prospect for future work, would remedy this.

We draw the following conclusions concerning proximity:

- On average the naive and chase procedures produce very similar results. We emphasise

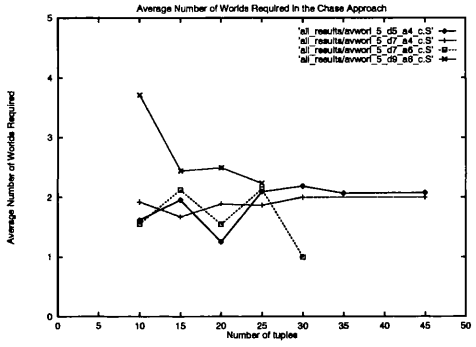

Figure A.5: Average Number of Worlds Required for FD set 5, domain size 5 - 9, indefinite cell arity 4 - 6

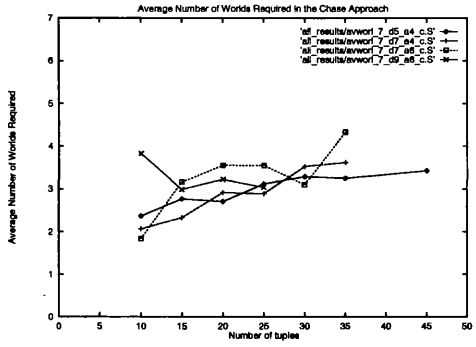

Figure A.6: Average Number of Worlds Required for FD set 7, domain size 5 - 9, indefinite cell arity 4 - 6

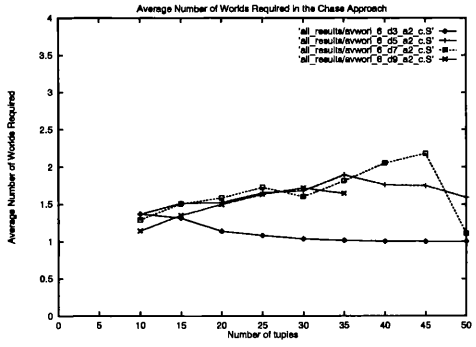

Figure A.7: Average Number of Worlds Required for FD set 6, domain size 3 - 9, indefinite arity 2

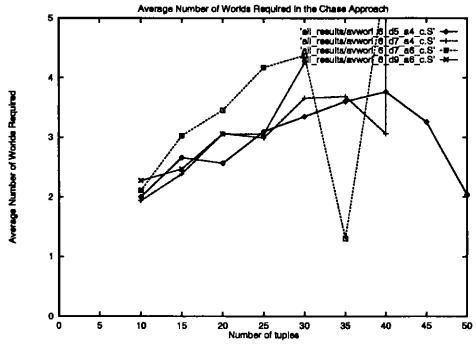

Figure A.8: Average Number of Worlds required for FD set 6, domain size 5 - 9, indefinite arity 4 - 6

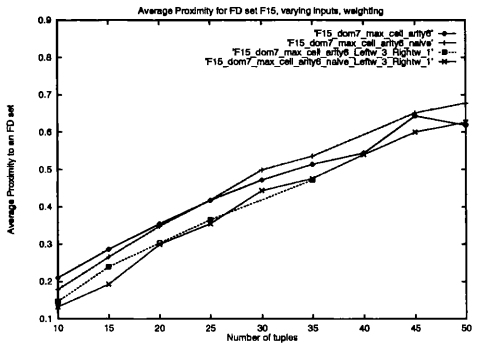

Figure A.9: Average Proximity to FD set 15, standard and reduced right hand side indefinite cell weighting

