Restoring Consistency in Ontological Multidimensional Data Models via Weighted Repairs

By Enamul Haque,

A Thesis Submitted to the School of Graduate Studies in the Partial Fulfillment of the Requirements for the Degree Master of Science

McMaster University
Master of Science (2020)
Hamilton, Ontario (Computing and Software)

TITLE: Restoring Consistency in Ontological Multidimensional Data Models via Weighted Repairs
AUTHOR: Enamul Haque (McMaster University)
SUPERVISOR: Dr. Fei Chiang
NUMBER OF PAGES: xi, 59

Abstract

High data quality is a prerequisite for accurate data analysis. However, data inconsistencies often arise in real data, leading to untrusted decision making downstream in the data analysis pipeline. In this research, we study the problem of inconsistency detection and repair of the Ontology Multi-dimensional Data Model (OMD). We propose a framework of data quality assessment, and repair for the OMD. We formally define a weight-based repair-by-deletion semantics, and present an automatic weight generation mechanism that considers multiple input criteria. Our methods are rooted in multi-criteria decision making that consider the correlation, contrast, and conflict that may exist among multiple criteria, and is often needed in the data cleaning domain. After weight generation we present a dynamic programming based Min-Sum algorithm to identify minimal weight solution. We then apply evolutionary optimization techniques and demonstrate improved performance using medical datasets, making it realizable in practice.

Acknowledgements

I cannot begin to express my gratitude to my parents for their sacrifices, encouragement and caring throughout my life. After the loss of my father, my mother single-handedly took care of the whole family and always kept on inspiring me for my higher studies, so here I am!

I wish to express my sincere appreciation to my supervisor Dr. Fei Chiang, who convincingly guided me in doing research and encouraged me to be professional and do the right thing even when things were not going smooth. I also gratefully acknowledge her assistance in publishing and presenting my research work at a conference.

I would also like to extend my gratitude to Prof. William M. Farmer and Prof. Emil Sekerinski for their insight, support and sharing of knowledge that have made this thesis possible. No matter when I requested a meeting with them, they managed time for discussions from which I learned a lot to overcome challenging areas of this research. Special thanks to Dr. Mostafa Milani also for his support in literature review and showcasing application of this work. I must also thank Prof. Franya Franek and Prof. Ryszard Janicki to be in my examination committee and providing me with valuable comments during the defense. I cannot leave McMaster University without mentioning Peter Yang, who was instrumental in dotting the i's and crossing the t's for my thesis manuscript.

I must express my profound acknowledgement to my wife and my son for their understanding and always being there for me through thick and thin to carry on such a huge project throughout all these years.

Last but not least, to all my relatives, friends, teachers, lab-mates and colleagues that helped me directly or indirectly to grow as a person and were always there for me during the good and bad times in my life, thank you very much!

Contents

Abstract **iii**

Acknowledgements **iv**

Declaration of Authorship **xi**

1 Introduction **1**
- 1.1 Data Quality 1
- 1.2 Inconsistency and Repair 3
- 1.3 Contributions 5

2 Background **6**
- 2.1 Data Dependencies 6
- 2.2 OMD Model 9
- 2.3 Multi-Criteria Decision Making 14
- 2.4 CRITIC Method 15
- 2.5 Genetic Algorithms 17

3 Related Work **20**
- 3.1 Overview 20
- 3.2 Consistent Query Answering 20
- 3.3 Priorities for Consistency 21
- 3.4 Database Repair 22

4 Consistency Restoration via Weighted Repairs **24**
- 4.1 Solution Overview 24
- 4.2 Inconsistency Detection 25
  - 4.2.1 Grounding IDB 25
  - 4.2.2 Source of Inconsistency 27
- 4.3 Weight Generation 28
  - 4.3.1 Deletion Criteria 28
- 4.4 Minimal Weighted Repair 31
  - 4.4.1 Min-Sum a Dynamic Programming Based Approach 31
  - 4.4.2 A Genetic Algorithm Based Approach 38

5 Experimental Evaluation **43**
- 5.1 System Configuration 43

5.2 Datasets 43
5.3 Source of Inconsistency 45
5.4 Weight Generation 45
5.5 Deletion Candidate Search 46

6 Conclusion and Future Research **50**

A System and Program **52**
A1 System Configuration Details 52
A2 Program and Source Codes 53

List of Figures

1.1 “Person” Dimensional Schema . . . 2
1.2 Tuple Generating Dependencies from table “AdmDrug” to table “Bills” . 3
1.3 Drug (a,b) and Person (c,d) Dimension Entity: Dimensional Schema (a,c) and Dimensional Instance (b,d). . . . 4

2.1 Doctors tuples . . . 6
2.2 TGD generating tuples . . . 7
2.3 Denial Constraint found an inconsistency . . . 8
2.4 Drug (a) and Person (b) Dimensional Schema . . . 9
2.5 AdmDrug and Bills categorical predicates schema and instance . . . 10
2.6 Drug(a) and Person(b) Dimensional Instance . . . 11
2.7 TGD rule generating tuple from table “AdmDrug” to table “Bills” . . . 12
2.8 Inconsistent Tuples . . . 13
2.9 Genetic Algorithm Steps . . . 18

4.1 OMD Model Consistency Restoration Process. . . . 25
4.2 Weight Generated by CRITIC method . . . 30
4.3 Algorithm-3 Steps in the matrix . . . 35
4.4 Algorithm-3 Steps in the matrix . . . 37

5.1 Drug (a) and Person (b) Dimension (with # of instances) . . . 44
5.2 Dataset Tables (with # of instances) . . . 44
5.3 Source of Inconsistency Search Performance . . . 45
5.4 Weight Generation . . . 46
5.5 Greedy and DP Based Algorithm Performance . . . 46
5.6 Total Number of Set Generated to Find Solution in Search Space . . . 47
5.7 Subsets, Tuples and Time Relation . . . 48
5.8 Genetic Algorithm Iterations for Minimal Weight Search . . . 48

List of Tables

2.1 Inconsistent Predicates 13

List of Abbreviations

OMD	**O**ntological **M**ultidimensional **D**ata
MD	**M**ulti-**D**imensional
EGD	**E**quality **G**enerating **D**ependency
EDB	**E**xtensional **D**ata**B**ase
TGD	**T**uple **G**enerating **D**ependency
IDB	**I**ntensional **D**ata**B**ase
NC	**N**egative **C**onstraint
DC	**D**enial **C**onstraint
HM	**H**urtado-**M**endelzon
MCDM	**M**ulti-**C**riteria **D**ecision **M**aking
CRITIC	**CR**iteria **I**mportance **T**hrough **I**nter-critera **C**orrelation
GA	**G**enetic **A**lgorithm
SAT	**SAT**isfiability

List of Symbols

ϕ, ψ, φ	Conjunction of Atomic Formulas
R^M	OMD Schema
H	Relational Schema
K	Set of Unary Category Predicates
L	Set of Binary Predicates
R^C	Set of Categorical Predicates
I^M	OMD Instance
D^H	Instance of Relational Schema
I^C	Instance of Categorical Predicates
σ	Rule
η	Constraint
ψ	Set of Ground Atoms
Ω	Set of Rules
μ	Set of Anomaly Predicates
W	Set of Weights
S	Sum of Weights
$\wedge$	Logical And
$\vee$	Logical Or
$\rightarrow, \Rightarrow$	Logical Implication
$\forall$	For all
$\exists$	There Exists
$\perp$	Falsum or False
$\in$	Is Member Of (Set)
$\Leftarrow$	Variable Assignment
$\models$	Model Of / Satisfies
$\not\models$	Not Model Of / Does Not Satisfy
$\setminus$	Set Subtraction
$\subseteq$	Subset Of
$\varnothing$	Empty Set

Declaration of Authorship

I, Enamul Haque, declare that this thesis titled, “Restoring Consistency in Ontological Multidimensional Data Models via Weighted Repairs” and the work presented in it are my own.

Chapter 1

Introduction

Data is changing the face of the world by vitalizing creation of new drugs to fight diseases, increasing company revenues, optimization of costs, targeted advertisements or precise prediction of weather. With computers becoming increasingly powerful, high speed networks and algorithms working on vast amount of data providing competitive advantage and plethora of benefits to industry and academia. This can only be useful if the data is of desired quality; otherwise, they can be misleading or even dangerous. "Garbage in, garbage out" applies here. The quality of the input data strongly influences the quality of the results produced. In the field of data management and knowledge representation, data quality, data cleaning and consistent query answering are critical tasks but quite challenging, resulting in costly problems if not handled properly [1]–[3].

1.1 Data Quality

The concept of data quality comprises different definitions and interpretations in two main research communities: databases and management. While both communities are interested in data cleaning, the database community mostly focuses on it from a purely technical perspective whereas the management community faces the additional challenge of assessing data quality in relation to end users' needs. In short, data quality refers to the degree to which the data adheres to a form of usage [1]. A survey listing data quality attributes that capture consumers' perspectives on data quality showed 179 data quality attributes, which were subsequently summarized into 20 dimensions of 4 categories: (1) accuracy, (2) relevancy, (3) representation and (4) accessibility of data [4]. In this research we focused on technical aspects of data quality of a particular format of data (known as the Ontological Multidimensional Data Models), keeping the end user in mind [5]. To ensure the quality of data, first we detect if there is any error or nonconformity and if found, we then remove the anomaly by repairing in the best possible way. Normally data quality rules, such as integrity constraints, are used as a declarative way to detect errors and describe correct or legal data instances. Any subset of data which does not conform to the defined rules or constraints is considered erroneous, hence subject to repair.

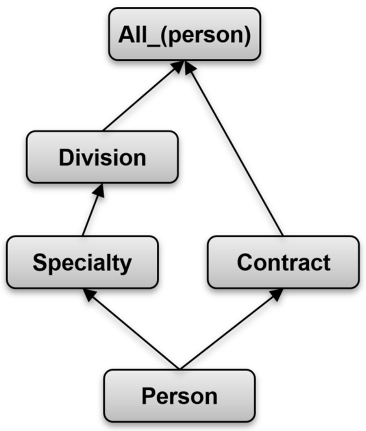
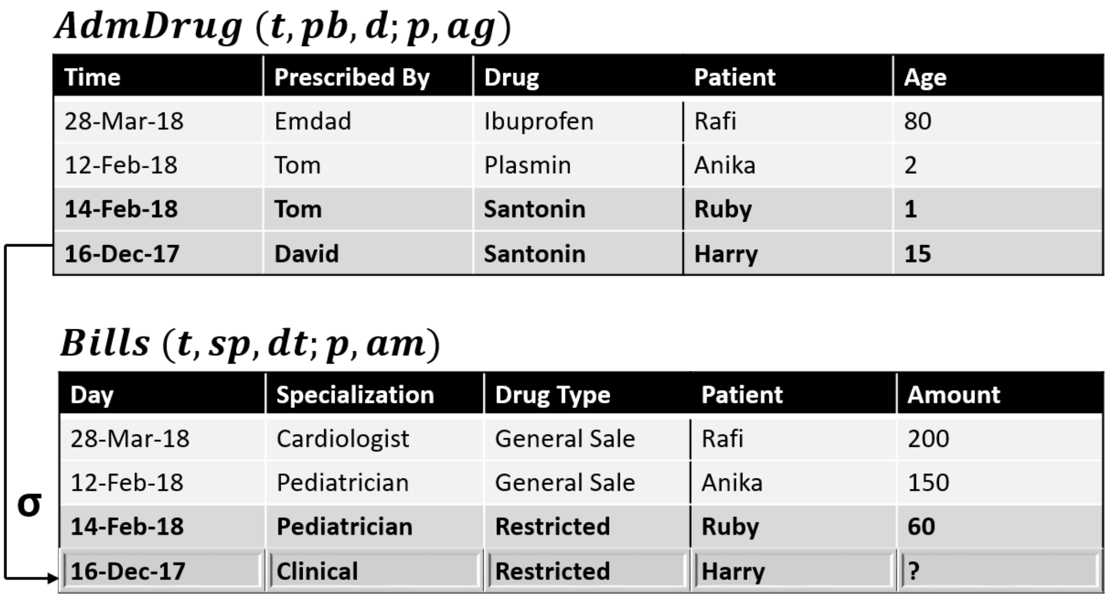


Figure 1.1: “Person” Dimensional Schema

The mechanism of data quality assessment and cleaning is often considered as a context-dependent activity [6]. Context can be external knowledge and/or connection to the external knowledge that confirm the validity of the given data items. Generally, context has been modeled as logic-based ontologies because of their semantic expressiveness [7]. These usually have to be expressive enough while keeping the computation complexity low, so that data extraction via query answering does not become intractable. A database can be expressed as a logical theory, a context for it can be another logical theory and there can be a logical mapping between them to embed the database into the contextual theory or ontology. Contextual ontologies can be realized as multidimensional (MD) ontologies, due to the multidimensional nature of contexts. These MD ontologies allow representation of dimensions as shown in the Figure: 1.1 the “Person” dimensional schema, which is similar to the multidimensional databases along with data tables under quality assessment. Dimensions of data are conceptual axes along which data are represented and analyzed. For example, any person can have attributes which can be considered as contexts to extend knowledge about the person or verify any data involving that person. Hence, adding constraints into this system eventually supports multidimensional data quality assessment [5]. Datalog$^{\pm}$ a declarative query language (extension from plain Datalog with syntactic restrictions and addition of features on the program [8]), has been widely used to define and extend dimension hierarchies with dimensional constraints, dimensional rules and to state formula for the quality data specifications. Dimensional rules and constraints are expressed in general syntactic forms of tuple generating dependencies (TGDs), equality generating dependencies (EGDs) or negative constraints (NCs), that extend classical integrity constraints. In this case, TGDs are useful in generating data values to tackle completeness problem (e.g. missing records) of data quality which is explained in the sub-section below using an example.



$AdmDrug\ (t, pb, d; p, ag)$

Time	Prescribed By	Drug	Patient	Age
28-Mar-18	Emdad	Ibuprofen	Rafi	80
12-Feb-18	Tom	Plasmin	Anika	2
14-Feb-18	**Tom**	**Santonin**	**Ruby**	**1**
16-Dec-17	**David**	**Santonin**	**Harry**	**15**

σ

$Bills\ (t, sp, dt; p, am)$

Day	Specialization	Drug Type	Patient	Amount
28-Mar-18	Cardiologist	General Sale	Rafi	200
12-Feb-18	Pediatrician	General Sale	Anika	150
14-Feb-18	**Pediatrician**	**Restricted**	**Ruby**	**60**
16-Dec-17	**Clinical**	**Restricted**	**Harry**	**?**

Figure 1.2: Tuple Generating Dependencies from table “AdmDrug” to table “Bills”

1.2 Inconsistency and Repair

The theory of ontological multidimensional data (OMD) models can become inconsistent with respect to user defined constraints over the database. Inconsistency can arise from dirty data, inconsistent quality rules, the dimensions, or the user defined constraints. The presence of inconsistency could also be detected before or after new tuple generation by TGDs.

For example, consider Figure 1.2, with two tables of a database a hospital: Administer Drug *(AdmDrug)* and Bills *(Bills)*. Administer Drug is the table which contains the information of patient, his/her age, prescriber’s name, drug name and date. The Bills table contains the patient name, doctor’s specialization, drug type, date and amount to be paid. In that particular hospital there is a rule that, if a drug is administered to a patient, s/he has to be billed on the same date. This rule (σ) can be enforced to generate tuples in the *Bills* table from the *AdmDrug* table. However, the schema of the two tables are different. Whereas, in the *AdmDrug* table we have “Prescribed by” doctors and “Drug” drug names, in the *Bills* table we have “Specialization” and “Drug Type”. To resolve this problem, we take advantage of the Drug and Person dimensions (Figure 1.3). Figure 1.3(b) shows that the drug “Ibuprofen” is of “General Sale” drug type and is prescribed by (person) “Emdad” who is a “Cardiologist” (Figure 1.3(d)), we can utilize this relationship and formulate that in the rule (σ) to populate tuples in the *Bills* table. The tuples in the *AdmDrug* table are ground predicates, as these are not generated through a rule but can be at the body of a rule to generate new tuples, say in the *Bills* table. If there is a semantic integrity constraint on these tables stating

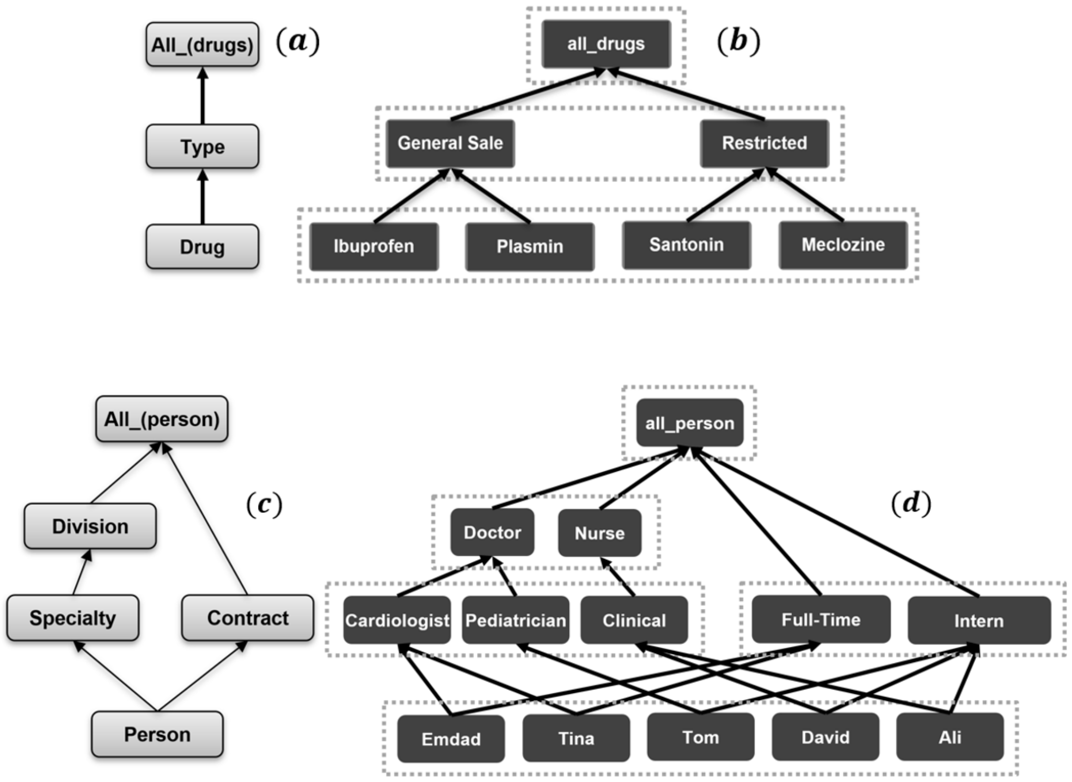


Figure 1.3: Drug (a,b) and Person (c,d) Dimension Entity: Dimensional Schema (a,c) and Dimensional Instance (b,d).

