International Journal of Principles and Applications of Information Science and Technology
December 2007, Vol.1, No.1

# Correcting Stored RFID Data with Non-Monotonic Reasoning

Peter Darcy [1], Bela Stantic [1], Roozbeh Derakhshan [2]

[1] Institute for Integrated and Intelligent Systems
Griffith University, Brisbane, Australia
[2] ETH Zurich, Switzerland
{P.Darcy, B.Stantic}@griffith.edu.au, droozbeh@inf.ethz.ch

**Abstract.** Radio Frequency Identification (RFID) technology has been held back from wide-scale commercial deployment for years due to the high level of errors in data. Despite the process of filtering the data at the edge (where RFID tags are being scanned), a significant portion of incorrect data is still inserted into the database. This incorrect data can cause inconsistency in database and can significantly influence a business logic. The errors that cause these inconsistencies include duplicate read, miss read and redundant read. While some work presented in the literature addressed the issue of cleaning RFID data, existing methods cannot solve problems if there is more that one solution or the solutions are coupled ambiguously. In this study, we present a Non-Monotonic Reasoning method that utilises two techniques to clean the RFID data stored in the database, in order to enhance its accuracy. Experimental results show, that *Non-Monotonic Reasoning* can be efficiently used for cleaning RFID data, and can also obtain a higher cleaning rate when compared with the two traditional cleaning techniques alone.

## 1 Introduction

For the past few years, Radio Frequency Identification (RFID) technology has been developed to the point of universal scale deployment. When analysing the quality of a scan, it may be observed that there is a need for improvement of the quality of recorded data. Researchers and industries have attempted to compensate the inaccuracies for RFID reading by providing users with smoothing filters for the RFID data before it is stored into the database and also, programming languages to write rules in order to clean the data when needed. At the moment, state of the art technology has not yet enabled the data to be transformed into perfectly accurate form and, as a result, there is a need to seek out other ways to enhance the performance of these methods used.

Several research studies such as [11] that use a deferred cleaning method have been found to be concerned with devising an efficient algorithm for cleaning data and thus focus more upon the time expenses and ease of use by an end user rather than the accuracy, whereas others such as [8] have been found to relate more to the use of probability to determine a high level set of data rather than low level raw data like the data used in this study. The majority of research discovered as, for example, [1] and [7] cannot be considered comparable due to the fact that they are concerned with filtering data at the edge rather than cleaning the data at a later time after the data input has been made. Additionally, existing methods for cleaning RFID data cannot cope with situations when more than one solution exists or when solutions are ambiguous.

In this study, we presents two novel techniques which have been combined with a Non-Monotonic Engine using Plausible Logic to find the correct course of action to clean dirty sets of data. To this end we have devised two algorithms, the *Later Instance* algorithm and the *Item Hierarchy* algorithm to demonstrate how the results may be compared to a simulated version of other rules. The *Later Instance* algorithm has been based on *Missing Rule* proposed in [11] whereas the *Item Hierarchy* algorithm has resembled what we believe a data mining algorithm would set out to accomplish in this work. After analysing the results, it has been discovered that this approach has enhanced the cleaning when compared with the utilisation of one of the techniques with no Non-Monotonic Reasoning.

The main contribution gained from this research includes the discovery that Non-Monotonic Reasoning may be used to determine the correct course of action when faced with ambiguity in cleaning.

## 2 Background

RFID is an efficient mechanism for tracking certain objects from destinations as it allows items to be located in real time for a low cost with relative accuracy. An RFID tag is comprised of a small chip, antenna and power source on more expensive tags [4]. The purpose of these tags is to emit a unique identifier to be picked up by a scanner. This provides a low cost solution (estimated at 5 cents a tag) and a work load for several items to be gathered and counted as well as tracked through destinations [4].

Industry, in particular the retail industry, may benefit in several ways by using the RFID. First, if everything is automated with the use of RFID, there is no reason to pay for a check-out operator swiping each item past a barcode reader. Secondly, there will be a highly accurate depiction of stock actually held when unforeseen circumstances become apparent. Also, the utilisation of RFID will ensure an automated system running constantly to monitor stock levels and will result in there being no longer instances where stock is hiding in a back room when it should be displayed on the shelf. Furthermore, retailers may paint a detailed picture of exactly where their stock is within the Supply Chain [5]. Due to these reasons, businesses may increase profits, reduce manual labor and provide a faster service for customers. RFID Technologies, however, suffer from different errors in data. Some of these include:

- Duplicate Reads: When a tag that is identical to another is entered into the database
- Missed Reads: Where a tag has been omitted from the read for various physical problems

- Redundant Reader: When two or more readers overlap the same space and record the same observations

## 2.1 Non-Monotonic Reasoning

A way to address the uncertainty accompanied with RFID technology is to develop a system that harnesses non-monotonic reasoning. This reasoning stems from the exact reverse of conventional logic. In conventional logic, there is a solution based on a set of information and assumptions. As more assumptions are made, in order to comprehend these, more possible solutions are revised and developed [6].