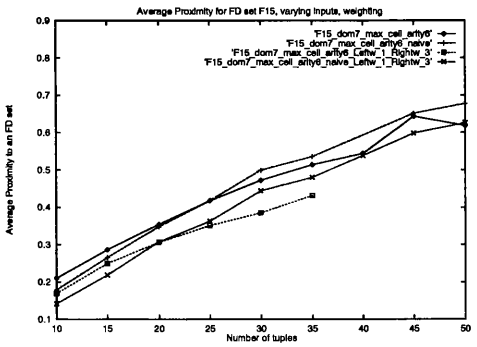

Figure A.10: Average Proximity to FD set 15, standard and reduced left hand side indefinite cell weighting

that our relations were created in a uniformly random manner. As such in the real world it may be highly likely that a relation with indefinite information may perform better with respect to utilising the chase and hill-climbing approach. This speculation is enforced by the encouraging discovery that the chase procedure produce slightly better results when a relation is sparse in indefinite cells in arity in either attributes on the left or right hand side of dependencies, as detailed in figures A.9 to A.12.

- Proximity to functional satisfaction increases, on average, with an increase in the number of attributes being determined. We can see these differences in figures A.15 and A.16. The difference in this case is slight but this was enforced by all results. This is due to additional attributes being examined within the hill-climbing process. The random nature of the relations generated did not provide us with any pathological data which might contradict this.

- Whether the relation is in BCNF or non-BCNF did not, in the data assessed, affect the results. We did not expect this to be otherwise.

A.3 Closest Proximity to FD sets

In contrast with the average results we find that, generally, the best result within a batch is obtained by our chase and hill-climbing procedure. Figures A.11 and A.12, as well as A.15 and A.16, serve to emphasise that occasionally the naive procedure, at a cost of efficiency, can outperform the chase and hill-climbing procedure.

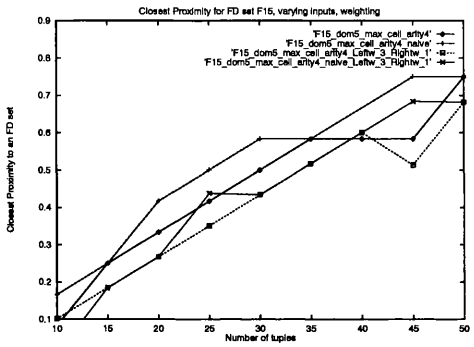

Figure A.11: Closest Proximity to FD set 15 for standard and reduced right hand side weighting of indefinite cells

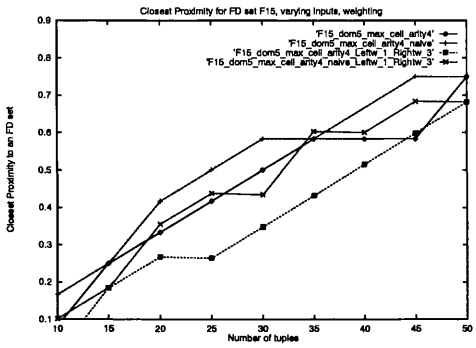

Figure A.12: Closest Proximity to FD set 15 for standard and reduced left hand side weighting of indefinite cells

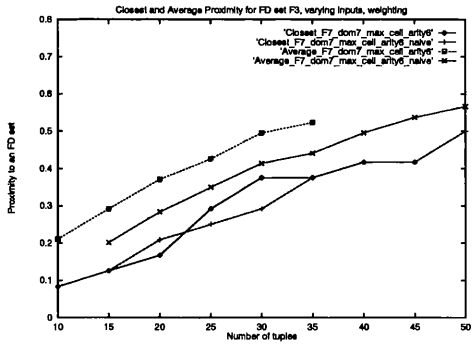

Figure A.13: Average Proximity to FD set 7, domain size 7, max indefinite cell arity 6