"Restricted drugs must be prescribed by the full time doctor", after matching with the dimension (Figure 1.3) and data tables (Figure 1.2) we find that 3rd and 4th records of both the tables do not satisfy this constraint, because, "Santonin" is a restricted drug which is prescribed by "Tom", who is a doctor but not full-time employee, and also prescribed by "David" who is not even a doctor but a nurse. Both of them are not full-time staff members too. The challenge is: these 4 tuples may not be the only root cause of inconsistency, it could also be the rules or constraints or even the dimension itself. As the first step to repair or restore consistency, the presence of inconsistency has to be detected and, following that, the sources of inconsistency have to be identified. Inconsistency detection involves the approach of logical query answering, which refers to running queries over the theory. The answer to those queries would contain the sources of inconsistency or nothing if the theory is consistent. The identification method uses logical negation of the constraints and then looking for the predicates among the dimensions and database tuples that satisfy the negation of the constraints. Such predicates, if found, are considered as the source of inconsistency. The concept is similar to consistent query answering [9] where the whole database might be inconsistent but the answers returned for any query running over it is consistent. Those answers avoid the inconsistent part of the database through the queries. In our research we identify and present a set of tuples or dimensional predicates which are the sources of inconsistency and eventually are subject to deletion. Generally, data repair can be done by either update or deletion. We consider deletion and present the subset of the tuples and dimensional predicates as deletion candidates. Deleting the items is not a straightforward process since we want to make changes in our database as minimum as possible. This minimality could manifest itself in many ways: subset, cardinality, priority level or weight based minimal [10]. Among these preferred repair semantics, we prefer weight based approach over others, because our source of inconsistency set under consideration has different types of predicates, i.e. tuples and dimensions, and among the tuples there could be generated or existing ones. Thus, weight gives more flexibility to the users compared to the other techniques in ranking those predicates and enumerating minimality of their weights. In such case, rather than plain subset, cardinality or priority based minimality, weight based minimality would provide better control. Until now, such weights for the predicates have been trusted to be assigned by an expert or regular user.

In terms of the Ontological Multidimensional Data (OMD) model, restoration of consistency has not been studied before. Previous methods of data repair only considered data tables with different integrity constraints or consistent query answering over inconsistent ontologies but not both of them as in our OMD model. Moreover, those studies did not involve Datalog$^\pm$ ontologies with deletion or update based repairing. Another challenge is with the allowed constraints, which are more general in nature than typical dependencies in databases for a single database table, which is precisely why such methods are not suitable for our OMD model repair.

1.3 Contributions

We have studied the problem of restoring consistency between a multi-dimensional data model (involving an ontology and data instance), and a set of dimensional constraints and rules via deletion. We propose a weight assignment methodology that allows us to rank and quantify the set of minimal weight repairs needed to achieve consistency. Overall, we make the following contributions:

1. We derive a mechanism to detect inconsistency of the ontological multi-dimensional data model and (if required) to generate ground atoms which are the cause of inconsistency. Then based on the properties of OMD, we reduce it to a Multi-Criteria Decision Making (MCDM) problem and propose a weight generation method that assigns weights to each tuple, dimensional predicate without user intervention [11].

2. We formally define minimal weighted repair for the OMD model, and recommend corrections via deletion of candidate predicates.

3. From the set of deletion candidates, to minimize impact we propose the Min-Sum algorithm based on dynamic programming. Finally, we design a genetic algorithm based optimization solution that selects minimal weight predicates. We show that our algorithm runs efficiently and can be deployed in practice.

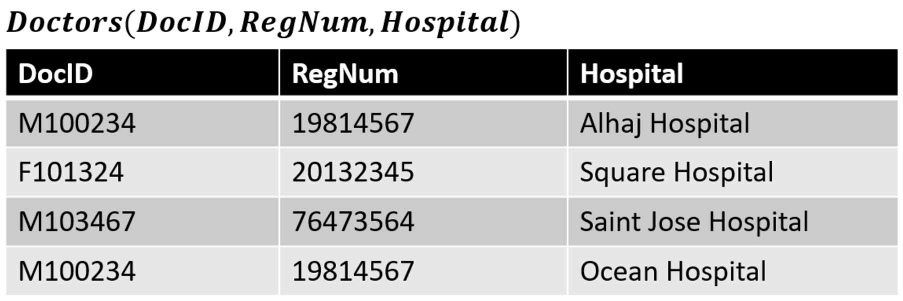
Chapter 2

Background

This chapter introduce preliminary concepts which are required to explain OMD model, inconsistencies and weight generation methods. We provide necessary background definitions of integrity constraints in databases, also known as data dependencies, and components of the OMD in details. Later we discuss Multi-Criteria Decision Making (MCDM) and the CRITIC method which we have utilized in our research for automatic weight generation. At the end, we outline the concept of Genetic Algorithm (GA) and discuss its application in finding minimal weighted subset for deletion.

2.1 Data Dependencies

Data dependencies are integrity constraints defining relationships that the data must satisfy to model correctly the part of the world under consideration. There are several kinds of dependencies but we consider three types of dependencies in this work (All three of them can be expressed in first order logic formulas [12]):



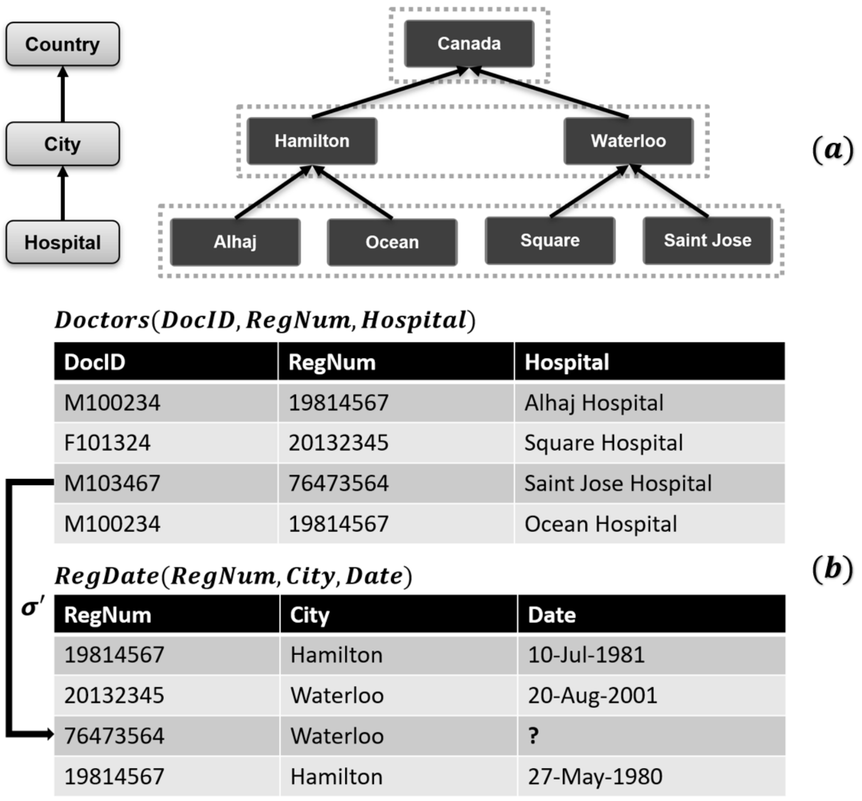
***Doctors*(*DocID*, *RegNum*, *Hospital*)**

DocID	RegNum	Hospital
M100234	19814567	Alhaj Hospital
F101324	20132345	Square Hospital
M103467	76473564	Saint Jose Hospital
M100234	19814567	Ocean Hospital

FIGURE 2.1: Doctors tuples

- **Equality-Generating Dependencies (EGDs):** Formulas of the form $\forall x : [\phi(x) \rightarrow (x_1 = x_2)]$ where, $\phi(x)$ is a conjunction of atomic formulas, all with variables among the variables in x. Every variable in x also appear in $\phi(x)$ and (x_1, x_2) are distinct variables in x.
For example, if there is a table "Doctors" (Figure 2.1) with fields like Doctor's ID,

Registration Number and Hopsital they are assigned to (i.e. predicates of Doctors(DocID, RegNum, Hospital)) and we want to enforce a constraint: "If each doctors IDs are same, the corresponding registration numbers should also be the same", that is, the same doctor cannot have two different registration numbers, then we can enforce EGD like this:
$Doctors(DocID_1, RegNum_1, Hospital_1) \bigwedge Doctors(DocID_2, RegNum_2, Hospital_2)$
$\bigwedge(DocID_1 = DocID_2) \rightarrow (RegNum_1 = RegNum_2)$.



Doctors(DocID, RegNum, Hospital)

DocID	RegNum	Hospital
M100234	19814567	Alhaj Hospital
F101324	20132345	Square Hospital
M103467	76473564	Saint Jose Hospital
M100234	19814567	Ocean Hospital

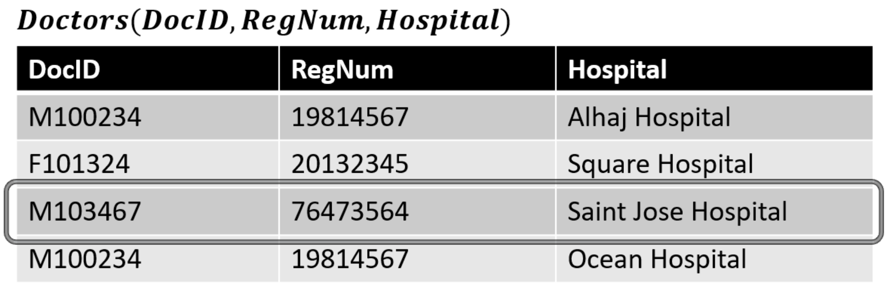
RegDate(RegNum, City, Date)

RegNum	City	Date
19814567	Hamilton	10-Jul-1981
20132345	Waterloo	20-Aug-2001
76473564	Waterloo	?
19814567	Hamilton	27-May-1980

Figure 2.2: TGD generating tuples

- **Tuple-Generating Dependencies (TGDs):** Formulas of the form $\forall x : [\varphi(x) \rightarrow \exists y : \psi(x, y)]$ where, $\varphi(x)$ is a conjunction of atomic formulas, all with variables among the variables in x. Every variable in x appears in $\varphi(x)$ but not necessarily in $\psi(x, y)$ which is also a conjunction of atomic formulas, all with variables among the variables in x and y.
  For example, there is a simple "Hospital" dimension (Figure 2.2(a)) which has the predicate $HospitalCity(Hospital, City)$ which is connecting Hospitals with the cities where those are situated in. We also have another table "RegistrationDate" (Figure 2.2(b)) with the predicate:
  $RegDate(RegNum, City, Date)$ which contains the information on registration of

the doctors done in which city and on what date. Now a rule can be enforced which says that, if there is a doctor who is working in a hospital of any city, he should also be registered in that city. To enforce that, we deduce a rule in the form of TGD, σ' : $Doctors(DocID, RegNum, Hospital) \bigwedge HospitalCity(Hospital, City) \rightarrow \exists Date : RegDate(RegNum, City, Date)$. This rule has the capability of generating a tuple in the "RegDate" table from the records in "Doctors" table, using the "Hospital" dimension.


$\boldsymbol{Doctors(DocID, RegNum, Hospital)}$

DocID	RegNum	Hospital
M100234	19814567	Alhaj Hospital
F101324	20132345	Square Hospital
M103467	76473564	Saint Jose Hospital
M100234	19814567	Ocean Hospital

Figure 2.3: Denial Constraint found an inconsistency

- **Denial Constraints (DCs):** Similarly $\forall x, y : [\psi(x, y) \rightarrow \bot]$ is the form of DCs that does not allow ($\bot$ is for *False*) those records which satisfy $\psi(x, y)$, which is also a conjunction of atomic formulas, all with variables among the variables in x and y. In Datalog$^{\pm}$ it is known as negative constraints (NCs) that can be seen as a special case of denial constraints over databases.
  For example, for doctor's registration number in the city of Waterloo should be between $10,000,000$ to $70,000,000$ (inclusive) and this can be expressed using NC as:
  $Doctors(DocID, RegNum, Hospital) \bigwedge HospitalCity(Hospital, City)$
  $\bigwedge(City = Waterloo) \bigwedge(RegNum < 10000000) \bigwedge(RegNum > 70000000) \rightarrow \bot$
  Here, (Figure: 2.3) in the table "Doctors" there is a hospital named "Saint Jose Hospital" and using the"Hospital" dimension (Figure 2.2(a)) it can be shown that it is in Waterloo. Hence, using the dimension and the above stated NC it is obvious that the doctor bearing the ID:M103467 has the registration number which is out of the range for registration values, allowed for the city of Waterloo. Thus, whichever predicate satisfies this body of the constraint are the source of inconsistency.

Codd in his seminal paper [13] proposed functional dependencies as constraints over databases, which is followed by multivalued dependencies later [14]–[16]. After that several researchers [17], [18] independently proposed equality and tuple generating dependencies (defined above) which actually generalizes previously stated types. Practically, these enforces that, if some tuples in the database satisfies certain equalities, some values in the tuple must be equal (EGD) or some other tuples must exist (TGD) or should not exist (DC) in the database.

2.2 OMD Model

The Ontological Multidimensional Data (OMD) model is a Datalog$^{\pm}$ based ontological extension of the Hurtado-Mendelzon (HM) model for multidimensional data [19]. Datalog$^{\pm}$ is used as it can express the EGDs, TGDs and NCs which are used to explain OMD model. Each of the OMD model's four components is described below [5].

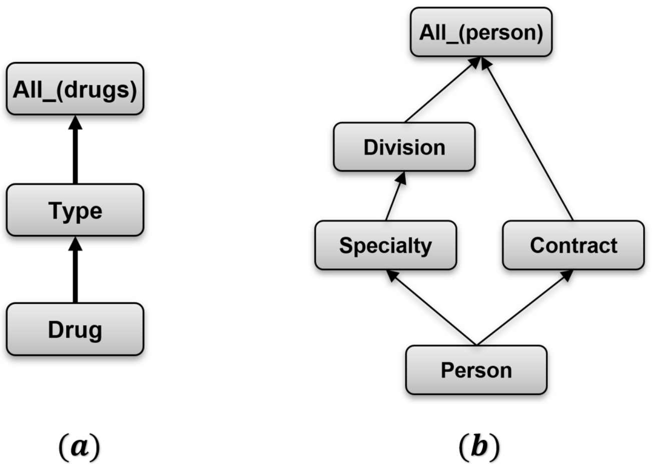
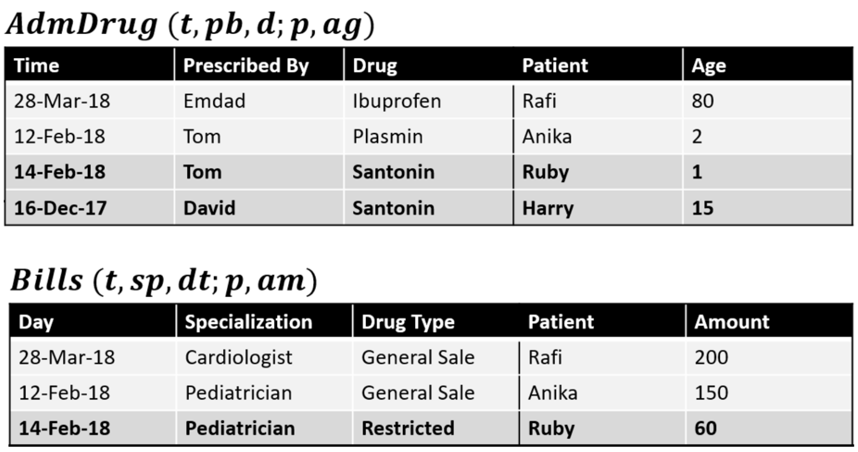


FIGURE 2.4: Drug (a) and Person (b) Dimensional Schema

1. **Dimensional Schema:** Let $R^M = H \cup R^C$ be an OMD schema, where H is a relational schema with multiple dimensions. Let $H = K \cup L$, where K is a set of unary category predicates ($k \in K$) and L is a set of binary child-parent predicates ($l \in L$) of the dimension. Let R^C is the set of categorical predicates ($r^c \in R^C$) that include data from both dimension and relational database tables. In the Figure 2.4 we have 2 dimensions, *Drug* and *Person*. Here we have the dimensional schema with unary category predicates (e.g. in Drug dimension (a) $Type(Q)$ and in Person dimension (c) $Division(R)$). Similarly we also have the child-parent predicates (e.g. in Drug dimension (a) $DataType(P, Q)$, and in Person dimension (c) $PersonSpeciality(X, Y)$). Now categorical predicates (Figure: 2.5) are the schema of the data tables i.e. $AdmDrug(t, pb, d; p, ag)$ which is basically *AdministerDrug* data table with the columns *(Time, Prescribed By, Drug, Patient, Age)*.



$\boldsymbol{AdmDrug\ (t, pb, d; p, ag)}$

Time	Prescribed By	Drug	Patient	Age
28-Mar-18	Emdad	Ibuprofen	Rafi	80
12-Feb-18	Tom	Plasmin	Anika	2
14-Feb-18	**Tom**	**Santonin**	**Ruby**	**1**
16-Dec-17	**David**	**Santonin**	**Harry**	**15**

$\boldsymbol{Bills\ (t, sp, dt; p, am)}$

Day	Specialization	Drug Type	Patient	Amount
28-Mar-18	Cardiologist	General Sale	Rafi	200
12-Feb-18	Pediatrician	General Sale	Anika	150
14-Feb-18	**Pediatrician**	**Restricted**	**Ruby**	**60**

Figure 2.5: AdmDrug and Bills categorical predicates schema and instance

2. **Dimensional Instance:** Let $I^M = D^H \cup I^C$ be the OMD model database instance, where D^H is a complete instance for dimensional sub-schema H containing the category and the child-parent i.e. binary predicates. Let us consider the unary categories instance for D^H are $(d_u \in D^H)$ and binary instances of child-parent predicates are $(d_b \in D^h)$. Sub-instance I^C contains possibly partal, incomplete extensions for the categorical predicates i.e. those in R^C.

   A database instance consists of an Extensional Database (EDB), which is defined by facts, i.e. existing records before applying any rules, i.e., $EDB(I^C) = \{d_e..., d'_e\}$. According to Logic Programming concepts, a term is either a constant or a variable. A term t is ground if and only if no variable occurs in t. In Datalog, this is known as a constants only term, which is similar to the fact or tuple in a database. EDB is a finite set of positive ground facts [20]. If an EDB is not clean due to incompleteness, then we may use some rules, such as TGDs, to generate the missing data. For example, we can have a Drug dimensional entity and a Person dimensional entity (Figure: 1.3) schema (Figure: 2.4) and instance (Figure: 2.6). In Figure 2.6, *Person(Emdad)* is an unary category instance, and *PersonSpeciality(Emdad, Cardiologist)* is a binary child-parent instance. For categorical predicates sub-instance (Figure: 2.5): *AdmDrug(28-Mar-18, Emdad, Ibuprofen, Rafi, 80)*.