**Plausible Logic** The specific logic that will be applied in this study will be a logic known as Plausible Logic (PL) [2]. PL refers to a logic that has been built from the existing logic known as Defeasible Logic [10]. There are three different attributes associated with Defeasible Logic:

- Strict Rules: Rules that are known facts
- Defeasible Rules: Rules that are usually correct
- Undercutting Defeaters: Anything that overrides the Defeasible Logic is stated here.

Plausible Logic is built upon the concept of defeasible logic but, it is important also to kept in mind that eventually this logic will be implemented on a computer [2]. For an example of Plausible Logic, the following argument could be considered: Birds usually fly. There is an exception to this rule as certain birds such as quails do not fly. In this situation there is no clear superior answer unless the information of what specie of bird is provided [3].

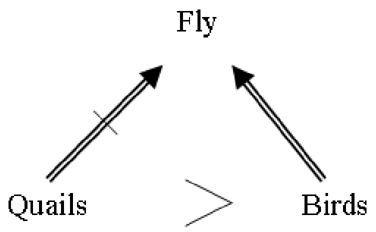

**Fig. 1.** A diagram logical map showing that birds usually fly however there is an exception for certain species of birds.

There are several proof algorithms which may be used as a setting to determine the strength of the reasoning. The algorithms used to reason a situation's validity are explained in detail:

- $\mu$**:** Only correct information is used when proving the theory. Anything against proving the item must be nulled out. There must be strict and plausible rules that lead to the item. The item's conjunction of its antecedents must be proved.
- $\Pi$**:** Ambiguity is increased in this method. Specifically, the antecedent of a statement is examined to prove the negative conjunction.
- $\beta$**:** As opposed to the $\Pi$ algorithm, ambiguity is blocked from the decision. Specifically, the proofs of the antecedent of a statement is examined to see if it fails in a number of steps that will not get into an infinite loop.
- $\alpha$**:** In this algorithm, both of the $\Pi$ and $\beta$ algorithms must be proven for any proof to be returned.
- $\delta$**:** In this algorithm, either $\Pi$ or $\beta$ algorithms must be proved to be correct for any proof to be returned.

## 3 Existing Techniques

Currently RFID technologies deal with cleaning the data sets after the items are stored in a database by developing a programming language for users to input their own algorithms to clean up data. A typical algorithm for cleaning missed reads has been shown in Algorithm 1 and Algorithm 2 [11]. As seen in Algorithm 2, an item "A" is only kept if two items including itself at one time was at the same location in under 5 minutes.

**Algorithm 1** Missing Rule - Case Near By

```
if A == Pallet AND (X != Pallet AND A.loc = X.loc AND A.Time - X.Time < 5
Minutes) OR (Y != Pallet AND A.loc = Y.loc AND Y.Time - A.Time < 5 Minutes)
then
    A.casenearby = 1
end if
end
```

**Algorithm 2** Missing Rule - Keeping A

```
if A != Pallet OR (A.casenearby = 0 AND B.casenearby = 1) then
    KEEP A
end if
end
```

## 4 Cleaning RFID Data with Non-Monotonic Reasoning

Considering the examples of off-line cleaning techniques such as that for missing data in Algorithms 1 and 2, it is reasonable to assume that there is a disadvantage present for there is little tolerance when faced with ambiguous situations. To this end, it has been sought out to determine if there is a method that may take all known data, conduct an analysis and perform the correct course of action for varying situations. In this work we present the use of the Plausible Logic engine which uses Non-Monotonic principles to obtain the correct response to any given situation.

### 4.1 Scenario

In a Supply Chain, Items $I$ will be inserted into Cases, $C$. The Cases then will be put on Pallets, $P$, and the Pallets will be loaded onto a Truck, $T$. It is important to know the whereabouts of a particular item for the sake of informing customers about the status of his/her order. In particular, is it important to present clean data to the customer who may wish to view the status of his/her order online and to eliminate any ambiguity that could be present when the locations are not known at every point.