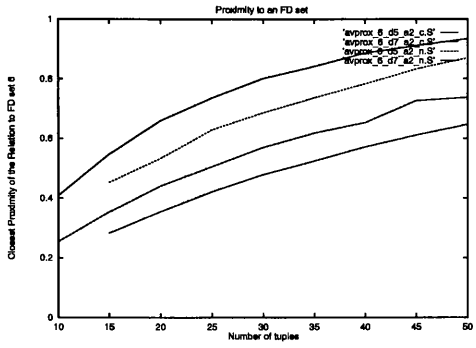

Figure A.14: Average Proximity to FD set 6, domain 5,7, indefinite arity 2

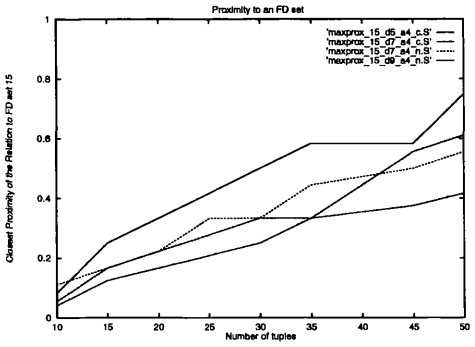

Figure A.15: Closest Proximity to FD set 15, varying domain sizes 5 - 9, chase and naive approaches, indefinite arity 4

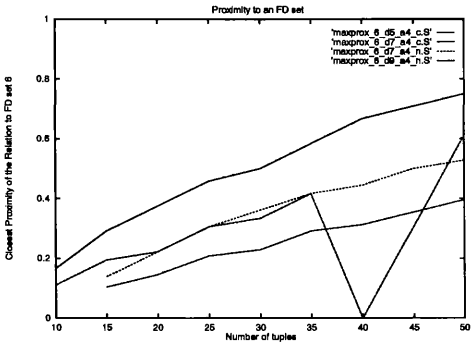

Figure A.16: Closest Proximity to FD set 6, domain size 5 - 9, indefinite arity 4

A.4 Jacknife and Bootstrap Comparisons

Figures A.17, A.18 and A.19 present examples of jackknife and bootstrap resampling used within our dynamic algorithm 10. To ensure a fair comparison we conducted these tests so that at each iteration each sample of possible worlds, and therefore each sample of ND sets satisfied, was equivalent before either bootstrap or jackknife resampling was performed. This accounts for much of the similarity in each figure.

We draw the following conclusions:

- Jackknife and Bootstrap resampling will reach approximate fixpoints, on average, at a similar number of possible worlds. Figures A.17, A.18 and A.19 are indicative of this.

- The jackknife resampling technique is more computationally intensive in such a dynamic setting. The size of the bootstrap replication is fixed, say at 50 or 100, found to be useful in this context. (Efron and Tibshirani, 1986; Efron and Tibshirani, 1993) note that sizes about 200 produce no additional information, in general. However, the jackknife procedure creates n replicates, each of size $n - 1$, when the sample size is n. Therefore we have to examine 299 jackknife resamples at sample size 300 as opposed to 100 for the bootstrap. The results show this to be sufficient to infer a suitable standard deviation, variance, or mean.

- Jackknife resamples are slightly smoother due to the fact that we are merely omitting one point in each resample. Though this may imply it is likely to reach a fixpoint at an earlier stage, results do not suggest this, validating, in some sense, our approach.

Figure A.20 highlights convergence of the standard deviation and variance as the sample size is increased. We are dealing with approximate fixpoint and this implies equality within confidence

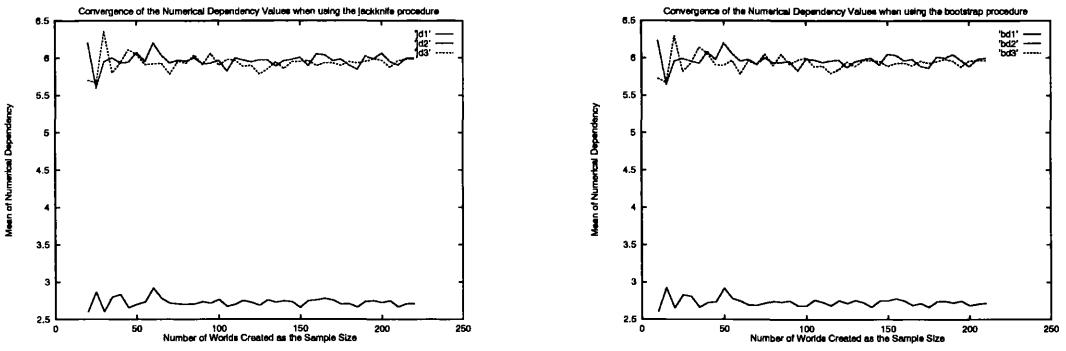

Figure A.17: A comparison of Jackknife and Bootstrap mean ND set values iterated to an approximate fixpoint of the mean using equivalent samples, for FD set 11, with a domain of 10, 50 tuples and a maximum indefinite cell arity of 3