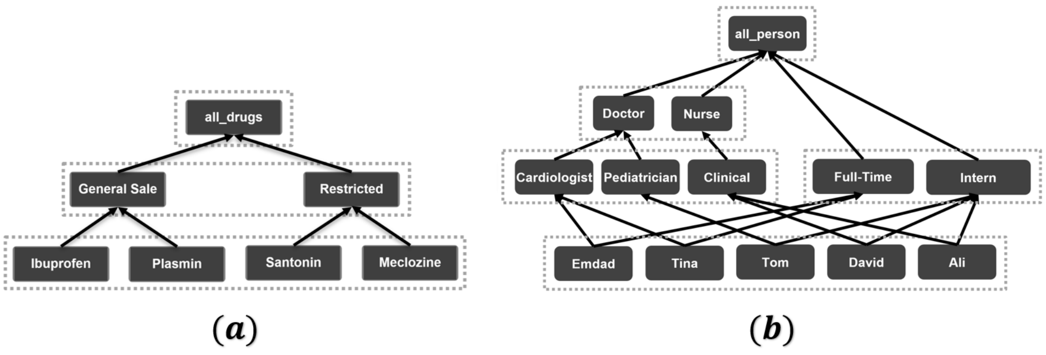


FIGURE 2.6: Drug(a) and Person(b) Dimensional Instance

3. **Dimensional Rules:** TGDs act as the dimensional rules to facilitate database completion. That is, if some records are missing, we can apply TGDs to generate those necessary tuples in the database. The TGD (σ) is of the format: $\sigma = R_1(\bar{x_1}; \bar{y_1}), ..., R_n(\bar{x_n}; \bar{y_n}), P_1(x_1, x_1'), ..., P_m(x_m, x_m') \rightarrow \exists y' R_r(\bar{x_k}; \bar{y})$

   If we compare this with the logical implication format, there is a body and a head. Let's consider:

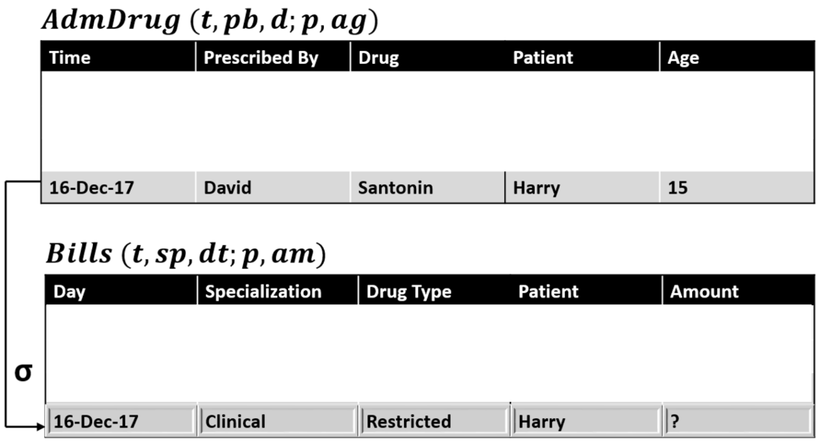
   $[Body(\sigma) = R_1(\bar{x_1}; \bar{y_1}), ..., R_n(\bar{x_n}; \bar{y_n}), P_1(x_1, x_1'), ..., P_m(x_m, x_m')]$ and $[Head(\sigma) = R_r(\bar{x_k}; \bar{y})]$ then

   for simplicity: $\sigma = r_1^c, r_2^c, ...r_n^c, l_1, l_2, ...l_m \rightarrow r_t^c$.

   Here, all r^c are the categorical predicates and all l are the child-parent predicates. If we take all the atoms which satisfy this rule it will be like: $\sigma = d_e, d_b, d_i' \rightarrow d_i$. It is indeed a TGD, and the generated tuples create the Intensional Database (IDB) part of I^C, i.e. we can state $IDB(I^C) = \{d_i, ..., d_i'\}$. Formally, the set of derived relations (known as views in databases) through the rules is called intensional database [20]. For example, we have two data tables *AdmDrug* and *Bills*. Consider a rule that generates tuples from the former table to the latter table, as shown in Figure 1.2. Rule: (used '?' for missing fields)

   $\sigma : AdmDrug(t, pb, d; p, ag), PersonSpec(pb, sp), DrugType(d, dt)$
   $\rightarrow \exists am : Bills(t, sp, dt; p, am)$
   generates the 4th tuple *Bills(16-Dec-17, Clinical, Restricted, Harry, N/A)* in the *Bills* table to assure completeness. From the Rule:

   $\sigma : AdmDrug(16\ Dec\ 17, David, Santonin; Harry, 15),$
   $PersonSpec(David, Clinical), DrugType(Santonin, Restricted)$
   $\rightarrow \exists am : Bills(16\ Dec\ 17, Clinical, Restricted; Harry, am?)$



$AdmDrug\ (t, pb, d; p, ag)$

Time	Prescribed By	Drug	Patient	Age
16-Dec-17	David	Santonin	Harry	15

$Bills\ (t, sp, dt; p, am)$

Day	Specialization	Drug Type	Patient	Amount
16-Dec-17	Clinical	Restricted	Harry	?

Figure 2.7: TGD rule generating tuple from table "AdmDrug" to table "Bills"

As stated above, $d_i' \in IDB(I^C)$, so there could also be another rule, σ' which generated this tuple. If we take all the ground atoms which satisfy $Body(\sigma')$, we will recursively obtain a set of predicates from the EDB and parent-child predicates i.e. $\{d_e, ..., d_b\}$, as shown in Algorithm 1. One important condition for repair generation is that, every rule should terminate, that is, it cannot keep on generating infinite tuples.

We define a look-up table to store the rules, and the corresponding generated tuples. Let's call this table $RuleRecord(d_i)$, that takes as input the tuple which has been generated by any dimensional rule, and returns the set of rules which have been used to generate this tuple.

4. **Dimensional Constraints:** We consider two types of constraints here (as special forms of EGDs and Denial Constraints for OMD models):

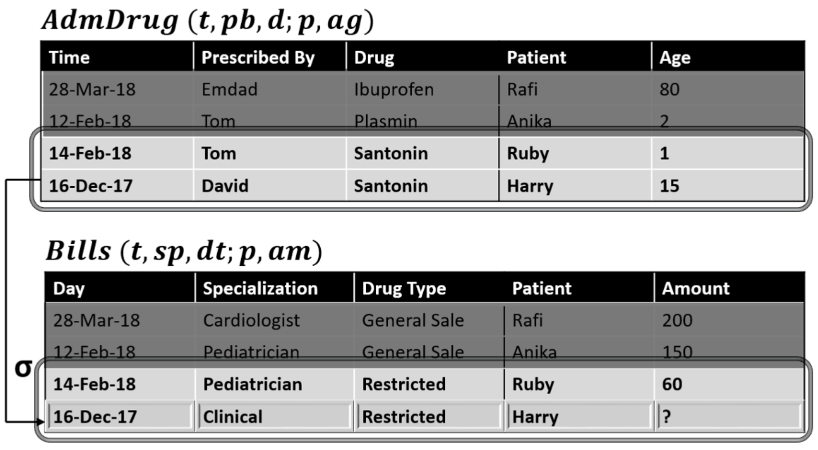
   $\eta_{egd} = R_1(\bar{x_1}; \bar{y_1}), ..., R_n(\bar{x_n}; \bar{y_n}), P_1(x_1, x_1'), ..., P_m(x_m, x_m') \rightarrow z = z'$

   $\eta_{dc} = R_1(\bar{x_1}; \bar{y_1}), ..., R_n(\bar{x_n}; \bar{y_n}), P_1(x_1, x_1'), ..., P_m(x_m, x_m') \rightarrow \perp$

   We assume these are user-defined constraints that should be applied and are consistent with the database and the dimensions. Henceforth, we assume that the possible sources of inconsistency are either the data tuples, or the dimensional predicates, or both.

   For example, consider a user-defined constraint defined as "Restricted Drugs must be prescribed by a full-time doctor" which can be written as:
   $\eta_1 = AdmDrug(t, pb, d; p, ag), DrugType(d, Restricted), PersonSpec(pb, sp),$
   $SpecDiv(sp, div),\ PersonContract(pb, con) \Rightarrow (div = Doctor), (con = FullTime)$
   and

$$\eta_2 = Bills(t, sp, dt; p, am), (dt = Restricted), PersonSpec(pb, sp),$$
$$SpecDiv(sp, div), PersonContract(pb, con) \Rightarrow (div = Doctor), (con = FullTime)$$


$\boldsymbol{AdmDrug\ (t, pb, d; p, ag)}$

Time	Prescribed By	Drug	Patient	Age
28-Mar-18	Emdad	Ibuprofen	Rafi	80
12-Feb-18	Tom	Plasmin	Anika	2
14-Feb-18	**Tom**	**Santonin**	**Ruby**	**1**
16-Dec-17	**David**	**Santonin**	**Harry**	**15**

$\boldsymbol{Bills\ (t, sp, dt; p, am)}$

Day	Specialization	Drug Type	Patient	Amount
28-Mar-18	Cardiologist	General Sale	Rafi	200
12-Feb-18	Pediatrician	General Sale	Anika	150
14-Feb-18	**Pediatrician**	**Restricted**	**Ruby**	**60**
16-Dec-17	**Clinical**	**Restricted**	**Harry**	**?**

Figure 2.8: Inconsistent Tuples

These constraints give us 4 inconsistent predicates from the data tables in Figure 2.8 (3rd and 4th tuples from both tables) and five child-parent inconsistent predicates from the Drug and Person dimension (Figure 1.3). Considering (d = dimension, t = tuple, 'N?' = Not determined value) in the Table 2.1:

d1 = DrugType(Santonin, Restricted)	*t1 = AdmDrug(14Feb, Tom, Santonin, Ruby, 1)*
d2 = PersonContract(Tom, Intern)	*t2 = AdmDrug(16Dec, David, Santonin, Harry, 15)*
d3 = PersonContract(David, Intern)	*t3 = Bills(14Feb, Pediastrician, Restricted, Ruby, 60)*
d4 = PersonSpec(David, Clinical)	*t4 = Bills(16Dec, Clinical, Restricted, Harry, N?)*
d5 = SpecDiv(Clinical, Nurse)	

Table 2.1: Inconsistent Predicates

To achieve consistency, we can delete all 9 tuples. Deletion from the data tables are easier than the deletion from the dimension. In case of the dimensions of the OMD model, there is a desirable property called summarizability, that allows the computation of aggregate queries at higher level categories using pre-computed answers at lower level categories, with significant positive impact on data warehouse efficiency [21]. TGDs and EGDs can also be used to enforce summarizability and there are significant amount of work have been done on how to repair such dimensions mostly by deletion or structural change for binary predicates. However, we aim to minimally delete which means the fewest number of changes. For example, if the child-parent predicate $DrugType(Santonin, Restricted)$ is the candidate for deletion, then we do not even have to delete any other tuple or child-parent atoms. However, this could be an important connection in the dimension, which should be differentiated. To allow this differentiation, we apply the multi-criteria decision making method (CRITIC) which will help us assign weights to the predicates (without human intervention), but on the basis of the available data and its statistics [22].

2.3 Multi-Criteria Decision Making

Decision making is the act of choosing between two or more courses of action. When the criteria for making decisions increases, the complexity to prioritize criteria becomes difficult. Multi-Criteria Decision Making (MCDM) is a sub-discipline of operations research, that explicitly evaluates such criteria. This has been an active area of research since the 1970s. MCDM draws upon knowledge in mathematics, decision analysis, economics, business, information technology and applied successfully in business, supply chain management, urban development and financial sectors. MCDM problems are traditionally solved through the concept of establishing the relative importance of attributes that influence the choice of decision alternatives [23]. The decision process involves [24]:

(a) Define the problem and objective

(b) Describe alternatives

(c) Define decision criteria

(d) Study and evaluate the criteria

(e) Prepare a decision matrix by arranging the alternatives against the criteria

(f) Determine the subjective or objective weights for the criteria

(g) Synthesis on results

(h) Decision making

For assigning differential weights to the decision criteria (Step f), we adopt computational methods to compute weights to maximize objectivity. There is a well developed method of ordering criteria known as pairwise comparisons, which involve the pairwise comparison of each criteria against every other criteria using scaling methods [25]. This is usually quite subjective, requiring an expert decision maker that is prone to variance and inconsistency. There exist past work to unify scaling methods, locate and reduce inconsistency [26], these are still often user-dependent. We adopt an objective weighting method, where the criteria is evaluated pairwise in a matrix and a score is given to compute the correlation, conflict and contrast between the evaluated criteria pair. We can then score each criteria by aggregating along the rows and columns of the matrix to derive a unified weight representing the importance of a criteria. There are several objective weighting methods that can be used:

- **Entropy:** Entropy is a measures of the amount of uncertainty represented by the probability distribution and is a measure of the amount of chaos or of the lack of information about a system [27]. For the availability of complete information, entropy $= 0$. If not complete then it is greater than zero [28].

- **Mean Weight:** The weights are defined as the mean $w_j = 1/n$, where n is the number of criteria. This method assumes that all criteria are of equal importance.
- **Standard Deviation:** Weights are assigned using the average standard deviation $w_j = \sigma_j / \sum_{j=1}^{n} \sigma_j$ [29].
- **Statistical Variance:** Weights are derived by calculating the statistical variance of information[30].

The entropy, standard deviation and statistical variance methods only capture contrast in the data, but not conflict. The mean weight assumes all criteria are equally important, leading to potential information loss. We consider the CRITIC weighting method that addresses the above shortcomings, and describe its details next.

2.4 CRITIC Method

The CRiteria Importance Through Inter-critera Correlation (CRITIC) is one of the most popular methods of measuring the divergence in performance ratings and can suitably be used to determine criteria weights for our problem. In their proposed CRITIC method the authors emphasized that each decision situation has particular characteristics independent of the decision maker's way of thinking [22]. Attributes can be viewed as information sources and that weight of importance reflect the amount of information contained in each of them. In addition to the information emitted by criteria they added another dimension which is conflict between different attributes. This is developed based on the analytical investigation of the evaluation matrix for extracting all information contained in the evaluation criteria. The steps are [22] :

1. Organize the data as performance matrix U with columns $C(C_1, C_2, ..., C_m)$ are the criteria and rows are the alternatives $P(P_1, P_2, ..., P_n)$ with entries p_{ij} being indicators of alternatives across criteria.
2. Transform this matrix U into a normalized score matrix X (having each value within the internal $[0, 1]$) with relative scores of alternatives. This transformation is based on the concept of ideal point [31]. For $(i = 1..n, j = 1..m)$:

$$(x_{ij} \in X) : x_{ij} = \frac{p_{ij} - p_j^{min}}{p_j^{max} - p_j^{min}} \tag{2.1}$$

3. Determine the standard deviation (σ) or entropy (H) scores for each criteria. These σ or H actually quantifies the contrast intensity of the corresponding criteria:

$$\sigma(C_j) = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n}} \tag{2.2}$$

$$H(C_j) = -\sum_{i=1}^{n}(p_i \times \log_2(p_i)) \tag{2.3}$$

4. Construct a symmetric matrix (dimension $m \times m$) with the generic element r_{jk} which is the Pearson's Correlation Coefficient (also can be Spearman Rank Correlation) between the vectors x_j and x_k. It can be seen that the more discordant the scores of the alternatives in criteria j and k, the lower the value of r_{jk} (Pearson) and ρ_{jk} (Spearman):

$$r_{jk} = \frac{\sum_{j=1,k=m}^{m,1}(x_j - \overline{x})(x_k - \overline{x})}{\sqrt{\sum_{j=1,k=m}^{m,1}(x_j - \overline{x})^2(x_k - \overline{x})^2}} \tag{2.4}$$

$$\rho_{jk} = 1 - \frac{6\sum d_j^2}{m(m^2-1)} \tag{2.5}$$

Where, $d =$ the pairwise distances of the ranks of the variables x_j and x_k.

5. Measure the conflict created by the criterion j with respect to the decision situation defined by the rest of criteria.

$$Conflict(C_j) = \sum_{k=1}^{m}(1 - r_{jk}) \tag{2.6}$$

6. Calculate the amount of information I_j emitted by the jth criterion by composing the measures that quantifies the conflict and contrast intensity through the following multiplicative aggregation formula:

$$I_j = \sigma_j \times \sum_{k=1}^{m}(1 - r_{jk}) \tag{2.7}$$

$$I_j = H_j \times \sum_{k=1}^{m}(1 - r_{jk}) \tag{2.8}$$

The higher the value of I_j the larger amount of information transmitted by the corresponding criterion and the higher it's relative importance for the decision making process.

7. Normalize these information values to unity to determine the objective weights of that particular criteria:

$$w_j = \frac{I_j}{\sum_{k=1}^{m} I_k} \tag{2.9}$$

8. Finally determine the relative weights for the alternatives according to the following aggregation formula:
$$w_i = \sum_{j=1}^{m} (w_j \times x_{ij}) \tag{2.10}$$

In our work, these are the weights (after integer scaling) of the deletion candidate predicates determined by this CRITIC method. With some initial experiments we have found that, if the data table has more variations and contains no outlier, then from step-3 above, the standard deviation works well. But if data table contains outliers and most of the data values are same the entropy based contrast provide more realistic weights. Here we cannot ignore the outliers because these values are counts of certain criteria features which could be very high or low and can impact significantly the determination of weights.

2.5 Genetic Algorithms

Genetic Algorithms (GAs) are adaptive (changes behavior at run-time) methods which are used to find the maximum or minimum of a particular function i.e. solution to optimization problems. In optimization the usual goal is to find the global optimal solution which is considered as the best solution in the whole solution space. But solution space can have obstructions associated with constraints, noise, unsteadiness, and a large number of local optima. In such situations, well designed GAs can find practically viable optimal solutions. The concept was first introduced by Holland [32] and later it was discussed under the field of study called Evolutionary Computation where these algorithms imitate the biological process of reproduction and natural selection to solve for the fittest solutions [33]. Just like nature, most of the genetic algorithms processes are stochastic type but efficient than random or exhaustive search algorithms.

The terminologies of this algorithm are borrowed from biology [34], yet they are much simpler, which are:

- A **Fitness** function for optimization.
- A **Population** of **Chromosomes**
- **Selection** of chromosomes which will reproduce
- **Crossover** to produce next generation of chromosomes
- Random **Mutation** of chromosomes in new generation

GA begins with the population which is a set of solutions to a particular problem or objective function (Figure 2.9). Each solution is usually encoded as a genotype or chromosome. If the values represented as the chromosome are continuous, those are called vectors, but if the values are just bits, those are called bit string. Ours is a discrete combinatorial problem, so we use bit string representation for the chromosomes. Each of the solutions or chromosomes is assigned a quality parameter or fitness score which measures how good the solution is to the problem. Fitness functions can also be used to differentiate infeasible solutions from the solution space, which we also did in designing our function. The highly fit chromosomes are randomly selected for reproduction or cross-breeding which produces a child chromosome that share some features taken from each parent. In GA more than two parents are allowed but we used very basic two parent model for the crossover operation. In general, to introduce new variation in the features slight disturbance or mutation is added to the child chromosomes. This mutation basically helps against local optima and crossover explores the more promising areas of the search space. Flexible termination criteria is another benefit of using GAs. GAs also allow multiple sub-optimal solutions to be provided upon termination. The termination criteria is normally set by the user, which can be defined as number of iterations achieved, or results satisfying a given threshold. The crux here is the design of these functions. If done well the population will converge to an optimal solution to the problem.

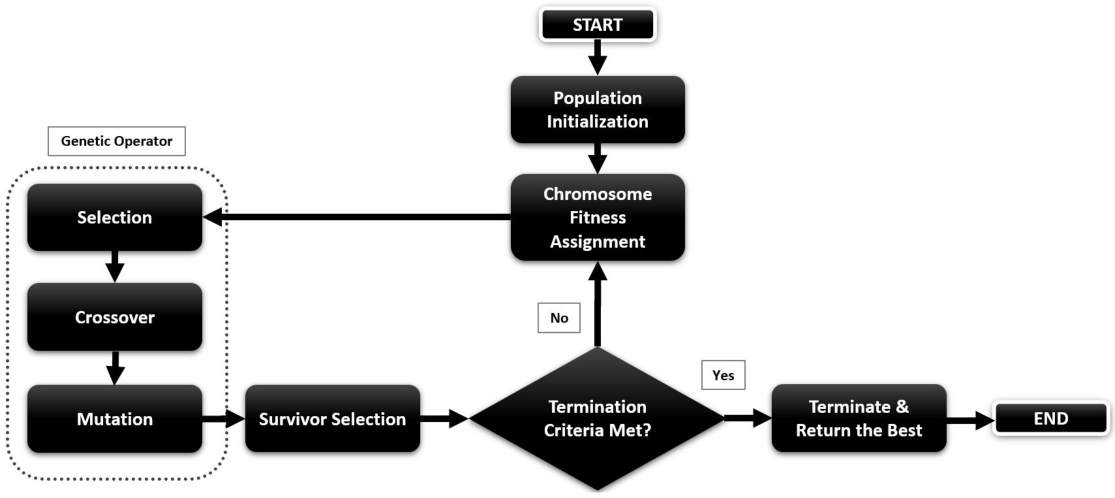

START
Population Initialization
Chromosome Fitness Assignment
Genetic Operator
Selection
Crossover
Mutation
Survivor Selection
No
Termination Criteria Met?
Yes
Terminate & Return the Best
END

FIGURE 2.9: Genetic Algorithm Steps

Genetic algorithms randomly explore the whole search space and evaluate samples in many regions simultaneously, which can even be amplified by parallel computation. This strength of genetic algorithms to focus their attention on the most promising parts of a solution space is a direct outcome of their ability to combine strings containing partial solutions [35]. In those cases, where traditional algorithms do not perform well with respect to time and space, GAs provide near-optimal practical solutions. Compared to other similar techniques like Gradient Methods, Iterated Search and Simulated Annealing, GAs offer robust and better solutions [36]. It does not require any derivative information and performs faster and less space hungry than traditional optimization algorithms like Greedy or Dynamic Programming. It also has very good parallel capabilities. GAs work with both the continuous and discrete optimization problems, even with multi-objective functions. It provides not just single solution but set of good solutions. At any point during iteration, it has at least a solution, which is improved over time. One weakness is calculating fitness function repeatedly might be computationally expensive for some problems. In our fitness function, we incorporate satisfaction checking with Datalog queries which is also very costly operation. Proving convergence with iterations is often not obvious and the speed at which convergence takes place is also very difficult to tackle. Also being stochastic, there are no guarantees on the optimum or quality of the solution. These limitations are also applicable for our version of the implementations described in Chapter 4.