For above reasons, it is important to track the locations of the articles even if they have already reached another destination. Usually, sensors will miss some of Electronic Product Code (EPC) Tags which cause errors and possible ambiguity. Cleaning algorithms along with non-monotonic reasoning need to be used to enhance the integrity of the data.

### 4.2 The Process of the Scenario

In order to demonstrate the process by which Non-Monotonic Reasoning may be used to clean up some ambiguity, the following scenario is proposed. A truck filled with several RFID tagged items is travelling enroute towards a destination and passes under several RFID readers along the way.

As shown in this example, there are several items which are shown at each location. As each tag is read at the different locations, it may be assumed that some tags will not be read due to various circumstances. An analysis will then pass findings onto the Logical Engine to have a response returned which determine the importance of the insertion of the data. Figure 2 presents how this idea would be implemented and how the program would interact with its various components.

### 4.3 Assumptions

There are a number of assumptions which need to be declared to follow through with this scenario. The first assumption is that the hierarchy structure of the Pallets, Cases and Items are known throughout the process. Next, the Pallets, Cases and Items have to be assumed that they can go missing at anytime. Finally it may be assumed that, if all the container's subset Items/Cases are present, then the container must also be found. This does not have to be physically true

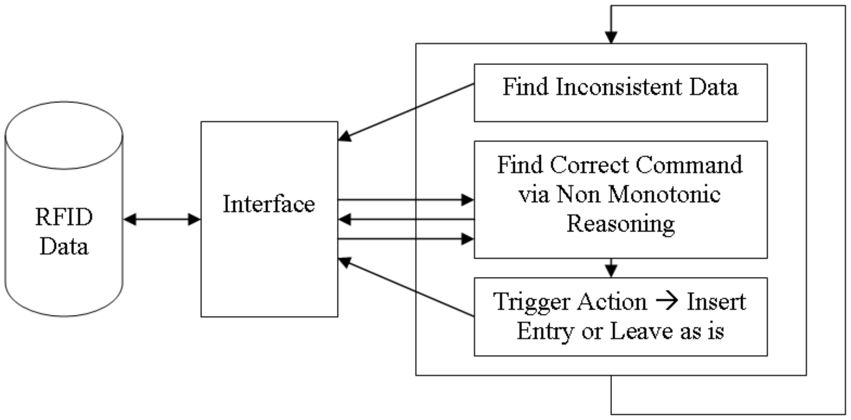
RFID Data
Interface
Find Inconsistent Data
Find Correct Command via Non Monotonic Reasoning
Trigger Action → Insert Entry or Leave as is

**Fig. 2.** A diagram demonstrating how the program will cope with a contradicting situation. The Program first looks for inconsistent data then finds the correct course of action to act upon this data and finally, usees a trigger to insert or delete the data.

as having all the subset Items/Cases present is sufficient reason to insert the container as it should just be an empty shell housing it. Therefore, the reading of the Pallet's/Case's tag is not as significant as the very presence of the Item's/Case's tag. Further evidence would involve a Pallet/Case not falling off/being stolen without the rest of its subset Cases/Items missing too. Additionally, it may be assumed that if being stolen, the thief would not only take the tags, but rather the whole package.

The assessments made in the experimentation coupled with the assumptions listed above have been chosen to simulate the realistic nature of an RFID enabled supply chain.

## 4.4 Contradictions within the Scenario

Within the chosen scenario, there are several instances when the Non-Monotonic Reasoning Engine must be used. These instances are based on the finding of subset relatives and the presence of later instances of the tag in question. For example, if it is found that there are subset relatives on the level below which are present, it may be assumed that the Tag should be inserted. However, if it is found that not all the relatives are found under that subset Tag several levels through the hierarchy, it should not be inserted. If there is a later instance of this value, then it may be ascertained that this is correct despite the lack of evidence of a complete subset and, the assumption that, requires insertion.

### 4.5 Database Setup

The database structure has its entity relationships setup resembling the Data Model for RFID Applications (DMRA) schema [9]. Out of a whole model, only the entities "Object", "Reader" and "Observation" have been chosen to be used. This is due to the fact that the "Observation" entity illustrates the raw data that later is transformed into meaningful data. The "Object" and "Reader" tables also need to be populated as the "Observation" entity is a derived and weak entity relying on both these tables for primary keys.