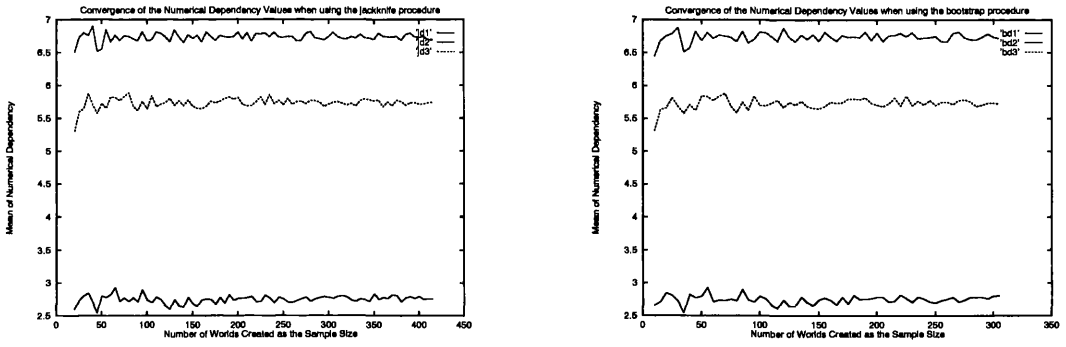

Figure A.18: A comparison of Jackknife and Bootstrap mean ND set values iterated to an approximate fixpoint of the mean using equivalent samples, for FD set 11, with a domain of 10, 50 tuples and a maximum indefinite cell arity of 5

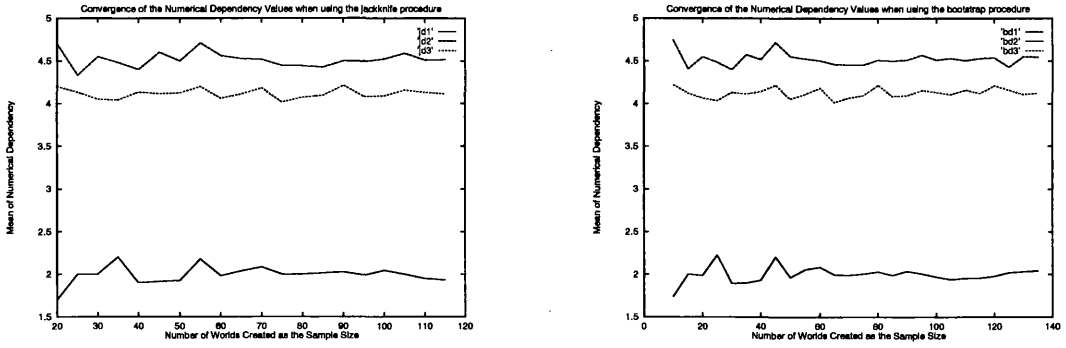

Figure A.19: A comparison of Jackknife and Bootstrap mean ND set values iterated to an approximate fixpoint of the mean using equivalent samples, for FD set 11, with a domain of 10, 25 tuples and a maximum indefinite cell arity of 3