The main challenge of using GAs is in the implementation phase. From Phenotype Space (actual solution space) to Genotype Space (computation space) encoding and when the solution is produced the decoding in the reverse direction is one of the complex tasks. Representation, population initialization, right fitness function defining, effective crossover strategy and finally intelligent mutation are the key to successful implementation of genetic algorithms. In Chapter 4 we discuss how we have addressed these challenges to solve our minimal weight problem in details.

Chapter 3

Related Work

This work find similarities to three main lines of works: consistent query answering, priorities for consistency, and database repair techniques. After brief overview in the area of data cleaning we explain how our research is connected to these fields.

3.1 Overview

Data cleaning is a very broad area discussed in two major segments: error detection and repair techniques. The overall trends in data cleaning under these segments have been discussed briefly in [37]. This work includes a taxonomy of anomaly detection techniques, error types, propagation, repair techniques, repair target, update model and automation of detection and repair process. The authors have classified anomaly detection techniques into 3 groups: (1) Error Type (What to detect?) (2) Automation (how to detect?) (3) Business Intelligence Layer (Where to detect?). Our work falls in the automatic detection of inconsistency as we are designing the condition and the automatic weight generation method to detect and assign weights for repair. The authors have also classified data repairing techniques: (1) Repair Target (What to repair?) (2) Automation (How to repair?) (3) Repair Model (Where to repair?) [37]. In our research, we actually touch all three of these repairing techniques: Our target can be both the dimensions and the database, we automate the process with minimum user intervention, our repair model follows the repair candidates via deletion of the tuples in the database or items in the dimensions.

3.2 Consistent Query Answering

Bry first proposed the notion of consistent query answer in inconsistent databases using proof-theoretic approach that provides semantics for formal languages [38]. Here, the author defined the terms associated with consistency but did not provide any mechanism for computing consistent query answers. The query modification based repair mechanism and formal definition of consistent query answer was given as: "A tuple t is a consistent answer to a query Q in a database instance D with respect to a set of integrity constraints IC, if t is an answer to the query Q in every repair D' of D with respect to the integrity constraints" [39]. According to this definition, consistent query answer contains those tuples that are valid in every possible repairs of the original database in the model-theoretic sense. They also considered repairs as subset minimal that means, each repair D' such that no proper subset ($D'' \subset D'$) of D' is a repair. But this method had some limitations, it modifies the queries, inclusion dependencies are not covered and works only for quantifier free conjunctive queries. For our case, repairs are query independent, we allow EGDs, TGDs and DCs, and also define minimality based on prioritized weights which are derived from the system automatically.

3.3 Priorities for Consistency

A non-monotonic logic is a formal logic that can capture and represent defeasible inferences where reasoners draw tentative conclusions, enabling reasoners revise the conclusions subject to further evidence [40]. Reiter introduced default reasoning in non-monotonic logic by extensions and inference rules but he did not represent priorities between defaults [41]. Poole resolved the priority issue based on hypothetical reasoning [42]. His basic idea was dividing the logical theories into two levels: core level was premises which must be consistent and the second level was less reliable hypothesis. Brewka then generalized Poole's approach with more than just two levels of hypotheses [43]. The approach proposes different levels of hypothesis, representing different degrees of reliability, starting with the innermost part as the most reliable one. If inconsistencies arise the more reliable information is preferred. Extended answer set semantics was introduced to deal with inconsistent programs containing classical negation [44]. But here, they proposed partial ordering on rules and showed applications in databases. The problem of conflicting updates in databases initially discussed in [45]. in this work, authors proposed conversion of the relational database into logical database first and then prioritized the set of sentences which were parts of the update. Priorities were considered transitive in nature that means a set of sentences had higher rank when compared to the another set of sentences. Inclusion of probabilities to each mutually conflicting facts as user preferences expressed in [46], but only one key dependency per relation is considered. In [47] a logic programming based framework has been proposed by means of evaluation function. It is similar to our weight based repair but they enumerate the repairs and generate all the stable models, whereas, we enumerate the predicates' weights and calculate minimal weight for a model. We optimize it initially by generating models in non-decreasing order of the summation of their weights. By combining several inconsistency handling approaches in [48] authors proposed a framework: Inconsistency Management Policies (IMPs) where anomalies are grouped as culprits, use deletion to restore consistency and user may incorporate their priorities. However, this does not guarantee total consistency but our techniques do by deleting the candidates and checking consistency.

3.4 Database Repair

The purpose of database repairs is to characterize the consistent data in an inconsistent database. Hence, data cleaning focuses on getting rid of semantic violations with minimal changes to the original database. The problem of repair checking independently from consistent query answering, under different repair semantics was studied in [49]. Database repairs and data cleaning were first discussed together with the introduction of attribute-based repairs to restore consistency with respect to functional dependencies [50]. According to their repair algorithm, they put every database cell in its own equivalence class i.e. set of of cells that should have the same value. Then greedily merge these classes until all the constraints are satisfied. During the merging process all the tuples which violate the constraints are tracked and checked that the cost of merging the equivalence classes is the lowest. This cost is measured in terms of the cost of changing values in tuples and the distances between the original and new values. Like functional dependencies, conditional functional dependencies (CFD) capture the consistency of data by incorporating bindings of semantically related values and that was used for data cleaning too [51]. In this paper, the authors guided detection of errors involving inconsistencies related to CFDs and inclusion dependencies. They also presented SQL based CFD violation detection technique which could check multiple constraints in a single query. We have used a similar approach using Datalog but involving TGDs for handling incomplete information. In another work denial constraint and contextual violations applied in holistic data cleaning [52]. They used cell level correction via conflict hypergraph (whose vertices correspond to the binary variables, and edges correspond to covers in the constraint matrix of the independent set polytope [53]) and probability to clean data. In our case, we have data but also we have dimension and data with TGDs. Although we may represent them as hyper-graphs, we are not considering cell level values but considering deletion of tuples and dimensional predicates.

Logic based data cleaning has a tight relationship with knowledge representation and reasoning. Relational databases can be considered as the special kind of knowledge bases. There are different classes of logic defined for reasoning with inconsistent knowledge [54]. In another work, non-classical annotated predicate logic based disjunctive logic programs were utilized to define minimal repairs for a inconsistent relational database [55]. Logic programs, because of their descriptive nature are well suited to specify the class of repairs and to reason over them. To restore consistency, premises have to be removed and selection of them is based on their relative reliability. Initial study was made on description logic for inconsistency tolerant semantics [56] but later it has been studied for Datalog$^{\pm}$ with complexity analysis and explanations[57], [58]. Some works have been done in the area of prioritizing repair subset [10] for description logic [59] including a proof that it increases complexity. In inconsistent description logic knowledge bases weight-based consistent query answering was proposed [60] but weight was kept as user defined parameter and preferred by the experts. Conflict hyper-graph based prioritized repair and consistent query answering is discussed in [61], but the weight assignment problem and the repair by deletion or update is not studied.

The combination of quantitative and logical data cleaning framework has been proposed in [62]. The authors used metric functional dependencies, a type of dependency that generalizes FDs to identify inconsistencies in domains where only large differences in metric data are considered to be a data quality problem. Their repair mechanism modifies the inconsistent data so as to minimize statistical distortion. In our case, we did not use statistics other than generating the weights which is different from repair generation process. FDs as ICs are used which work well only for a single table. In our case, we do not have single schema data table but dimensions including TGDs as rules to generate tuples. We do not consider all the tuples to be of equal weights, as they could be part of EDB or IDB. Chase is a fixed-point algorithm to determine if a certain database dependency logically follows from a given set of database dependencies [63]. Chase based repair semantics was discussed in [64], using EGDs as constraints but not considering DCs and TGDs which we consider in our work. Besides, we have designed and developed a genetic algorithm based solution to find near-optimal repairs; to the best of our knowledge it was not done before in this area.

For multidimensional databases, data cleaning involves dimension repair and which is typically the recovering summarizability i.e. strictness and homogeneity properties that broadly fall under instance based repair [65]–[68] and schema based repairs[69], [70]. Summarizability is an important property, lack of which will hamper correct query answering when using pre-computed views or lose efficiency as it might require to compute results from the beginning[71]. By using EGDs, strictness can be enforced in dimensions. Strictness ensures each member at one level, has at most one parent in the same category with the child-parent predicate. The instances of the dimensions, again have to be homogenous that ensures each element in a category has at least one parent element in each parent category. In our research, when we find any source of inconsistency in the dimension, we propose deletion. As we delete an edge, it may become inconsistent and lose these properties. This can be fixed by utilizing the above techniques that handle instance or schema based repairs.

Chapter 4

Consistency Restoration via Weighted Repairs

We discuss inconsistency detection and explain the outline of the repair procedure with a process diagram. Then we present methodology to determine the source of inconsistency, optional grounding mechanism and automatic weight generation algorithms along with relevant examples demonstrating each step. Feature extraction which is an integral part of weight generation procedure, is described next followed by the weighted repair searching. Instead of slowest brute force searching, we present a faster greedy-dynamic programming based optimum search algorithm and fastest genetic algorithm based sub-optimal solution to find the repairs to restore consistency in OMD models.

4.1 Solution Overview

The OMD model is comprised of data tables, dimensions, rules (TGDs) and constraints. When an OMD model becomes inconsistent with respect to certain constraints, it cannot have any models, and entails everything, that is, using standard classic semantics, no meaningful conclusions can be drawn from it.

- As the first step towards restoring consistency, we need to detect the presence of inconsistency and then identify the sources responsible for that inconsistency. Inconsistency detection involves a logical query answering approach. The identification method uses logical negation of the constraints. Then it searches for the predicates among the dimensions and database tuples that satisfy the negation of the constraints, and those predicates and tuples that satisfy, are considered sources of inconsistency. The concept is similar to consistent query answering where the database remains inconsistent but the answers returned for any query are consistent. In our case, we identify and present a set of tuples or dimensional predicates that are the sources of inconsistency, hence, subject to deletion.
- Due to the presence of IDB which contains generated tuples, we may have to deduce the ground EDB atoms of those tuples. Because of the rules, such tuples were first generated and to prevent the rules from generating these tuples again,

we need to delete the source EDB atoms which participated in generating those IDB tuples.

- After finding all the sources of inconsistency we need to prioritize deletion candidates among those items. Based on the OMD model features, we define the criteria that will be used to define the weights for selecting repair candidates for deletion. We evaluate the criteria via a matrix of predicates, where we apply the CRITIC method to obtain the weighted predicates.
- Finally, we search for an aggregated minimal weight solution consisting of those predicates to delete to achieve consistency in the OMD model.

Figure-4.1 is the process diagram showing the above steps with their functions and states. We represented functions as $f(x)$ and states as strings below the icons.

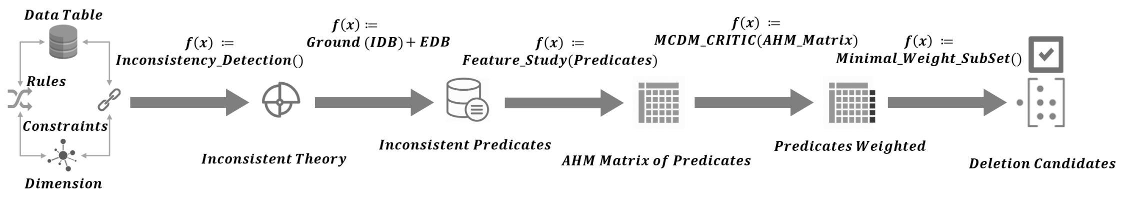

Data Table
Rules
Constraints
Dimension
f(x) := Inconsistency_Detection()
Inconsistent Theory
f(x) := Ground (IDB) + EDB
Inconsistent Predicates
f(x) := Feature_Study(Predicates)
AHM Matrix of Predicates
f(x) := MCDM_CRITIC(AHM_Matrix)
Predicates Weighted
f(x) := Minimal_Weight_SubSet()
Deletion Candidates

Figure 4.1: OMD Model Consistency Restoration Process.

4.2 Inconsistency Detection

Inconsistency detection involves optional grounding predicates and finding the source of inconsistency. We describe both procedures next.

4.2.1 Grounding IDB

The source of an inconsistency may be a tuple in the EDB or a dimensional predicate. An error may also occur in the IDB. This may be generated by one or more dimensional rules, then we need to ground them to get the atoms from the EDB which participated in generating them. This is similar to the mathematical logic where a ground expression is a term that does not contain any free variable. But in our case, ground expressions are the predicates or tuples in the EDB which took part in generating the new tuples in the IDB through the rules. Algorithm 1 is a recursive algorithm that computes all the EDB instances, and binary child-parent predicates by applying a predicate in the IDB with TGD rules, which consequently generate new tuples in another relation. This uses the table $RuleRecord()$, where the information for each tuple's generating rule is saved. The core concept is, taking those predicates which satisfy the $Body(rule)$ of the rule and its conjunction with generated tuple (d_i). The algorithm returns the ground EDB atoms (ψ), which participated in the generation of the predicate of IDB under consideration. These participating atoms are candidates for deletion since they are elements of the inconsistency in the IDB. We present the running example following Algorithm 1:

Algorithm 1 *Ground IDB*

```
Input: Predicate of IDB ($d_i$)
Output: Set of EDB predicates and Parent-Child predicates $\psi$
 1: procedure GROUNDIDB($d_i$)
 2:     $\psi \Leftarrow \varnothing$
 3:     $\Omega \Leftarrow RuleRecord(d_i)$                    ▷ rules that generated predicate $d_i$
 4:     for all $\sigma \in \Omega$ do
 5:         for all $d : (d \in D^H \vee d \in I^C)$ do
 6:             if $d \vDash (d_i \wedge Body(\sigma))$ then
 7:                 if $d \in IDB(I^C)$ then
 8:                     $\psi \Leftarrow \psi \cup GroundIDB(d)$
 9:                 else
10:                     $\psi \Leftarrow \psi \cup \{d\}$
11:                 end if
12:             end if
13:         end for
14:     end for
15:     return $\psi$
16: end procedure
```

In this algorithm, the search procedure for all the ground atoms start with the generated atom provided as input (d_i) and follows the Depth-First-Search (DFS) technique. For DFS we know that the worst case running time is $\mathcal{O}(|edges| + |nodes|)$. In our case, the number of rules are $|\Omega|$ which are like edges and number of predicates in the dimension and data tables are $(|D^H| + |I^C|)$ which are like nodes. So the complexity of Algorithm 1 is: $\mathcal{O}(|\Omega| + (|D^H| + |I^C|) * \tau)$ where, τ is the time taken by the query $d \vDash (d_i \wedge Body(\sigma))$. This is a Datalog query with the worst-case combined (data and expression) complexity of EXPTIME [72].

From Figure 1.2, rule:

$\sigma : AdmDrug(t, pb, d; p, ag), PersonSpec(pb, sp), DrugType(d, dt)$
$\rightarrow \exists am : Bills(t, sp, dt; p, am)$

generates the 4th tuple *Bills(16-Dec-17, Clinical, Restricted, Harry, N/A)* in the *Bills* table and hence this generated tuple is a part of the IDB. Now if this particular tuple is found as the source of inconsistency, then we create a conjunctive formula consisting of that tuple and the *Body(σ)* and among all the tuples and dimensional (parent-child) predicates whichever satisfies the formula (or formulas if many rules have created the same tuple) are ground atoms. The instance of the rule that created the tuple *Bills(16-Dec-17, Clinical, Restricted, Harry, N/A)* is:

$\sigma : AdmDrug(16\ Dec\ 17, David, Santonin; Harry, 15), PersonSpec(David, Clinical),$
$DrugType(Santonin, Restricted) \rightarrow \exists am : Bills(16\ Dec\ 17, Clinical, Restricted; Harry, am)$

So, the following 3 items (1 tuple and 2 dimensional predicates) are ground atoms.

1. $AdmDrug(16\ Dec\ 17, David, Santonin; Harry, 15)$

2. $PersonSpec(David, Clinical)$
3. $DrugType(Santonin, Restricted)$

4.2.2 Source of Inconsistency

A common assumption here is that the dimensional instance and the dimensional constraints are consistent in isolation. Inconsistencies may arise when instances and dimensions are applied and co-exist together. To detect inconsistencies we check the predicates whether they satisfy the constraint. In Algorithm 2, we identify the set of predicates that violate the constraints, hence form the super set of candidates for deletion. This algorithm uses negation of the constraints ($\neg\eta$) to detect inconsistency in the tuples [73].

As an input it takes dimensional instance (D^H, I^C) and dimensional constraints (η). After negating the constraint ($\neg\eta$), the algorithm looks for predicates which satisfy this negative constraint. If those are from EDB they directly go to the output set of predicates (μ) that violate the constraint (η), but if those are from the IDB they are grounded first.

Algorithm 2 *Source of Inconsistency*

Input: Dimensional Instance (D^H, I^C) and Dimensional Constraints (η)
Output: Set of predicates that violate the constraints

```
 1: procedure INCONSISTENTDATA($D^H, I^C, \eta$)
 2:     $\mu \Leftarrow \varnothing$                                  ▷ Set of Inconsistent Predicates
 3:     if $(D^H \wedge I^C) \nvDash \eta$ then
 4:         for all $d_b \in D^H$ and $d \in I^C$ do
 5:             if $(d_b \wedge \neg\eta)$ then
 6:                 $\mu \Leftarrow \mu \cup d_b$
 7:             end if
 8:             if $(d \wedge \neg\eta) \wedge (d \in EDB(I^C))$ then
 9:                 $\mu \Leftarrow \mu \cup d$
10:             end if
11:             if $(d \wedge \neg\eta) \wedge (d \in IDB(I^C))$ then
12:                 $\mu \Leftarrow \mu \cup GroundIDB(d)$
13:             end if
14:         end for
15:     end if
16:     return $\mu$
17: end procedure
```

This algorithm checks for inconsistency through the Datalog query: $(D^H \wedge I^C) \nvDash \eta$, which is of EXPTIME complexity in the worst case. Then for each predicate in the dimension and the data table identifies whether it is part of EDB or IDB. The checking is done in linear time but the IDB part may call $GroundIDB()$ function which is again an EXPTIME-complete program [72].