### 4.6 Logical Engine Set up

The Logical Engine used in this study is based upon logic which is implemented on the Sony AIBO Robots in the 2005 RoboCup [3]. As discussed previously, the Non-Monotonic Reasoning logical engine comprises two main components. First is the input being passed to it, and secondly, the logical rules that must be followed to derive a conclusion. To initialise the engine, the code is written in Decisive Programming Language (DPL) which is used to illustrate the logic in a recognisable set of commands [3].

For the Logical Engine, several rules have been defined to allow the engine to determine which course of action is correct. The Non-Monotonic rules which have been implemented is represented graphically in Figure 3. The rules which outweigh others are defined as an *arc* connecting the more important anticlockwise rule to the less important clockwise rule.

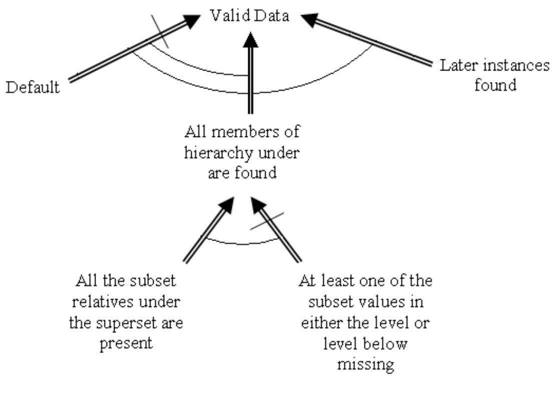
Valid Data
Default
Later instances found
All members of hierarchy under are found
All the subset relatives under the superset are present
At least one of the subset values in either the level or level below missing

**Fig. 3.** A Graphical Representation of how the rules are known to act and interact with each other in the Non-Monotonic Reasoning Engine.

The following rules are the DPL equivalent of the Non-Monotonic Reasoning engine's logic illustrated in Figure 3:

*type (x)* $\leftarrow$ *suspicious data*

*DefaultRule:* $\{\} \Rightarrow$ *-validData.*

*SubsetRule: subsetValuesFound(x)* $\Rightarrow$ *allSubsetFound.*

*NonSubsetRule: oneSubsetMissing(x)* $\Rightarrow$ *-allSubsetFound.*

NonSubsetRule > SubsetRule:

*AllSubsetRule: allSubsetFound(x)* $\Rightarrow$ *validData.* AllSubsetRule > DefaultRule.

*LaterRule: laterInstancesFound(x)* $\Rightarrow$ *validData.* LaterRule > AllSubsetRule.

Prior to the programs execution, $x$ is declared to have *suspicious data* contained in it that needs to be analysed. The *DefaultRule* states that, if there is nothing passed into it via $x$, the missed value should be invalid. The *SubsetRule* states that, if the subset values are found in $x$, must be indicated. The *NonSubsetRule* states that, if there is only one subset value missing in any part of the hierarchy of $x$, then *AllSubsetFound* must be stated as false. The *NonSubsetRule* outweighs the *SubsetRule*. The *AllSubsetRule* states that, if *AllSubsetFound* is correct, the missed value must be declared valid and this rule overrides the *DefaultRule*. The *LaterRule* states that if there is a *LaterInstanceFound* is true in $x$ then declare the missed value as valid. The *LaterRule* defeats the *AllSubsetRule* which, in turn, defeats the *DefaultRule*.

## 5 Algorithms

The Algorithms proposed in this work consist of three main reasoning systems. The first checks the presence of a later instance of the missed value; the second checks to see if all the subset relations of the missed value are present; and the third combines these algorithms and employs Non-Monotonic Reasoning to find the superior algorithm in any situation.

### 5.1 Later Instance Reasoning Algorithm

The *Later Instance Reasoning* algorithm draws its fundamental principles from the evidence that a later instance of a value [9] exists and, therefore, it may be assumed that every missed instance prior to this may be inserted. This reasoning architecture may be described in pseudo code in Algorithm 3.

**Algorithm 3** Later Instances approach to correct the data

```
for All the Missing Data do
   Check if there is a later instance
end for
if There is a Later Instance then
   Insert value into Database
end if
end
```

### 5.2 Item Hierarchy Reasoning Algorithm

The *Item Hierarchy Reasoning* algorithm was inspired by various data mining techniques, which states, if all subset relations for any superset exist then the superset may also exists. This reasoning architecture is described in pseudo code in Algorithm 4.