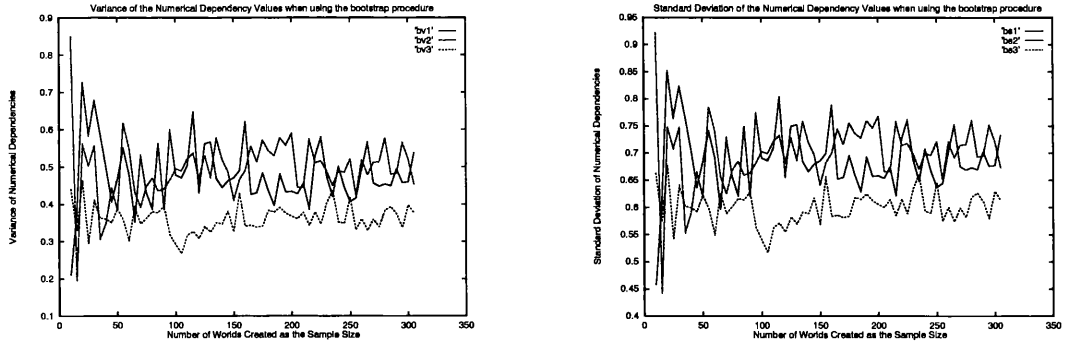

Figure A.20: Bootstrap variance and standard deviation convergence, for FD set 11, with a domain of 10, 25 tuples and a maximum indefinite cell arity of 3

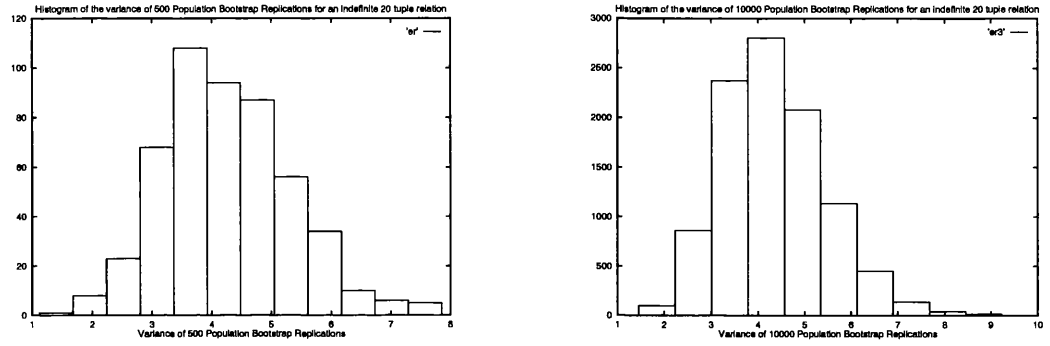

Figure A.21: Histograms displaying variance of 500 and 10000 bootstrap replications

limits shown in Figure 4.14.

A.4.1 Bootstrap Variance Results

Figure A.21 display the overall similarity in variances achieved for 500 and 10000 BRS, respectively, complementing Figure 4.13.

APPENDIX B

Simulation Methodology

We now describe our process for conducting experiments, expanding the outlines given in Chapters 3, 4, and 6, demarcated into sections on evolving relations, the consistency problem, and simulations on our temporal logic, respectively.

The code was implemented in GNU C++ version 2.7.2 on a UNIX platform running Sun Solaris 2.5.1. C++ with the embedded CORAL deductive database interface (Ramakrishnan et al., 1992) was also used for evolving relations in Chapter 3. For efficiency reasons the code for procedures described in Chapters 4 and 6 was implemented in C++ alone.

B.1 Simulation Details: Evolving Relations

B.1.1 Simulation Range Decisions

We selected 72 FD sets many of which originated from a number of well known DB texts including (Mannila and Räihä, 1992a; Abiteboul et al., 1995; Atzeni and De Antonellis, 1993). These sets were divided into BCNF and non-BCNF for investigative purposes. Given that an Armstrong Relation can only be generated for a set of FDs F when the relation size has at least | GEN(F) | + 1 tuples, where GEN(F) is defined in Definition 2.2.16, then we chose to vary the domain and tuple sizes from $G/2$ to G and $G/2$ to $3G$, respectively. This allowed for the scale of the randomly generated relations to be related to F, as well as ensuring that we would have a good chance of finding an AR for each FD set. This choice was justified by finding ARs in 63 out of our 72 selected FD sets. We created a batch of 1000 runs, the process of evolving a randomly generated relation to absorption, for each domain and tuple combination. 1000 runs allowed us to find reliable averages for each domain and tuple combination.