Recall in the preliminaries section of Chapter 3 the constraint "Restricted Drugs must be prescribed by a full-time doctor" which is expressed as:
$\eta_1 = AdmDrug(t, pb, d; p, ag), DrugType(d, Restricted), PersonSpec(pb, sp),$
$SpecDiv(sp, div),\ PersonContract(pb, con) \Rightarrow (div = Doctor), (con = FullTime)$
and
$\eta_2 = Bills(t, sp, dt; p, am), (dt = Restricted), PersonSpec(pb, sp),$
$SpecDiv(sp, div), PersonContract(pb, con) \Rightarrow (div = Doctor), (con = FullTime)$

From classical logic we know, $\neg(P \Rightarrow Q) \equiv P \wedge \neg Q$

Now, if $P = (A \wedge B)$ and $Q = (C \wedge D)$, the above equivalence becomes, $(A \wedge B) \wedge \neg(C \wedge D)$. After applying De Morgan's law: $(A \wedge B) \wedge (\neg C \vee \neg D)$.

Following the concept above, the negation of the constraints of the running example, can be written as:
$\neg\eta_1 = AdmDrug(t, pb, d; p, ag) \wedge DrugType(d, Restricted) \wedge PersonSpec(pb, sp) \wedge$
$SpecDiv(sp, div) \wedge\ PersonContract(pb, con) \wedge (div \neq Doctor \vee con \neq FullTime)$
and
$\neg\eta_2 = Bills(t, sp, dt; p, am) \wedge (dt = Restricted) \wedge PersonSpec(pb, sp) \wedge$
$SpecDiv(sp, div) \wedge PersonContract(pb, con) \wedge (div \neq Doctor \vee con \neq FullTime)$

As per Algorithm 2, nine predicates which satisfy the above negation of the constraints are the expected sources of inconsistency, as shown in Table 2.1. Among these the predicates which are part of IDB, have been grounded using Algorithm 1. We compute the list of tuples and dimensional child-parent predicates that cause the inconsistency against the constraints. Any subset of this set form the candidate for deletion to achieve overall consistency.

4.3 Weight Generation

As the OMD model consists of multiple data tables, dimensions, rules and constraints, it is very difficult to assign relative priority to the predicates. Sometimes numerical weights can be assigned based on the features of the deletion candidates but again that is humanly impossible if the number of predicates increases. Algorithm 2 identifies the erroneous predicates, thereby generating a set of possible candidates for deletion. One straight forward way is to delete all of the candidates, but our goal is to find a minimal weight subset of candidates, deleting them would satisfy the constraint by the rest of the predicates. To assign the weights, we need to explore criteria to select suitable candidates.

4.3.1 Deletion Criteria

Deletion is an aggressive operation on databases. In data repair techniques, we try to minimize negative impact on the existing information, and that is why, our focus is to delete minimally. So, here our approach is to identify deletion candidates with minimal weights. We define six criteria below, to compute weights, where higher weights indicate greater importance. The notion of importance for deletion candidates here, is relative reliability and the impact on the OMD model, if deleted. These are heuristic based assumptions, which can be verified by an expert user for correctness.

1. **Path distance from ground atoms (PathGround):** Each predicate is either part of the EDB or the IDB. Predicates that are part of IDB are grounded using Algorithm 1. This feature measures the total path of transformation of the created atom through rules.

   We define a cost function that counts the number of rules involved in creating an atom. The more rules that are involved in creating an atom, the greater the impact of deleting that atom, and hence, we assign a larger weight.

2. **Number of ground atoms (Grounds):** We define a cost function, that traces from IDB to EDB, and computes the total number of ground atoms that are participating in the generation of a new tuple.

   If there are many ground atoms created an atom through the rules, the common assumption is, that created atom carries more weight than the atoms which are created by relatively fewer ground atoms.

3. **Dimensional distance from the leaves (Dimdist):** If the predicate is a child-parent predicate of the dimension, Dimdist measures the maximum height from the leaf level.

   As per the common structure of the dimensions, the higher the position of the predicates the more reliable general concepts they represent. Deleting such predicate would impact all the predicates underneath it. As we want to keep intact the most reliable one in comparison to the less reliable one, we assign higher importance to them.

4. **Number of dimensional child (Children):** We define a weight function that counts the number of descendants for a candidate node.

   The intuition is that if a dimensional predicate is deleted, then these children nodes and their associated tuples/rules/constraints will be impacted. That means, the child nodes will be dangling nodes without connection to the parents, along with rules and constraints which use that predicate will no longer be valid. So, we are assigning more weights to those predicates, which have many child nodes compared to the ones which have fewer child nodes.

5. **Number of rule appearances (RuleAppear):** We assign weights by counting the number of rules a candidate predicate appears in.

   Deleting predicates associated with the rules will invalidate those rules. The larger the number of rule appearance, the larger the impact, and the larger the weight.

6. **Number of constraint appearances (ConAppear):** We define a weight function, that counts the number of constraints in which the candidate predicate appears in.

   Deleting such predicate means, that predicate will be false in those constraints.

The larger the number of constraint appearance, the larger the impact, and the larger the weight.

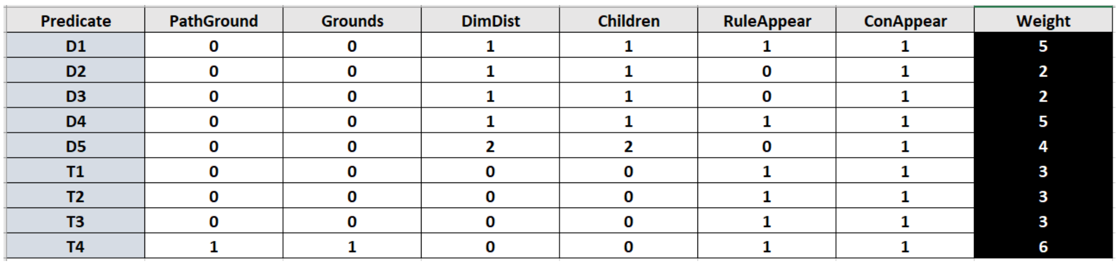
To consider all these features in a principled way, we chose the CRITIC method for the following reasons [29]:

1. The computed weights incorporate both contrast and conflict among the criteria values.
2. The evaluation matrix models consider pairwise comparisons among the criteria.
3. To compute weights, we consider the interaction between conflicting criteria, as well as criteria that are pairwise independent.
4. The CRITIC method performs well with correlated criteria.

In our study, we observe that several of our defined criteria do exhibit correlations. For example, the distance from the ground, and the number of atoms in the ground can be correlated; the children and the tree height can also be correlated, similar to a binary tree, if the height of the tree is higher it has more children and vice versa.

After we apply this CRITIC method in the running example with nine predicates which are sources of inconsistency, we get the nine weights $[2, 2, 3, 3, 3, 4, 5, 5, 6]$. For example, in the Figure-4.2 below, we see that, $D1$ is a predicate. This is actually an edge of the dimensional instance of "Drug" dimension, $D1 = DrugType(Santonin, Restricted)$ (Figure 2.6). Just because this predicate is not generated through TGD rules, it's $PathGround = 0$ and $Grounds = 0$. But $D1$ in the dimension, is at the bottom level, so, $DimDist = 1$ and $Children = 1$. It also appears once in the running example's rule (Figure-2.7) and constraint (Figure-2.8), resulted in $RuleAppear = 1$ and $ConAppear = 1$.

After we get this matrix with predicates as rows and six criteria as columns we apply the CRITIC method steps (Equation 2.1 to 2.10) which produces the weights as shown in the output table (Figure 4.2). Now we feed this sorted weights to Algorithm 3.



Predicate	PathGround	Grounds	DimDist	Children	RuleAppear	ConAppear	Weight
D1	0	0	1	1	1	1	5
D2	0	0	1	1	0	1	2
D3	0	0	1	1	0	1	2
D4	0	0	1	1	1	1	5
D5	0	0	2	2	0	1	4
T1	0	0	0	0	1	1	3
T2	0	0	0	0	1	1	3
T3	0	0	0	0	1	1	3
T4	1	1	0	0	1	1	6

Figure 4.2: Weight Generated by CRITIC method

4.4 Minimal Weighted Repair

Recall that our objective is to improve consistency in an OMD model that does not satisfy the constraints and rules, which is logically expressed as $I^M \nvDash \eta$. Since we assume η is correct, the source of inconsistency is in I^M.

Our objective is to find a version (I) of I^M such that, $I \vDash \eta$. Since we consider only deletion of tuples in the database, or child-parent predicates in the dimension as possible repair operations, we select those tuples and predicates with the lowest weights. Specifically, if $\mathit{Repair}(I^M, \eta)$ is a function that takes the instances, the constraints, and returns the set of repaired versions after deletion, i.e. $\mathit{Repair}(I^M, \eta) = \{I, I_1, I_2, ...I_n\}$, then we seek the repair leading to an instance (version of I) where the sum of weights from the deleted predicates and tuples is minimal.

We have data, dimensions, rules and constraints. We have assumed the constraints are correct, implying that any inconsistencies lie in the data, or the dimension, or both. We know that in a logically consistent system, a predicate cannot satisfy both the condition and it's negation. So, to identify the source of inconsistency, we compute the logical negation of the constraint to obtain the predicates that satisfy the negation of the constraint (Algorithm 2).

After inconsistency detection and grounding, we have the set of predicates which are candidates for deletion to restore consistency. We use the CRITIC method to assign weights to the tuples, the parent-child predicates in the dimension. We then evaluate different subsets of those tuples and binary predicates to determine the minimal weight subset for deletion.

We use an auxiliary look-up table $WeightAtom()$, that stores the generated weights for each inconsistent atom. As an input it takes atoms and return their respective weights.

Formally, if set of inconsistent predicates $\mu : \mu \subseteq I^M$, then deletion candidate $\varphi : \varphi \subseteq \mu$, if deleted from μ the resulting data, dimensions, and constraints are consistent keeping the weights of φ minimal, $I^M \backslash (\mu \backslash \varphi) = I$. Hence, if I is the minimal repair, logically we can express:

$$[\forall \varphi' : (\varphi' \subseteq \mu) \wedge (I^M \backslash \varphi' \vDash \eta) \;\; \rightarrow \sum_{\forall a : a \in \varphi} WeightAtom(a) \leq \sum_{\forall a' : a' \in \varphi'} WeightAtom(a')]$$

This means, the summation of weights of all the items of the set φ, is smaller than, summation of weights of all the items of any other subset φ', where deleting φ' or φ from I^M, makes I consistent with respect to the constraint η.

4.4.1 Min-Sum a Dynamic Programming Based Approach

We focus now on the problem of searching a subset of predicates from the inconsistent sources. If these predicates are deleted, the OMD model will be consistent. We reduce the problem into a special type of subset sum problem. This Min-Sum algorithm greedily searching for minimal weights from sum $= 1$ towards sum of all weights of the items in that particular super-set of predicates under consideration. It then dynamically pre-computes the sums and when a new item is added, it reuses previous sums and add on top of the current sum rather than regenerating them from scratch [74]. The algorithm generates multisets of weights in the ascending order of their sums. For a multiset of weights $(2, 2, 3, 3, 3, 4, 5, 5, 6)$ the sub multisets would be generated as:

$(2), (2), (3), (3), (3), (2, 2), (4), (2, 3)$.........$(2, 2, 3, 3, 3, 4, 5, 5, 6)$

Each of these sub multisets of weights connected to the predicates, then undergoes testing of whether removing them from the model makes it consistent or not with the provided constraint (η). If it is found consistent, then that multiset is being assigned as candidate for deletion.

For the pre-computation table, Algorithm 3 uses the recurrence relation:

$$M[i, w] \Leftarrow M[i-1, w] \vee M[i-1, w-w_i]$$

Here, M is a two-dimensional table containing boolean entries. The columns are indexed from 0 to sum of the all weights S, and rows are indexed from 0 to the number of weights n. Each row contains the weight of the respective source of inconsistency predicates, w_1 to w_n. Each entry in the boolean table is “true”, if and only if, the solution to the sub-problem exists there, otherwise “false”. The idea is similar to the subset-sum dynamic programming algorithm, that is, if subtracting weight at i^{th} row w_i, from any sum in the column (w), there exists a weight $(w - w_i)$, which has already been calculated, there is no need to solve that problem again. We can use that sub-problem solution $(w - w_i)$ and add w_i to get this current column sum w. To keep track of the weights that sum up to the column w, we use another table M'which corresponds to set of integers of weights, in the corresponding “true” value of the table M. For multiset, the cell in this tables would contain multiple entries and the iteration would run upward along the columns.

Algorithm 3 *Deletion Candidate Search*

Input: Set of inconsistent predicates (μ) and constraints (η)
Output: Minimum weight atoms for deletion (φ)

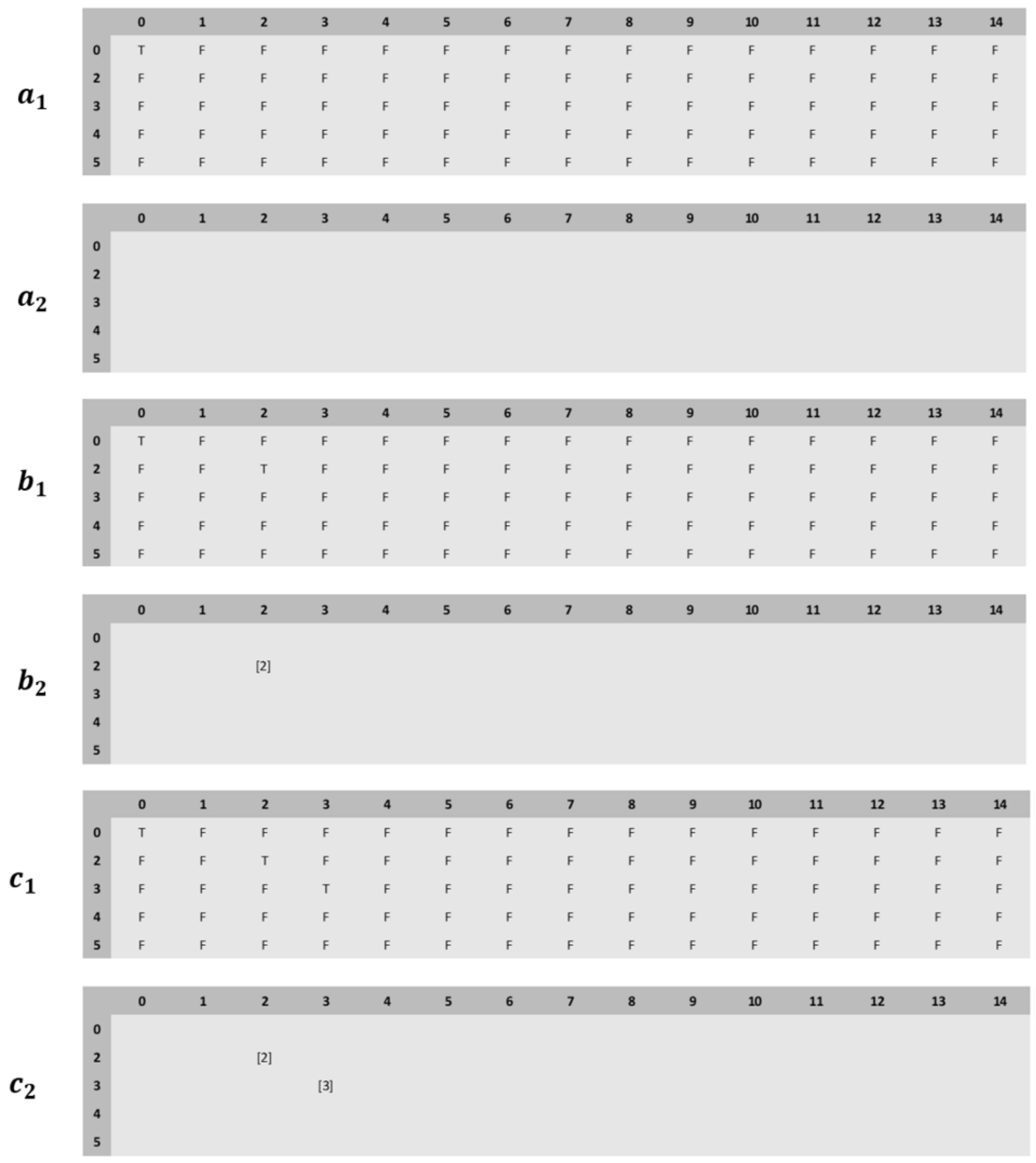
```
1: procedure DeletionCandidate($\mu, \eta$)
2:     $W \Leftarrow \varnothing$                                        ▷ Set of weights
3:     for each $m \in \mu$ do
4:         $W \Leftarrow W \cup WeightAtom(m)$
5:     end for
6:     $S = 0$                                                ▷ Sum of all weights
7:     for each $w_i \in W$ do
8:         $S = S + w_i$
9:     end for
10:    $n \Leftarrow Count(W)$                          ▷ Number of inconsistent atoms
11:    $M[0..n, 0..S] \Leftarrow$ "$False$"                      ▷ Boolean Table
12:    $M[0, 0] \Leftarrow$ "$True$"
13:    $M'[0..n, 0..S] \Leftarrow \varnothing$                           ▷ Set table
14:    for all $w \leftarrow 0..S$ do                     ▷ Iterate over column sums
15:        for all $i \leftarrow 1..n$ do                    ▷ Iterate over row weights
16:            $\varphi \Leftarrow \varnothing$                      ▷ Set of atoms to be deleted
17:            $M[i, w] \Leftarrow M[i-1, w] \vee M[i-1, w-w_i]$
18:            if $M[i, w] \neq M[i-1, w]$ then
19:                $M'[i, w] \Leftarrow M'[i-1, w-w_i] \cup w_i$
20:            else
21:                $M'[i, w] \Leftarrow M'[i-1, w]$
22:            end if
23:            if $M'[i, w] \neq \varnothing$ then
24:                for all $m' \in M'[i, w]$ do
25:                    $\varphi \Leftarrow \varphi \cup AtomWeight(m')$
26:                    $\delta \Leftarrow \mu \backslash \varphi$              ▷ Subset removed from superset
27:                    if $\delta \vDash \eta$ then        ▷ Checking consistency after deletion
28:                        return $\varphi$            ▷ Deletion candidate of minimal weight
29:                    end if
30:                end for
31:            end if
32:        end for
33:    end for
34: end procedure
```

This algorithm uses pre-computation tables (also shown in the forthcoming example) similar to the dynamic programming based subset sum algorithm [74]. Initial weight lookup and summation of weights take linear time to compute. But, the core part of the Algorithm 3 (two nested for loops at steps: 14, 15) in worst case, goes through the whole table of all the $|W|$ weights as rows and $|S|$ sums of weights as columns. Again, in each iteration there is a Datalog$^{\pm}$ query to check satisfiability ($\delta \vDash \eta$) which is a EXPTIME-complete problem itself [75]. If we consider this particular query is taking τ time in the worst case, then the complexity of this algorithm can be shown as: $\mathcal{O}(|S| \times |W| \times \tau)$.

If the number of the records and the size of the dimensions are finite, then the sum of weights S and the number of weights n will be finite too, and hence, the algorithm will terminate. The algorithm uses a sorted summation list, and starts checking from minimum weight 0 until it returns the minimal weight. If there are more than one combinations of the equal, sum of minimal weights found, it returns the first one obtained with the smallest individual weights possible. Cardinality minimal of the subset items, is not considered here, but if we want that alongside weights, we can modify this algorithm as lattice based approach, where number of atoms will increase from null to all in each layers. Besides, to choose between cardinality and weight, one has to be prioritized but we can explore that lattice based algorithm further, in the future research.

Let's look at a simple example to demonstrate how Algorithm 3 works: We define two matrices, one boolean M and another one M' for weight subsets, as mentioned in the algorithm. In each of the steps a to f in the Figure-(4.3, and 4.4), the upper matrix (e.g. a_1) represents boolean (M) and bottom one (e.g. a_2) represents weight subset (M'). To denote, each of the entries of these matrices, we will use row and column labels *(Row Label, Column Label)* instead of indices for simpler explanation.