**Algorithm 4** Item Hierarchy approach to correct the data

```
for All the Missing Data do
   Check if items are present one level under missed value in hierarchy
   Check if all items are present under missed value in hierarchy
end for
if Subset members are present then
   Insert value into Database
end if
end
```

### 5.3 Non-Monotonic Reasoning Algorithm

The *Non-Monotonic Reasoning* will reflect the reasoning of *Later Instance* and *Item Hierarchy* algorithms but it will also have the ability to distinguish which one of the two has precedence using the engine. The algorithm essentially searches for evidence of the suspicious data being valid. If there is a lack of evidence or no complete subset can be found then the algorithm will leave the database in its current state, however, evidence of a later instance or a complete subset will allow the algorithm to confidently insert the missed value. The process to which the *Non-Monotonic Reasoning* engine reaches a decision is illustrated in Figure 3.To see the DPL logic code used in this Figure, please see Section 4.6.

## 6 Experimental Evaluation

To show the relevance of our approaches to clean RFID data, we conducted extensive experiments. The experiments were conducted for each of the reasoning

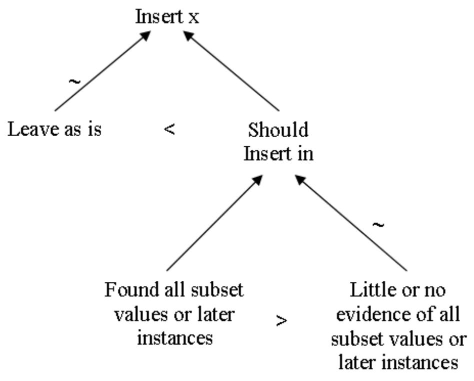
Insert x
~
Leave as is
<
Should Insert in
~
Found all subset values or later instances
>
Little or no evidence of all subset values or later instances

**Fig. 4.** A graphical representation what the program will be looking at and checking to get enough confidence to insert a value back into the database.

algorithms on each data set. More specifically, *Later Instance Algorithm*, *Item Hierarchy Algorithm* and *Non-Monotonic Algorithm* are tested on each of the data sets created. Furthermore, within the *Non-Monotonic Reasoning* algorithm, each of the five proof formulas ($\mu$, $\alpha$, $\beta$, $\pi$ and $\delta$) are tested.

## 6.1 System Architecture

The system implemented in this experiment is utilised in Oracle RDBMS using PL/SQL environment and calling external procedure written in C. As seen in Figure 2, the Oracle code searches for data that might be incorrect based on a mining algorithm, whenever there is an ambiguous situation it calls the *C Plausible Logic Engine* to determine if the data needs to be inserted back into the database or not. C *Plausible Login Engine* returns True or False to Oracle code which fulfills appropriate action by inserting the data in or alternatively leaves the database state as it is. The Oracle System communicates to the C logical engine by calling a command and passes certain numbers of parameters. These parameters are discussed in detail in Section 4.6.

## 6.2 Environment

he environment being used in this paper consists of using the general Data Model for RFID Applications (DMRA). Specifically, this consists of using the tables of OBSERVATION where the EPC of an object is recoded, the location of where the object was scanned is also stored and the time of starting the scanning of the item and ending of scanning the item is noted as well [9] (For more information about the Database Setup please see Section 4.5). The computer being used to run these experiments is a Microsoft Windows XP Professional Version 2002 with Service Pack 2 system with Intel(R) Pentium(R) 4 CPU 2.79GHz and has 1.00 GB of RAM. The Database was written and compiled in Oracle SQL*Plus 8.0

while the Logical Engine was written in C and implemented using the Cygwin Version 1.5.24-2.

### 6.3 Data Sets

The data set being inserted into the database consists of generated test data based on the chosen scenario. This consists of three different readers which have recorded fifteen different EPC tags at one given moment. Additionally twelve different tags have been deleted from the complete data set to align properly with the scenario. Along side this test data, four other sets of data have been created with randomly chosen tags to be removed to attempt to simulate how missed readings would affect a database. A small example of a data set is shown in Table 1.