Random Relations were created by random number selection within a uniform distribution to prevent unwanted discrepancies in ND set satisfaction. For smaller domain sizes a normal distribution in random relations would often lead to relations satisfying FD sets with few steps to

absorption as there are likely to be fewer partitions on attributes on the left hand side of the FDs and fewer differences between attributes values on the right hand side of the FDs.

B.1.2 Use of Random Number Generation

A number of algorithms in this thesis use randomised techniques. To circumvent any potential problems with non-random behaviour we used a linear congruent procedure taken from the algorithm provided by Park and Miller (Park and Miller, 1988) which avoids cycles by incorporating multiplier and modulus having 534 million full period generators.

B.1.3 C++ libraries

The program, a direct implementation of Algorithm 6, was written in C++ with the embedded CORAL deductive database C++ interface to manipulate the relations. Each randomly generated relation was created and stored as a database in CORAL.

Via the C++ interface in CORAL, using functional and numerical dependency classes and a partition class for the tuples, the relation is then mutated according to the uniform random selections made in the algorithm. C++ with embedded CORAL was also used for assessing the quality of the relations after evolution, the knowledge discovery component of our system.

B.2 Simulation Details: The Consistency Problem

For this work we concentrated on 12 FD sets, detailed in Appendix A and Chapter 4. Again, for coverage these were demarcated in BCNF and non-BCNF sets. Our simulation details are presented in Table 4.3. The 12 FD sets range from containing a small to a significant number of dependencies, 8 of which are presented in Table A.1. Those FDs not discussed directly within the text provided results subsumed by those FD sets which are presented.

For each domain, tuple and maximum indefinite-cell arity we ran a batch containing 500 runs. A batch was run for both naive and the chase and hill-climbing instances of the program. Again these batches allowed us to infer acceptable mean behavior for each input combination.

B.2.1 Indefinite Information Data

The lack of availability of real-world data containing indefinate information dictated our use of randomly generated data. Though there are cases, as we have seen in Chapter 4, where it would be useful to represent disjunction within cells, current RDBMS systems do not generally support anything more than the ability to store NULL values. This also applies to deductive databases, as experienced by our use of the CORAL deductive database. Due to this we chose to conduct our experiments on randomly generated relations with a uniform distribution of values across a given domain size.

Further simulations were conducted where attributes on the left hand side or right hand side of FDs were specified as containing indefinite cells with either a low, medium, or high probability of containing indefinite information, as detailed in Table B.1. This allowed us to study the behaviour of indefinite information in sparsely generated random relations, sparsity being elaborated upon in Definition B.2.1. This direct control over the presence of indefinacy within a randomly generated relation created with a uniformly random distribution was preferable to that of a random relation created with a normal distribution, giving us direct control over the relationship between indefinacy in cells and their appearance in either the left or right hand side of an FD.

Sparsity	
LOW	25% probability of indefinite cell
MEDIUM	50% probability of indefinite cell
HIGH	75% probability of indefinite cell

Table B.1: Depicting the range of indefinite cells in a relation

Definition B.2.1 (Sparsity) Sparsity is defined to be the fraction of indefinite cells within a relation. If relation R has m tuples and n attributes such that it is of size m x n and there are k indefinite cells in the relation then its sparsity is $\frac{k}{mn}$. For example a relation with 20 tuples and 5 attributes will have a low, medium, or high sparsity with 25, 50, and 75 indefinite cells respectively. □

B.2.2 A note on randomly generated relations

The use of randomly generated relations places a limit on the size of the relations which we can use. For example, as shown in Table 4.3, we restricted relation size to 50 tuples. Though this is small given the requirement of a fixed domain size any increase in relation size is likely to lead to all randomly created relations satisfying the given FDs as NDs with the branching factor equivalent to the domain size in all possible worlds.