Now, say, we get a 4-items set of predicates $\{\mu_1, \mu_2, \mu_3, \mu_4\}$ which are sources of inconsistency in an OMD model and the sorted weights for these predicates are $\{2, 3, 4, 5\}$, generated through the CRITIC method. We also assume, the minimal weight for deletion is $(2 + 5) = 7$. Consistency will be restored by deleting $\{\mu_1, \mu_4\}$ predicates. So, in the matrices both in M and M', weights $\{0, 2, 3, 4, 5\}$ will be at the rows, and columns will start from 0 to sum of all the weights, $(2 + 3 + 4 + 5) = 14$. Now any cell in M which is "true", the corresponding entry in M' contains the set of weights, whose sum is equal to the column label of that cell under consideration.



a_1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
3	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
4	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
5	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

a_2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0															
2															
3															
4															
5															

b_1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F
2	F	F	T	F	F	F	F	F	F	F	F	F	F	F	F
3	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
4	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
5	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

b_2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0															
2			[2]												
3															
4															
5															

c_1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F
2	F	F	T	F	F	F	F	F	F	F	F	F	F	F	F
3	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
4	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
5	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

c_2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0															
2			[2]												
3				[3]											
4															
5															

Figure 4.3: Algorithm-3 Steps in the matrix

For this example, in Figure (4.3 and 4.4), Algorithm 3 Steps are:

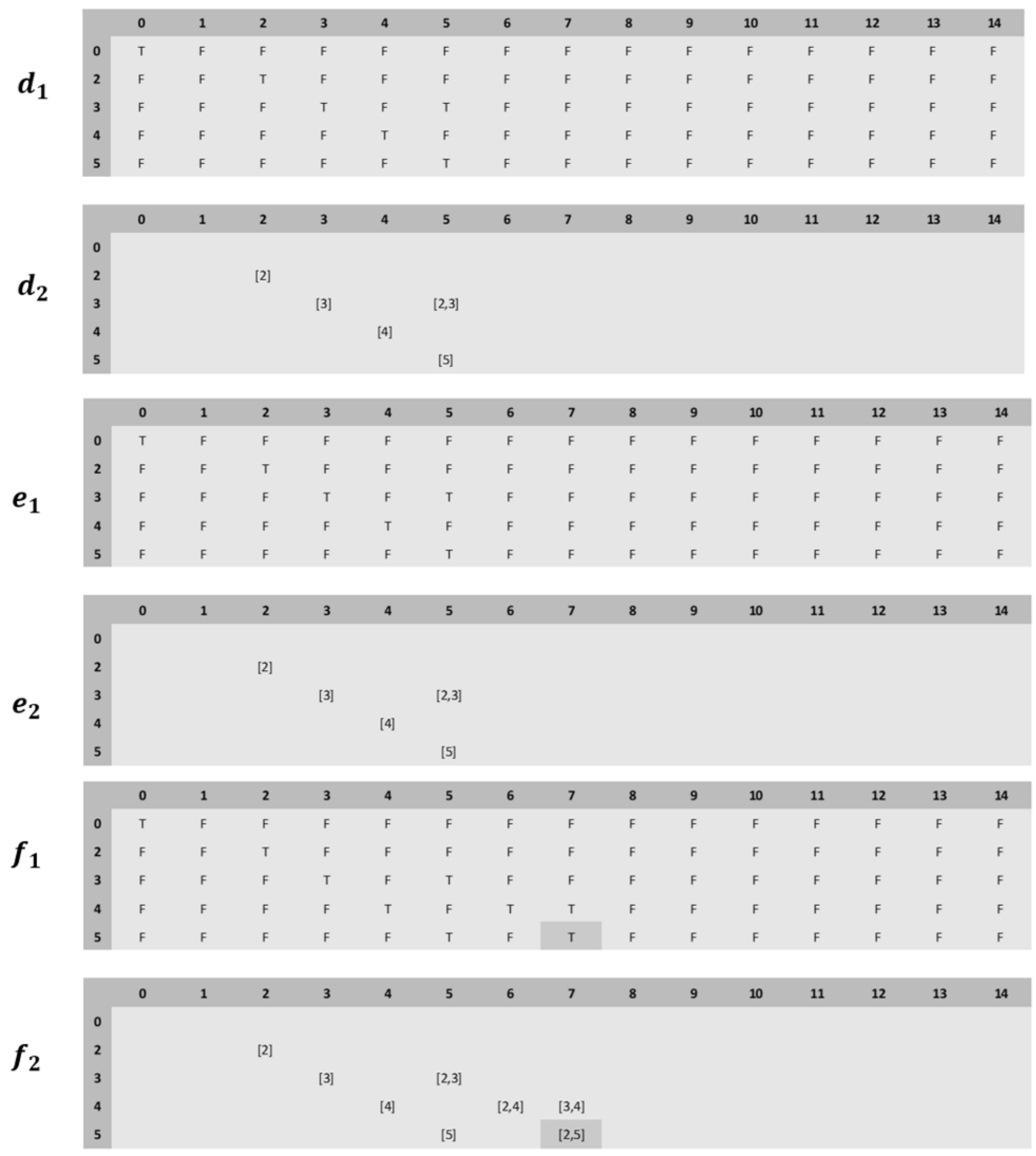
1. Initial Stage: Figure 4.3(a) Boolean matrix (a_1), all false “F” except (0,0) and subset matrix (a_2) is of empty set (∅).
2. In both matrices, Figure 4.3(b) from position (2,2), as row starts from weight 2, going up one row and minus 2 position, takes to position (0,0) which is true ,“T”

in boolean matrix (b_1), so (2,2) cell will be “T” and in the subset marix (b_2) it will contain: [0,2] = [2] which is the weight of μ_1. So, we need to see whether the remaining predicates $\{\mu_2\,\mu_3, \mu_4\}$ satisfies η or not. As per our assumption of the minimal weight in this example is 7, it does not satisfy η.

3. Next one Figure 4.3(c) is (3,3) position. Going up one row, that is in (2,3) then minus 3 will take to position (2,0) which is False. Now going up from (2,0) until reach any T or end, we reach at (0,0) which is “T”, so (3,3) position will be 3 and in the subset it will contain: [0,3] = [3] which is the weight of μ_2. So, we need to see whether $\{\mu_1, \mu_3, \mu_4\}$ satisfies η or not. No, it does not satisfy the constraint. (Figure: 4.3(c))

4. In this way in Figure 4.4(d), from row 3, we reach say at 5 in the column. Then going back until 3 in the column, following the same procedure we find 2. So, (3,5) position is “T” in boolean (d_1)and in the set (d_2) it will have [2,3]. So if we delete them i.e. [2,3], the rest $\{\mu_1, \mu_4\}$ does not satisfy η.

5. Similarly [5], [2,4] and [3,4] weight deletion will not satisfy the constraint (Figure: 4.4(e,f)). But at position (5,7) in the set matrix (f_2) we get [2,5], deleting them satisfies the constraint η. So, our solution is: delete $\{\mu_1, \mu_4\}$ of weights [2,5], and the rest: $\{\mu_2, \mu_3\}$ satisfies η.



d_1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F
2	F	F	T	F	F	F	F	F	F	F	F	F	F	F	F
3	F	F	F	T	F	T	F	F	F	F	F	F	F	F	F
4	F	F	F	F	T	F	F	F	F	F	F	F	F	F	F
5	F	F	F	F	F	T	F	F	F	F	F	F	F	F	F

d_2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0															
2			[2]												
3				[3]		[2,3]									
4					[4]										
5						[5]									

e_1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F
2	F	F	T	F	F	F	F	F	F	F	F	F	F	F	F
3	F	F	F	T	F	T	F	F	F	F	F	F	F	F	F
4	F	F	F	F	T	F	F	F	F	F	F	F	F	F	F
5	F	F	F	F	F	T	F	F	F	F	F	F	F	F	F

e_2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0															
2			[2]												
3				[3]		[2,3]									
4					[4]										
5						[5]									

f_1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F
2	F	F	T	F	F	F	F	F	F	F	F	F	F	F	F
3	F	F	F	T	F	T	F	F	F	F	F	F	F	F	F
4	F	F	F	F	T	F	T	T	F	F	F	F	F	F	F
5	F	F	F	F	F	T	F	T	F	F	F	F	F	F	F

f_2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0															
2			[2]												
3				[3]		[2,3]									
4					[4]		[2,4]	[3,4]							
5						[5]		[2,5]							

Figure 4.4: Algorithm-3 Steps in the matrix

The above algorithm guarantees optimum solution, as it explores each and every possible combination of the inconsistent predicates from the minimum sum of their weights. So, it is impossible to leave behind any subset solution which has the minimal weight. This algorithm was developed with the assumption that, most of the time, we do not need to generate all the models of the theory to take the minimum one if we can generate them in the ascending order of their sums. As soon as we obtain the first combination of expected predicates, we consider those for deletion to restore consistency. In the worst case running time of this algorithm is exponential as we have to generate all the possible subsets. So, there was a need to design a faster algorithm which should also be scalable. We explored different Heuristic Algorithms like Swarm Intelligence, Simulated Annealing and found Genetic Algorithms has been successfully used in similar problems like Knapsack Problems [76]. Hence, we implemented a version of GAs to solve this problem.

4.4.2 A Genetic Algorithm Based Approach

We have a discrete optimization problem that involves selecting a subset from a set of weights that satisfies certain criteria i.e. deleting the predicates associated with those weights from the subset, will restore consistency in the OMD model. This is a bag of weights with duplicates. Not all the subsets which are subject to deletion, if deleted, will restore consistency. Our total solution space has feasible and infeasible solution regions. As the first step called the initial population generation, we utilize the fitness function such that its value not only ranks each solution but also sets an outlier mark for those which are not satisfying the given constraint. As mentioned earlier about the limitations of GAs, we also cannot guarantee the convergence with iterations in our version. But we start with the chromosome containing the superset of all the candidate inconsistent predicates, which is obviously a solution in the solution space. Then, we keep on selecting chromosomes of proper subsets randomly and calculate their fitness to evaluate as a possible solution with minimal weights.

In the discrete optimization problem, Genetic Algorithms (Figure: 2.9) usually consider chromosomes as binary strings which consist of 1s and 0s indicating whether the indexed item is selected. For example, if the set of predicates is $\{\mu_1, \mu_2, \mu_3, \mu_4\}$ which are the sources of inconsistency in an OMD model, and the respective weights for these predicates are $\{2, 3, 4, 5\}$, one chromosome could be $[1, 0, 0, 1]$, this represents selecting predicates $\{\mu_1, \mu_4\}$ with weights $\{2, 5\}$ for deletion. In our examples below, for the sake of simplicity, we show weight set $\{2, 5\}$ format instead of the binary set $[1, 0, 0, 1]$ format.

Fitness: The fitness function takes as input the chromosome consisting of the predicates which are selected for calculating fitness, then source of inconsistency, and the set of constraints. After excluding the predicates (κ) from the set of inconsistent predicates (μ), we check whether the constraints are satisfied. If the remaining predicates ($\mu\backslash\kappa$) satisfies the constraint (η), we return the sum of weights for all predicates in κ with the weights in W. If the predicates do not satisfy the constraint, we return the infinite number to indicate this is an unfit chromosome, hence, ignore this candidate. Formally, we define our fitness function as:

$$fitness(W) = \begin{cases} \sum_{i=0}^{|W|} w_i & , (\forall w_i \in W) \land (\mu\backslash\kappa \vDash \eta) \\ \infty & , (\mu\backslash\kappa \nvDash \eta) \end{cases}$$

This defined fitness function terminates as we always return a value. In the case when the constraint is satisfied, we iterate through a finite set of elements bounded by the size of W, and return the sum of the predicate weights. In the case the constraint is not satisfied, we simply return a large (infinite) value.

Algorithm 4 *Fitness*

Input: List of predicates under consideration, all inconsistent predicates, constraints
Output: Fitness score of the chromosome

```
1: procedure FITNESS($\kappa, \mu, \eta$)
2:     $W \Leftarrow \varnothing$                              ▷ Set of Weights
3:     for each $m \in \kappa$ do
4:         $W \Leftarrow W \cup WeightAtom(m)$
5:     end for
6:     $S = 0$                                  ▷ Sum of all weights of the sub multiset
7:     for each $w_i \in W$ do
8:         $S = S + w_i$
9:     end for
10:    $\delta \Leftarrow \mu \backslash \kappa$                        ▷ Subset removed from superset
11:    if $\delta \vDash \eta$ then                      ▷ Checking consistency after deletion
12:        return $S$
13:    end if
14:    return $\infty$
15: end procedure
```

In this algorithm, initial weight lookup and summation of weights take linear time to compute. But, the core part of the Algorithm 4 is a Datalog$^{\pm}$ query to check satisfiability ($\delta \vDash \eta$) which is a EXPTIME-complete problem itself [75]. If we consider this particular query is taking τ time in the worst case, then the complexity of this algorithm can be shown as: $\mathcal{O}(|\kappa| + |W| + \tau)$.

Algorithm 4 shows the fitness calculation. We take the predicates from the chromosome κ (i.e. potential deletion candidates) and drop those predicates from μ and check the consistency against the constraint without the remaining atoms in μ. If satisfied, we return the total weights S of all the predicates in κ. If not satisfied, we return a large integer, at least larger than sum of all the weights, to designate this chromosome into the infeasible solutions space of the population. We consider only those chromosomes, where deleting the items indexed there, will satisfy the constraint and their total weight is less than sum of all the weights of predicates in the source of inconsistency list. For example, if the weight list is like: $[1, 2, 3, 5, 10]$ and sum of these are $(1 + 2 + 3 + 5 + 10) = 21$. Given two chromosomes, say $[2, 3, 5]$ and $[1, 2, 3]$, deleting them both satisfies the constraint, then their respective fitness score is: $(2 + 3 + 5) = 10$ and $(1 + 2 + 3) = 6$. Any candidates which do not fall into this range, are assigned a score larger than 21 so that it is discarded. We consider low fitness scores as better chromosomes, as our objective is to find minimal weight.

Population: As we know that, not all the regions in the total solution space are feasible, we have to design this population initialization Algorithm 5 in a way so that it can only produce those chromosomes which are feasible. We utilize the fitness function to determine feasibility. It also takes into account the size of the population.

Algorithm 5 *Population Generation*

Input: Size, total list of inconsistent predicates, constraint
Output: Priority Queue of Chromosomes

```
 1: procedure POPULATION($N, \mu, \eta$)
 2:     $\Theta \Leftarrow \mu$                                        ▷ Priority Queue
 3:     for all $i \leftarrow 1..N$ do
 4:         $Max \Leftarrow Top(\Theta)$
 5:         $MaxWeight \Leftarrow Fitness(Max)$
 6:         $\kappa \Leftarrow RandomChromosome(\mu)$
 7:         if $Fitness(\kappa) < MaxWeight$ then
 8:             $Insert(\Theta, \kappa)$
 9:         end if
10:     end for
11:     return $\Theta$
12: end procedure
```

Algorithm 5, already contains the fitness function (Algorithm 4 with the worst case complexity $\mathcal{O}(|\kappa| + |W| + \tau)$. There is also a priority queue "Insert" function with the worst case complexity of $\mathcal{O}(logN)$ where N is the size of the queue. As these two functions run N times to generate the population, the overall complexity of this algorithm becomes $\mathcal{O}(N \times (|\kappa| + |W| + \tau + logN)$.

The population function returns a max priority queue of a user-defined size. The idea behind using the priority queue, is to narrow down solution space and cross-over area. Whenever any new chromosome is generated and found to be fit, then if its sum of weights are smaller than the max in the queue, it pops out the max item and inserts the new chromosome. For example, if the max priority queue consists of weights: $[10, 7, 3]$ and a new chromosome comes with the weight 15, which is feasible as it's less than $(10 + 7 + 3) = 20$. But in the max priority queue, the weight 10 is the maximum, so this candidate will not be inserted. If a new chromosome with fitness weight 5, that is less than 10, to keep the size of the queue 3, it will pop out the current max 10 and insert 5. Hence, the new priority queue will be $[7, 5, 3]$.

Crossover: The crossover breeds new chromosomes which are better in quality, that means they have better fitness score i.e. of lower value. Crossover takes features from both the parent chromosomes. Here it takes the max priority queue as input and produces the new chromosome as output.

Algorithm 6 *Crossover*

Input: Max Priority Queue of Population
Output: New Breed Chromosome

```
1: procedure CROSSOVER($PQ$)
2:     $\kappa_1 \Leftarrow RandomChromosome(PQ)$
3:     $\kappa_2 \Leftarrow RandomChromosome(PQ)$
4:     $\kappa \Leftarrow \kappa_1 \cap \kappa_2$
5:     return $\kappa$
6: end procedure
```

Algorithm 6, has just two constant time random chromosome selection functions and an union of two chromosome operation which has the running time of $\mathcal{O}(|\kappa|)$ where κ is the length of the chromosome.

The idea behind our crossover Algorithm 6 is that, those chromosomes which are of minimal weighted set of atoms, they have high probability of appearing in the super multi-sets containing them, where these super multi-sets if deleted, restores the consistency. To get minimal weights we can take the common atoms among the two parent chromosomes. For example, if the two feasible parent chromosomes are: $[1, 2, 3, 4]$ and $[2, 3, 5, 7, 8]$, then there is a possibility that the multi-set with the common items $[2, 3]$, is the minimal chromosome which has better fitness $(2+3) = 5$. So, we randomly choose two parents from the max-priority queue and produce a new chromosome by selecting the common predicates between them.

Mutation: Mutation is the technique to introduce new features which may or may not be present in the parents. In our context, this is just to mutate or change some bits such that it introduces new features outside of the current domain. Our mutation algorithm takes in the new breed generated from the crossover, changes a bit if applicable and produces the new child chromosome for fitness testing and adding to the priority queue.

Algorithm 7 *Mutate*

Input: Chromosome, Population
Output: Child Chromosome

```
1: procedure MUTATE($\kappa, Population$)
2:     $\kappa_1 \Leftarrow RandomChromosome(Population)$
3:     $\kappa_2 \Leftarrow RandomChromosome(Population)$
4:     $i \Leftarrow RandomeInteger(0, Length(chromosome))$
5:     if $\kappa[i] == 1$ then
6:         $\kappa[i] = (\kappa_1 \cap \kappa_2)[i]$
7:     end if
8:     return $\kappa$
9: end procedure
```

All the operations in the mutation function are of constant time, so the complexity of this algorithm is just $\mathcal{O}(1)$.

In Algorithm 7, mutation works in the population's feasible solution region and selects two of the fit chromosomes randomly which are not in the priority queue. Then randomly chose one position in the child (input) and matches that position with the two randomly selected chromosomes from the population. The algorithm changes the bit in the child with the one found in the randomly selected chromosomes, if they are equal. For example, if the child is $[1, 2, 3, 4, 5]$, and both of the randomly selected chromosomes do not have 4 in the 4^{th} position, then we mutate the child into: $[1, 2, 3, 5]$ by discarding 4, and if it is a good fit, we can insert this child into the queue.

Iteration of Genetic Algorithm: Finally we implement the iteration phase of the genetic algorithm, where the crossover and mutation continue running until it converges to a point where no other improvement is observed. Other termination criteria such as number of iterations or solutions or even fixed running time can also be used. The benefit of using GA is, at any iteration, there is a solution available. Although it may not be the optimal one, over subsequent iterations, the solution improves.

Chapter 5

Experimental Evaluation

This research is the first step towards inconsistency restoration of ontology multi dimensional data models. This is also based on Datalog$^{\pm}$, for which there are not many matured tools or libraries readily available. So, we developed a working prototype and synthetic datasets to test our algorithms. Our objective here is to discuss about the system used for implementation and associated challenges, synthetic datasets preparation and description, validation of assumptions and performance evaluation of the designed algorithms.