We stress that though these relations are small there is generally a significant number of possible worlds to select from. In a randomly generated relation with each cell having a 50% chance of being indefinite, far higher than likely in the real world, a relation with 50 tuples, 4 attributes and a maximum indefinite cell size of 4 has a maximum $4^{50.4}$ possible worlds. A much larger relation, say with 1,000 or 10,000 tuples but with only 20 indefinite cells, none with more than 4 items in any indefinite cell, would have 4^{20} possible worlds. Larger relations in such a case would not have been any more comprehensive with regard to results concerning our use of the bootstrap.

B.2.3 Bootstrap Parameter Size Selection

The Bootstrap Replication Size (BRS), B, is the number of times we resample from a sample. As discussed in Section 4.3.3 we restate, from (Efron and Tibshirani, 1986), that there is *little improvement* setting B above 100. We decided to conduct a number of tests with B starting at 25 and approximately increasing B by a factor of 2 until we reached B = 10,000 on relations. Our tests on a suitable BRS were conducted on two relations presented together in Table B.2 with Attribute A as the only left hand side for the FDs guaranteeing FD violation. These relations contain every cell as indefinite implying that the variance within each resample would be much higher than for usual implying that our conclusions on a suitable BRS would be robust for a randomly generated relation. Empirical results allowed us to conclude that setting B = 100 would provide reliable resampling results. Figure A.21 emphasises the minimal difference in variance between 500 and 10000 resamples; a similar result was also found for 100 resamples.

A	$\mathbf{B_1}$	$\mathbf{B_2}$	...	$\mathbf{B_{n-1}}$	$\mathbf{B_n}$
$[2,3]$	$[1,2,3]$	$[1,2,3]$	...	$[1,2,3]$	$[1,2,3]$
$[2,3]$	$[4,5,3]$	$[4,5,3]$	...	$[4,5,3]$	$[4,5,3]$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$[2,3]$	$[n_{m-3}, n_{m-2}, 3]$	$[n_{m-3}, n_{m-2}, 3]$	...	$[n_{m-3}, n_{m-2}, 3]$	$[n_{m-3}, n_{m-2}, 3]$
$[2,3]$	$[n_{m-1}, n_m, 3]$	$[n_{m-1}, n_m, 3]$	...	$[n_{m-1}, n_m, 3]$	$[n_{m-1}, n_m, 3]$

Table B.2: Indefinite relations r_1 with 10 tuples when $m = 21$ and r_2 with 20 tuples and $m = 41$

B.2.4 Use of the Original Sample and Fixpoint Selection

We experimented with using the original indefinite relation for each resampling iteration from which n possible worlds are sampled each time. The variance is much higher in this case as we have all possible worlds to select from for each sample of size n. These experiments were conducted on 5 batches for different domain, tuple and indefinite cell-arity combinations.

Initially we experimented with using our dynamic resampling algorithm, WORLD_LIMIT, with different degrees of approximate equality for out statistical estimators (standard deviation, variance). We found that 1 decimal place to provide too many *false* convergence results, though the averages within a batch were similar to those for 2 decimal places. We chose to use 2 decimal places as our degree of approximation in these tests.

B.2.5 Using the Bootstrap to determine confidence intervals

Based on the values generated by the Bootstrap samples we used the generally accepted assumption that as the Bootstrap replication size increases the sampling approximates a normal distribution and so we can actually determine the confidence intervals empirically, shown to converge for

a relation in Figure 4.14. For example to find the 95% confidence intervals we determine what the values of the parameter of interest are within the ordered B Bootstrap samples at the 25th and 975th points for the replications with B = 1000. The extra information provided by confidence intervals for a sample implies that they need more computational effort, as remarked in (Efron and Tibshirani, 1993).

The bootstrap could also be used to find the distribution of *good* approximations to the FD set. An example of this may be that NDs such as $AB \rightarrow^3 CD$ are found in the $\bar{s}(\tilde{p}_b^\star) + 2.\hat{se}_B$ to $\bar{s}(\tilde{p}_b^\star) + 3.\hat{se}_B$ range of the distribution which contains 2.1% of a normal distribution, and are therefore considered *good* for the indefinite relation in question. In this case it will be unlikely to achieve anything better other than by an exhaustive search which is impossible. Possible problems associated with this are that it is too naive to tell us anything when there may be very few *good* approximations such as in r_2, shown in Table B.2. We chose not to apply this technique.

Alternatively we can use the standard error for the Bootstrapped sample to assign approximate confidence intervals given the bootstrapped estimate of standard error $\hat{se}_B$ and the bootstrapped variance $\hat{\theta}$ whilst assuming a normal distribution. For example, we determine a 95% confidence interval as $\hat{\theta} \pm 1.960 \cdot \hat{se}_B$.

B.2.6 Jackknife and Bootstrap Resampling

For 3 FDs an additional batch of simulations were run, using jackknife resampling in addition to bootstrap resampling for 10 different domain/tuple/indefinite-cell arity combinations. We enforced that each step of resampling would have the same original sample in each case for more comprehensive comparisons. The only difference was that for jackknife resampling our statistical estimators would be based on n resamples for a sample of size n and for 100 resamples for bootstrapped resamples. This implied that whenever the number of worlds in a sample exceeded one hundred in our algorithm, WORLD_LIMIT, that the bootstrap became more computationally efficient.

B.3 Simulation Details: Numerical Dependency Temporal Logic

For our simulations on property discovery we used real world data downloaded from the following sources:

- Statlib, a data set resource, `http://lib.stat.cmu.edu`
- Financial Data sets are available in the public domain from a number of sources, we cite `http://www.market-eye.co.uk` and `http://www.moneyworld.co.uk`

We implemented our objects within template classes in C++ using the classes for functional and numerical dependencies created for our work on evolving relations and the consistency problem. A sequence object (class) was created to hold the temporal sequence data. This was implemented as a template in C++ to allow data independence. This object contained the required time series function values which would allow us to difference and create moving averages of the time series. Additionally it included a sign for the trend, initial and maximum values within the sequence. Moving Block procedures were also implemented upon this object. Statistical functions such as correlations, variance and covariance were also implemented in this class. These time series functions were verified for correctness via testing and comparison of results presented in (Kendall and Ord, 1990). Based on the results of our procedures we are able to add the modal operators $\boxminus^n$ and $\diamondminus^n$ to a sequence so that it may represent a potentially interesting property within our discovery model.

B.3.1 Sequence Size Selection

Input was a Time Series and two sequence sizes. Given Theorem 5.5.4 we know that for fixed small and large sequence sizes the discovery of properties can be achieved in polynomial time.

We conducted our experiments with a small sequence size n and large sequence size $2n$. We increased n by a factor of 2 until it was considered too large to provide meaningful results with respect to the sequence size at hand, when n is over half the size of the complete temporal relation sequence implying that all sequences overlap by at least one point, discussed in Section 6.6.1.

Often we changed the sequence sizes based upon the presence or absence of response and persistence rules as discussed in Section 6.8. When the small sequence size n is much smaller than the large sequence size m then we are unlikely to find a response rule and if they are nearly the same size we are guaranteed to find one due to overlapping of sequences. A similar criteria holds for persistence rules. We found our use of 1:2 as an initial ratio to provide interesting results, though we freely changed this when results from simulations suggested so. This highlights the interactive nature of these simulations, stressed in much data mining research.

B.3.2 Moving Average and Moving Block Size Selection

Moving Averages ranged from 3 to 10 as our simulation shows. This was to avoid excessive smoothing with a temporal relation sequence.

Experiments were conducted on a range of different moving block sizes. These were arbitrary choices based on our goal to obtain resampled sequences which preserved relationships within the blocks. The choice of block size in our use of the moving block bootstrap for large relations depended on our goal of whether we are seeking to discover properties relating to ei-

ther grouping of short ranges or over a *synopsis* of the original sequence. Short range behaviour will be found if we select fewer blocks of a larger size whilst long range behaviour is found by selecting more blocks of a smaller size with respect to the size of the original temporal relation sequence.

INDEX