5.1 System Configuration

We ran our experiments using virtualization of the Linux server (Architecture x86-64) with 32GB RAM, running Linux Mint 19.1 operating system on Intel Xeon (CPU E5-2687W v4 @ 3.00GHz). All of the implementations were done in the Python programming language, except weight generation algorithm which was done in R. To simulate the Datalog$^{\pm}$ behavior, we used python's plain Datalog library known as pyDatalog [77].

5.2 Datasets

Our datasets are the based on the running example we created but much larger in size to resemble practical usage. The information to enrich dimensions are also inspired from real world knowledge bases.

We introduced two dimensions "Person" and "Drug" and their dimensional instances as a toy example (Figure-1.3). Here, we have kept the same dimensional schema but extended dimensional instance, indicating the number of instances in each category by a circled integer (Figure-5.1).

"Person" has 2 Divisions, "Doctor" and "Nurse". "Doctor" has 7 specializations and "Nurse" has 3, that is in total 10 specialities, therefore circled 10 displayed beside category "Speciality" in the Figure-5.1. Specialities of the Doctors are: "Cardiologist", "Pediatrician", "Medicine", "Gynecologist", "Surgeon", "Dermatologist", "Neurologist". Specialities of the Nurses are: "Clinical", "Forensic", "Orthopedic". For the "contract",

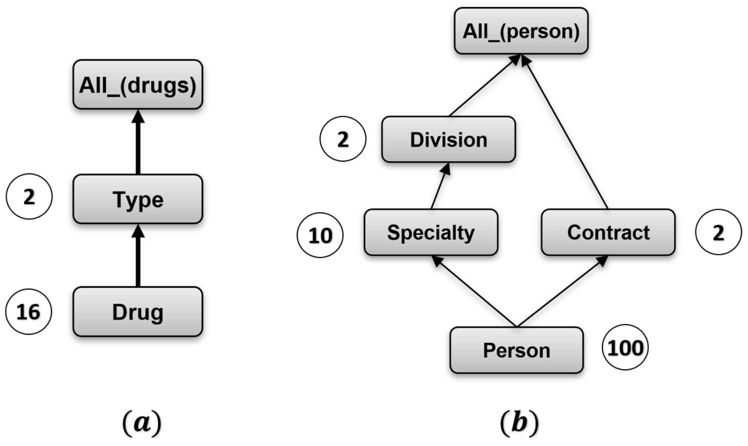
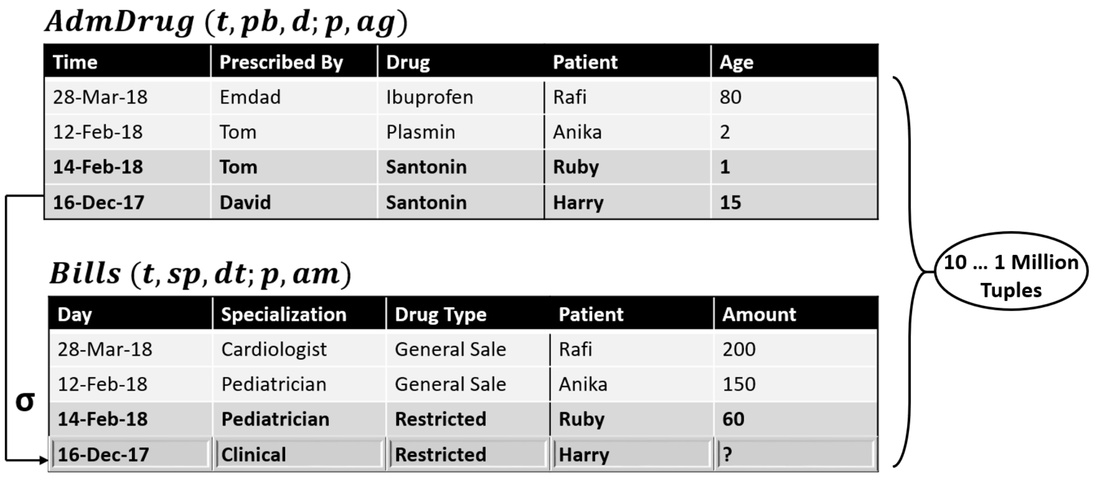


Figure 5.1: Drug (a) and Person (b) Dimension (with # of instances)

it can be of "Full-time" or "Intern" i.e. 2 types of contracts. At the bottom, we have “Person” category containing names of the 100 doctors or nurses.

Drugs are of 2 types "Restricted" and "General Sale". There are total 16 drugs are in “Drug” category; 3 of them are listed as "Restricted" type and 13 of them are of "General Sale" type. Restricted drugs are: "Santonin", "Meclozine", "Ketamine" and general sale drugs are: "Ibuprofen", "Plasmin", "Carprofen", "Histamine", "Lipitor", "Nexium", "Plavix", "Abilify", "Seroquel", "Singulair", "Crestor", "Actos", "Epogen".



$AdmDrug\ (t, pb, d; p, ag)$

Time	Prescribed By	Drug	Patient	Age
28-Mar-18	Emdad	Ibuprofen	Rafi	80
12-Feb-18	Tom	Plasmin	Anika	2
14-Feb-18	**Tom**	**Santonin**	**Ruby**	**1**
16-Dec-17	**David**	**Santonin**	**Harry**	**15**

$Bills\ (t, sp, dt; p, am)$

Day	Specialization	Drug Type	Patient	Amount
28-Mar-18	Cardiologist	General Sale	Rafi	200
12-Feb-18	Pediatrician	General Sale	Anika	150
14-Feb-18	**Pediatrician**	**Restricted**	**Ruby**	**60**
16-Dec-17	**Clinical**	**Restricted**	**Harry**	**?**

Figure 5.2: Dataset Tables (with # of instances)

For the data tables, “Administer Drug” and “Bills” the same rule and schemas (Figure-2.5) were kept, but we generated different sizes from 10 to millions of records (Figure-5.2) for testing the algorithms developed for restoring consistency with the dimensions built above.

5.3 Source of Inconsistency

We implemented the source of inconsistency search Algorithm-2. This includes detection and searching the ground atoms which are responsible for the inconsistency. This part is developed using “pyDatalog” library and search procedure expressed in the form of query answering. As we can see in the Figure-5.3, it is almost linear in nature, that means, the time (seconds) required to get all the ground predicates is proportional to the number of records in the dataset.

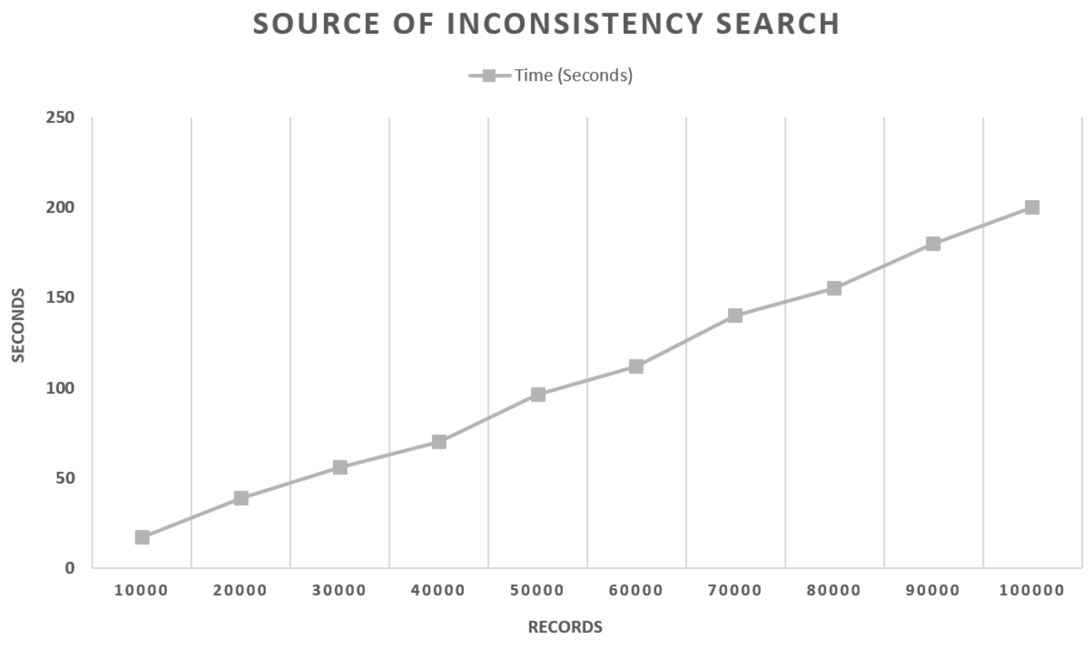


FIGURE 5.3: Source of Inconsistency Search Performance

5.4 Weight Generation

We have introduced 6 criteria and after obtaining the deletion candidate predicates, we arranged them in a matrix (Figure-4.2) and used the CRITIC method to generate the weights. Figure-5.4 shows the runtime (in milli-seconds) as we scale the number of predicates. We used the R language for this purpose. All the steps in CRITIC method are mathematical functions operating on the single matrix, so the calculations are very fast and scalable. For 1,000,000 predicates it took only 2.5 seconds to generate the weights.

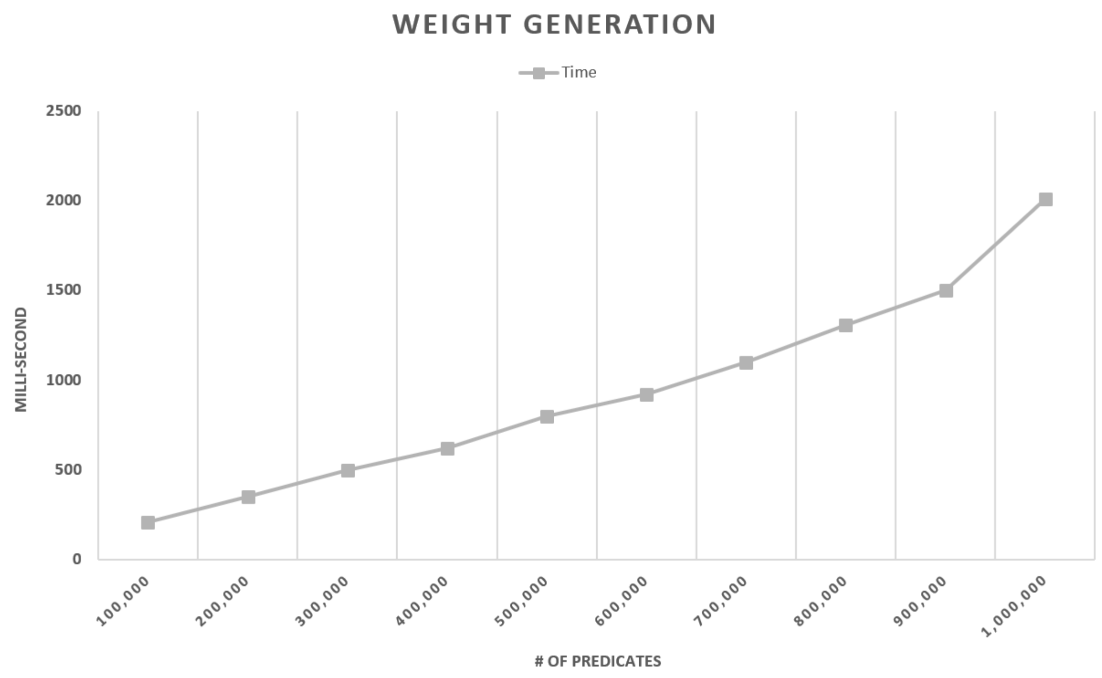


Figure 5.4: Weight Generation

5.5 Deletion Candidate Search

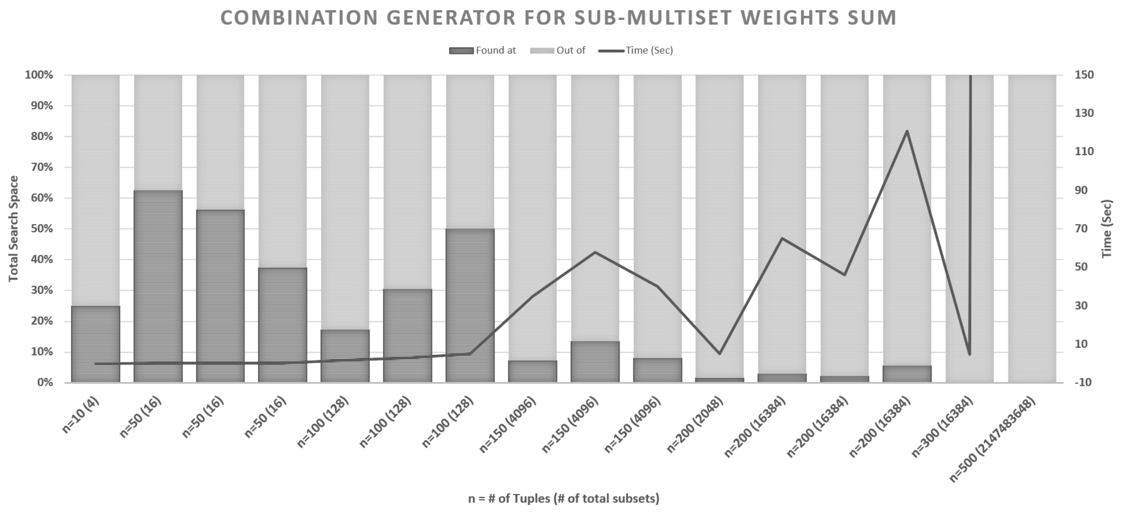


Figure 5.5: Greedy and DP Based Algorithm Performance

After receiving all the predicates, identified as the sources of inconsistency from Algorithm-2, we execute Algorithm-3, which computes the deletion candidates with minimal weight sum to the end user. Figure-5.5 displays the search space, number of tuples with subsets and performance (time), in the single graph to help us easily comprehend their interrelationship associated with Algorithm-3. It shows the running times (right Y-Axis) on varying input size and also how many of the predicates (in percentage, left Y-Axis) are being considered as deletion candidates among the total inconsistent predicates. Here, X-Axis shows different input sizes, that means number of tuples in the fact tables. Inside the parenthesis, it shows the total number of subsets required to generate in the worst case. For example, the 2nd data point, $n = 50(16)$, expresses that, there were 50 records and out of them 4 were found as sources of inconsistency i.e. number of subsets to generate at worst case was 2^4 or 16. As it was mentioned earlier that, the idea behind developing this Greedy-DP based algorithm was to generate combinations of sub-multisets in the ascending order of their sums, so that all the models or sets were not required to generate which could be exponentially growing. This graph (Figure-5.6) is actually empirical evidence that not all the models need to be generated. In our experiment, in all of the cases, out of all the models i.e. out of 100% (lighter part), around 20% (darker parts) were required to generate to get the minimal weight. Figure-5.7, shows that time is proportional to the number of subsets generated. For this experiment with Algorithm-3, time performance measurement is actually trivial, because even though there could be 1 million records but the first smallest weight could be the only one deletion candidate and it would take less than a second to find it, whereas with 500 records only, if the deletion candidate is far away from the minimum weight, it could take longer time to find the expected subset of minimal weight.

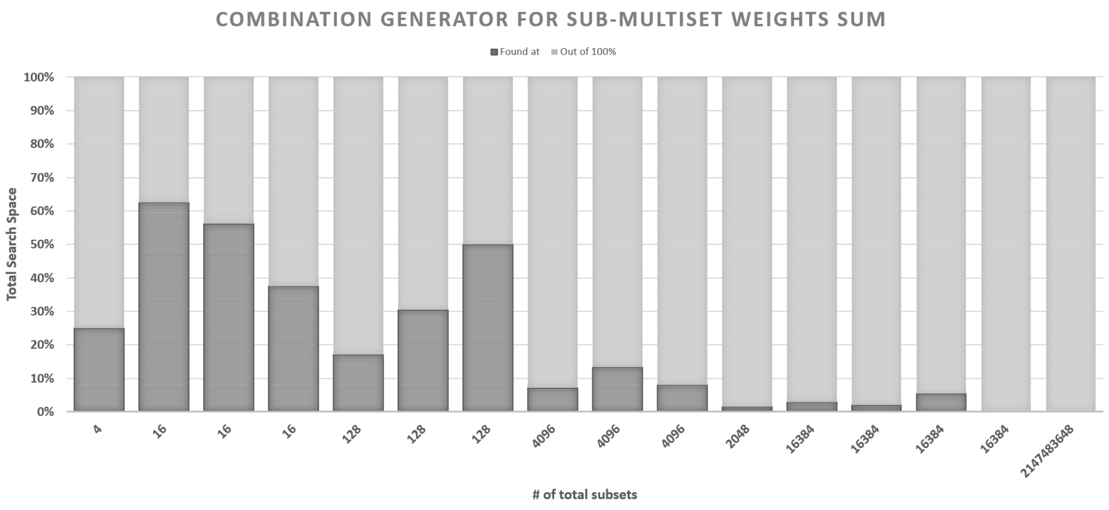


FIGURE 5.6: Total Number of Set Generated to Find Solution in Search Space

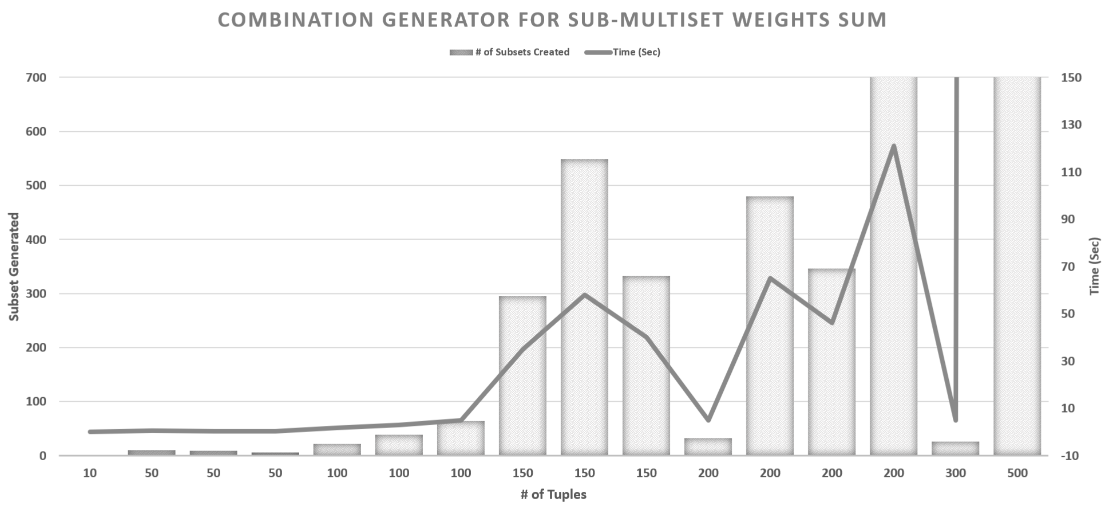


FIGURE 5.7: Subsets, Tuples and Time Relation

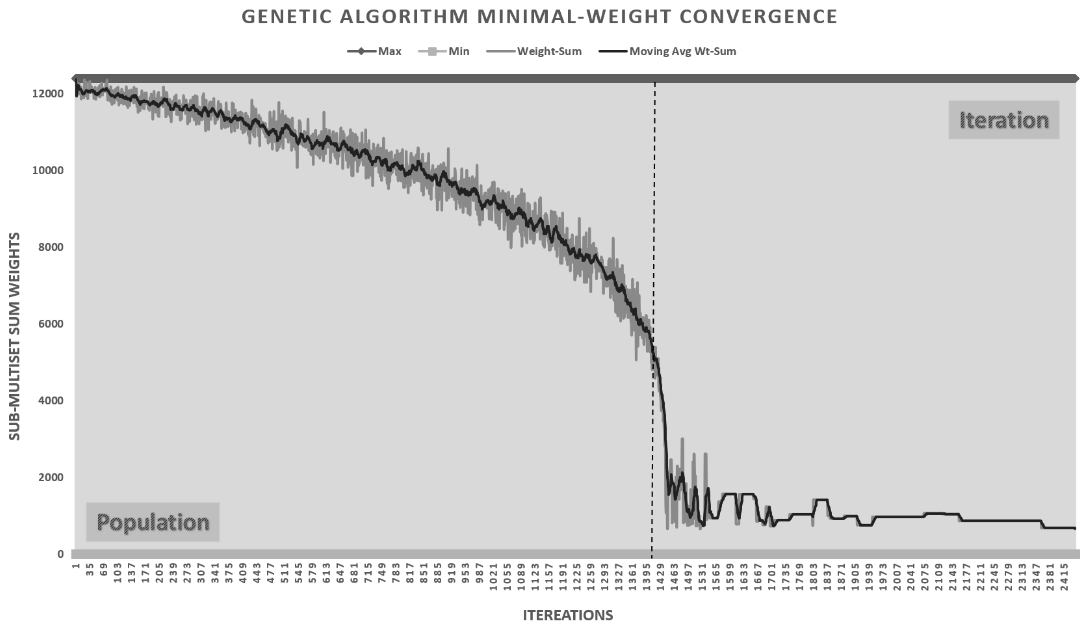


FIGURE 5.8: Genetic Algorithm Iterations for Minimal Weight Search

To solve the worst case scalability problem with the Greedy-DP based algorithm, we trade off guaranteed minimum weight and utilize sub-optimal genetic algorithms for better performance with respect to time and space. After following the steps described in the Algorithms-(4,5,6,7) we found better results. For example, with the designed dimensions and fact-tables of 1500 records (AdmDrug and Bills), which had 80 source of inconsistency ground atoms (with $2^{80} = 1,208,925,819,614,629,174,706,176$ Models), The population ran for 1 hour 45 minutes and then crossover-mutation ran for another 15 minutes, in total 2416 iterations in 2 hours resulted in exact minimum solution whereas same problem took more than 3 days to be solved by the fastest optimum Greedy-DP based solution. As we can see in the Figure 5.8, at first the population is initialized by taking all the items i.e. sum of all the weights (12401) of the 80 inconsistent predicates, then, converged slowly towards lower minimal weights. In the iteration phase, the result kept on improving with crossover and mutation, which sharply converged towards the desired solution (sum of weights 163). The practical benefit of using this algorithm is, a user can still run the iteration phase and at anytime when the algorithm stops, a minimum weight subset solution is generated up to that time. Deleting such candidates would enable consistency in the OMD model.

Chapter 6

Conclusion and Future Research

We studied the inconsistency detection and repair problem for the OMD model with respect to a set of dimensional constraints and rules. We showed, how rules and constraints complicate the repair generation process. We presented our technique to detect inconsistencies in the tuples or predicates of the dimensions, and proposed a weight-based repairing algorithm to restore consistency in OMD models. Given the multiple criteria that may be needed to generate an objective set of weights, we used the CRITIC method to compute a set of weights based on multiple criteria decision making, without user intervention. We also formally defined the minimal-weight repair semantics for the OMD model, and presented algorithms to identify the source of inconsistencies and to ground the generated predicates. We then developed a greedy and dynamic programming based minimal weight searching algorithm which outputs the predicates of minimal weight as final deletion candidates. The idea behind this algorithm was that, we did not need to generate all the models of a given theory to get minimal weights, if we could generate models in the ascending order of their sum of weights, that would be sufficient assuming that on an average, the expected set of predicates would be found at the mid-point of the search procedure. Our evaluation showed that, our assumption was correct about Min-Sum algorithm. This approach is faster than using brute force technique. We also implemented Genetic Algorithms for practical usage, which could be sub-optimal, however, our experiments demonstrated superior performance in terms time and space.

This research encompasses consistent query answering, ontologies, data cleaning, number theory and evolutionary algorithms. However, we see three avenues for future research:

1. We have proposed deletion based weighted repair, but update-based weighted repairs would be more interesting to explore.
2. From the perspective of mathematical logic and SAT solvers, to investigate if there is a way to generate models in ascending order of their summation of weights. This would eliminate costly satisfaction checking in the Min-Sum algorithm.
3. There is prior research on updating dimensions [65]–[71]. However, dimensions can be replaced with graphs or other types of ontologies in combination with tuples. OMD model has structural constraints, if those are relaxed, it can be a more

expressive ontology or more general like graphs. The techniques used here i.e. weights generation and finding their minimal subset using Min-Sum and GAs, are generic in nature. So, the application of such algorithms can be extended to diverse ontologies or graph databases.

Appendix A

System and Program

A1 System Configuration Details

Our system was hosted inside McMaster University (Computing and Software) department server. It was a virtualized workstation. Details of the server:

Virtualization: VMware
Operating System: Linux Mint 19.1
Kernel: Linux 4.15.0-54-generic
Architecture: x86-64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
CPU(s): 4
On-line CPU(s) list: 0-3
Thread(s) per core: 1
Core(s) per socket: 1
Socket(s): 4
NUMA node(s): 1
Vendor ID: GenuineIntel
CPU family: 6
Model: 79
Model name: Intel(R) Xeon(R) CPU E5-2687W v4 @ 3.00GHz
Stepping: 1
CPU MHz: 2996.529
BogoMIPS: 5993.05
Hypervisor vendor: VMware
Virtualization type: full
RAM: 32GB
L1d cache: 32K
L1i cache: 32K
L2 cache: 256K
L3 cache: 30720K
NUMA node0 CPU(s): 0-3
Programming Languages: R and Python

IDE: PyCharm and RStudio

A2 Program and Source Codes

All the programs were tested and source codes of all the experiments were uploaded in the following link: `https://github.com/ehmoni/Datalog-Weighted-Repairs`

Bibliography

[1] C. Batini and M. Scannapieco, “Data and information quality: Concepts”, *Methodologies and Techniques. Switzerland: Springer International Publishing*, 2016.

[2] W. W. Eckerson, “Data quality and the bottom line”, *TDWI Report, The Data Warehouse Institute*, pp. 1–32, 2002.

[3] T. C. Redman, “The impact of poor data quality on the typical enterprise”, *Communications of the ACM*, vol. 41, no. 2, pp. 79–83, 1998.

[4] R. Y. Wang and D. M. Strong, “Beyond accuracy: What data quality means to data consumers”, *Journal of management information systems*, vol. 12, no. 4, pp. 5–33, 1996.

[5] L. Bertossi and M. Milani, “Ontological multidimensional data models and contextual data quality”, *Journal of Data and Information Quality (JDIQ)*, vol. 9, no. 3, p. 14, 2018.

[6] L. Bertossi, F. Rizzolo, and L. Jiang, “Data quality is context dependent”, in *International Workshop on Business Intelligence for the Real-Time Enterprise*, Springer, 2010, pp. 52–67.

[7] G. Orsi and L. Tanca, “Context modelling and context-aware querying”, in *International Datalog 2.0 Workshop*, Springer, 2010, pp. 225–244.

[8] A. Cali, G. Gottlob, T. Lukasiewicz, B. Marnette, and A. Pieris, “Datalog+/-: A family of logical knowledge representation and query languages for new applications”, in *2010 25th Annual IEEE Symposium on Logic in Computer Science*, IEEE, 2010, pp. 228–242.

[9] L. Bertossi and J. Chomicki, “Query answering in inconsistent databases”, in *Logics for emerging applications of databases*, Springer, 2004, pp. 43–83.

[10] T. Eiter and G. Gottlob, “The complexity of logic-based abduction”, *Journal of the ACM (JACM)*, vol. 42, no. 1, pp. 3–42, 1995.

[11] E. Haque and F. Chiang, “Restoring consistency in ontological multidimensional data models via weighted repairs”, *Procedia Computer Science*, vol. 159, pp. 1085–1094, 2019.

[12] J. M. Nicolas, “First order logic formalization for functional, multivalued and mutual dependencies”, in *ACM SIGMOD international conference on management of data*, 1978, pp. 40–46.

[13] E. F. Codd, “Further normalization of the data base relational model”, *Database systems*, pp. 33–64, 1972.

[14] C. A. Zaniolo, “Analysis and design of relational schemata for database systems.”, *University of California, Los Angeles*, 1976.

[15] R. Fagin, “Multivalued dependencies and a new normal form for relational databases”, *ACM Transactions on Database Systems (TODS)*, vol. 2, no. 3, pp. 262–278, 1977.

[16] C. Delobel, “Semantics of relations and decomposition process in the relational data model”, *ACM Trans. Database Systems*, vol. 3, pp. 201–222, 1978.

[17] R. Fagin, “Horn clauses and database dependencies”, in *Proceedings of the twelfth annual ACM symposium on Theory of computing*, 1980, pp. 123–134.

[18] M. Yannakakis and C. H. Papadimitriou, “Algebraic dependencies”, *Journal of Computer and System Sciences*, vol. 25, no. 1, pp. 2–41, 1982.

[19] C. A. Hurtado, C. Gutierrez, and A. O. Mendelzon, “Capturing summarizability with integrity constraints in olap”, *ACM Transactions on Database Systems (TODS)*, vol. 30, no. 3, pp. 854–886, 2005.

[20] S. Ceri, G. Gottlob, and L. Tanca, *Logic programming and databases.* Springer Science & Business Media, 2012.

[21] C. A. Hurtado and A. O. Mendelzon, “Olap dimension constraints”, in *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2002, pp. 169–179.

[22] D. Diakoulaki, G. Mavrotas, and L. Papayannakis, “Determining objective weights in multiple criteria problems: The critic method”, *Computers & Operations Research*, vol. 22, no. 7, pp. 763–770, 1995.

[23] M. Zeleny, *Multiple criteria decision making Kyoto 1975.* Springer Science & Business Media, 2012, vol. 123.

[24] C. Yoe, “Trade-off analysis planning and procedures guidebook”, *US Army Corps of Engineers*, vol. 310, 2002.

[25] T. L. Saaty, “A scaling method for priorities in hierarchical structures”, *Journal of mathematical psychology*, vol. 15, no. 3, pp. 234–281, 1977.

[26] R. Janicki, “Finding consistent weights assignment with combined pairwise comparisons”, *International Journal of Management and Decision Making*, vol. 17, no. 3, pp. 322–347, 2018.

[27] C. E. Shannon, “A mathematical theory of communication”, *Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[28] V. P. Singh, “The entropy theory as a tool for modeling and decision-making in environmental and water resources”, *Journal of the Water Society of America*, vol. 1, 2000.

[29] A. Jahan, F. Mustapha, S. Sapuan, M. Y. Ismail, and M. Bahraminasab, “A framework for weighting of criteria in ranking stage of material selection process”, *The International Journal of Advanced Manufacturing Technology*, vol. 58, no. 1-4, pp. 411–420, 2012.

[30] R. Rao and B. Patel, "A subjective and objective integrated multiple attribute decision making method for material selection", *Materials & Design*, vol. 31, no. 10, pp. 4738–4747, 2010.

[31] R. Estrella, D. Cattrysse, and J. Van Orshoven, "Comparison of three ideal point-based multi-criteria decision methods for afforestation planning", *Forests*, vol. 5, no. 12, pp. 3222–3240, 2014.

[32] J. Holland, "Adaptation in natural and artificial systems: An introductory analysis with application to biology", *Control and artificial intelligence*, 1975.

[33] K. E. Kinnear Jr, "A perspective on the work in this book", *Advances in Genetic Programming*, pp. 3–19, 1994.

[34] M. Mitchell, "Genetic algorithms: An overview", *Complexity*, vol. 1, no. 1, pp. 31–39, 1995.

[35] J. H. Holland, "Genetic algorithms", *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.

[36] D. Beasley, D. R. Bull, and R. R. Martin, "An overview of genetic algorithms: Part 1, fundamentals", *University computing*, vol. 15, no. 2, pp. 56–69, 1993.

[37] I. F. Ilyas, X. Chu, *et al.*, "Trends in cleaning relational data: Consistency and deduplication", *Foundations and Trends® in Databases*, vol. 5, no. 4, pp. 281–393, 2015.

[38] F. Bry, "Query answering in information systems with integrity constraints", in *Working Conference on Integrity and Internal Control in Information Systems*, Springer, 1997, pp. 113–130.

[39] M. Arenas, L. Bertossi, and J. Chomicki, "Consistent query answers in inconsistent databases", *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 68–79, 1999.

[40] J. McCarthy, "Circumscription - a form of non-monotonic reasoning", *Artificial intelligence*, vol. 13, no. 1-2, pp. 27–39, 1980.

[41] R. Reiter, "A logic for default reasoning", *Artificial intelligence*, vol. 13, no. 1-2, pp. 81–132, 1980.

[42] D. Poole, "A logical framework for default reasoning", *Artificial intelligence*, vol. 36, no. 1, pp. 27–47, 1988.

[43] G. Brewka, "Preferred subtheories: An extended logical framework for default reasoning.", in *IJCAI*, vol. 89, 1989, pp. 1043–1048.

[44] D. Van Nieuwenborgh and D. Vermeir, "Preferred answer sets for ordered logic programs", in *European Workshop on Logics in Artificial Intelligence*, Springer, 2002, pp. 432–443.

[45] R. Fagin, J. D. Ullman, and M. Y. Vardi, "On the semantics of updates in databases", in *Proceedings of the 2nd ACM SIGACT-SIGMOD symposium on Principles of database systems*, 1983, pp. 352–365.

[46] P. Andritsos, A. Fuxman, and R. J. Miller, “Clean answers over dirty databases: A probabilistic approach”, in *22nd International Conference on Data Engineering (ICDE’06)*, IEEE, 2006, pp. 30–30.

[47] S. Greco, C. Sirangelo, I. Trubitsyna, and E. Zumpano, “Preferred repairs for inconsistent databases”, in *7th International Database Engineering and Applications Symposium*, IEEE, 2003, pp. 202–211.

[48] M. V. Martinez, F. Parisi, A. Pugliese, G. I. Simari, and V. Subrahmanian, “Inconsistency management policies.”, in *KR*, 2008, pp. 367–377.

[49] F. N. Afrati and P. G. Kolaitis, “Repair checking in inconsistent databases: Algorithms and complexity”, in *12th International Conference on Database Theory*, ACM, 2009, pp. 31–41.

[50] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi, “A cost-based model and effective heuristic for repairing constraints by value modification”, in *SIGMOD international conference on Management of data*, ACM, 2005, pp. 143–154.

[51] W. Fan, P. Bohannon, F. Geerts, X. Jia, and A. Kementsiets, “Conditional functional dependencies for data cleaning”, in *IEEE 23rd International Conference on Data Engineering*, 2007.

[52] X. Chu, I. F. Ilyas, and P. Papotti, “Holistic data cleaning: Putting violations into context”, in *29th International Conference on Data Engineering (ICDE)*, IEEE, 2013, pp. 458–469.

[53] T. Easton, K. Hooker, and E. K. Lee, “Facets of the independent set polytope”, *Mathematical programming*, vol. 98, no. 1-3, pp. 177–199, 2003.

[54] N. Roos, “A logic for reasoning with inconsistent knowledge”, *Artificial Intelligence*, vol. 57, no. 1, pp. 69–103, 1992.

[55] P. Barceló, L. Bertossi, and L. Bravo, “Characterizing and computing semantically correct answers from databases with annotated logic and answer sets”, in *International Workshop on Semantics in Databases*, Springer, 2001, pp. 7–33.

[56] D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi, and D. F. Savo, “Inconsistency-tolerant semantics for description logics”, in *International Conference on Web Reasoning and Rule Systems*, Springer, 2010, pp. 103–117.

[57] A. Arioua, N. Tamani, and M. Croitoru, “Query answering explanation in inconsistent datalog+/- knowledge bases”, in *International Conference on Database and Expert Systems Applications*, Springer, 2015, pp. 203–219.

[58] T. Lukasiewicz, M. V. Martinez, and G. I. Simari, “Inconsistency handling in datalog+/- ontologies”, in *20th European Conference on Artificial Intelligence*, IOS Press, 2012, pp. 558–563.

[59] M. Bienvenu, C. Bourgaux, and F. Goasdoué, “Querying inconsistent description logic knowledge bases under preferred repair semantics”, in *28th AAAI Conference on Artificial Intelligence*, 2014.

[60] J. Du, G. Qi, and Y. D. Shen, "Weight-based consistent query answering over inconsistent shiq knowledge bases", *Knowledge and Information Systems*, vol. 34, no. 2, pp. 335–371, 2013.

[61] S. Staworko and J. Chomicki, "Consistent query answers in the presence of universal constraints", *Information Systems*, vol. 35, no. 1, pp. 1–22, 2010.

[62] N. Prokoshyna, J. Szlichta, F. Chiang, R. J. Miller, and D. Srivastava, "Combining quantitative and logical data cleaning", *VLDB Endowment*, vol. 9, no. 4, pp. 300–311, 2015.

[63] A. V. Aho, C. Beeri, and J. D. Ullman, "The theory of joins in relational databases", *ACM Transactions on Database Systems (TODS)*, vol. 4, no. 3, pp. 297–314, 1979.

[64] F. Geerts, G. Mecca, P. Papotti, and D. Santoro, "The llunatic data-cleaning framework", *VLDB Endowment*, vol. 6, no. 9, pp. 625–636, 2013.

[65] M. Caniupan, L. Bravo, and C. Hurtado, "Logic programs for repairing inconsistent dimensions in data warehouses", *Journal submission*, 2010.

[66] M. Caniupan, L. Bravo, and C. A. Hurtado, "Repairing inconsistent dimensions in data warehouses", *Data & Knowledge Engineering*, vol. 79, pp. 17–39, 2012.

[67] M. Caniupan, A. Vaisman, and R. Arredondo, "Efficient repair of dimension hierarchies under inconsistent reclassification", *Data & Knowledge Engineering*, vol. 95, pp. 1–22, 2015.

[68] J. Ramirez, L. Bravo, and M. Caniupán, "Extended dimensions for cleaning and querying inconsistent data warehouses", in *16th international workshop on Data warehousing and OLAP*, ACM, 2013, pp. 39–46.

[69] S. Ariyan and L. Bertossi, "A multidimensional data model with subcategories for flexibly capturing summarizability", in *25th International Conference on Scientific and Statistical Database Management*, ACM, 2013, p. 6.

[70] C. A. Hurtado, A. O. Mendelzon, and A. A. Vaisman, "Updating olap dimensions", in *2nd ACM international workshop on Data warehousing and OLAP*, 1999, pp. 60–66.

[71] H.-J. Lenz and A. Shoshani, "Summarizability in olap and statistical data bases", in *9th International Conference on Scientific and Statistical Database Management (Cat. No. 97TB100150)*, IEEE, 1997, pp. 132–143.

[72] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of databases.* Addison-Wesley Reading, 1995, vol. 8.

[73] D. M. Gabbay and M. J. Sergot, "Negation as inconsistency. i", *The Journal of Logic Programming*, vol. 3, no. 1, pp. 1–35, 1986.

[74] R. Bellman, "On the theory of dynamic programming", *National Academy of Sciences of the United States of America*, vol. 38, no. 8, p. 716, 1952.

[75] P. G. Kolaitis, J. Panttaja, and W.-C. Tan, "The complexity of data exchange", in *25th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2006, pp. 30–39.

[76] W. Shen, B. Xu, and J. p. Huang, “An improved genetic algorithm for 0-1 knapsack problems”, in *2nd International Conference on Networking and Distributed Computing*, IEEE, 2011, pp. 32–35.

[77] P. Carbonnelle. (2018). Pydatalog, [Online]. Available: `https://sites.google.com/site/pydatalog/`. (accessed: 05.01.2020).