**Table 1.** Sample data set used in experimentation (The tag 9994534562440003 is missing at reader 1000000000000001)

| Reader | Tag | Timestamp |
|---|---|---|
| 1000000000000001 | 9994534562440001 | 11:05:23-29-SEP-07 |
| 1000000000000001 | 9994534562440002 | 11:05:23-29-SEP-07 |
| 1000000000000001 | 9994534562440004 | 11:05:23-29-SEP-07 |
| ... | ... | ... |

## 7 Results and Analysis

The results of the testing of the five data sets and three different reasoning algorithms are shown in Table 2. The numbers inside the table represent how many inserts took place for that cross-reference. The abbreviations used in this table has the following meaning:

- L.I.R. = Later Instance Reasoning.
- I.H.R. = Item Hierarchy Reasoning.
- $\mu$, $\alpha$, $\pi$, $\beta$, $\delta$ = Non-Monotonic Reasoning Proofs

When analysing the results provided in Table 2 it may be seen that when using the algorithms which incorporate *Non-Monotonic Reasoning*, achieved greater precision results than the *Later Instance* and *Item Hierarchy Algorithms.* Figure 5 displays in a graphical representation of the results extrapolated from Table 2. As observer, the *Later Instance Algorithm*, the *Item Hierarchy Algorithm* and the *Non-Monotonic Reasoning Algorithm* are all compared with each other to see how many inserts have been made. The expected number of inserts for the sample and the first random data sets is 10, for the second and third

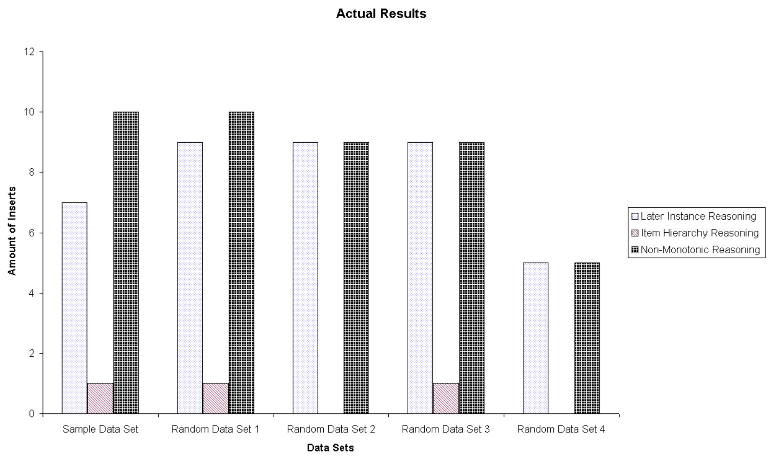

**Fig. 5.** A Bar Graph showing the Results of Table 2 of both algorithms from this experiment with Percentage of Amount of Inserts on the y axis vs. the Data Sets on the x axis.

**Table 2.** A table illustrating the results of number of inserts

| Data Sets | L.I.R. | I.H.R. | $\mu$ | $\alpha$ | $\pi$ | $\beta$ | $\delta$ |
|---|---|---|---|---|---|---|---|
| Sample | 7 | 1 | 0 | 10 | 10 | 10 | 10 |
| Random 1 | 9 | 1 | 0 | 10 | 10 | 10 | 10 |
| Random 2 | 9 | 0 | 0 | 9 | 9 | 9 | 9 |
| Random 3 | 9 | 1 | 0 | 9 | 9 | 9 | 9 |
| Random 4 | 5 | 0 | 0 | 5 | 5 | 5 | 5 |

random data sets the expected amount of inserts to be obtained is 9 and the fourth random data set has an expected number of inserts of 5.

The *Non-Monotonic Reasoning* algorithm performed the most successfully in both the Sample data set and Random data set 1. The *Later Instance* algorithm tied with the *Non-Monotonic Reasoning* algorithm for amount of inserts in Random data sets 2 to 4. Finally, the *Item Hierarchy Reasoning* algorithm performed the least successfully one row of data in the Sample data set and Random data sets 1 and 3 being inserted. The results for the *Non-Monotonic Reasoning* algorithm may be explained by the algorithm having both the advantages of the *Later Instance* and *Item Hierarchy* algorithms. The results for the *Later Instances* may be explained by the fact that there is very little chance in a Random data set where the *Item Hierarchy* algorithm may be used. Therefore, the *Later Instance* receives the same amount of *Non-Monotonic Reasoning* inserts for the Random data sets 2, 3 and 4.

## 8 Conclusion

In this study, we addressed the issue of cleaning RFID data stored in the database. We provided an experiment that evaluated the effects of the *Non-Monotonic Reasoning*, *Later Instance Reasoning* and *Item Hierarchy* algorithms on one sample data set and five randomly generated data sets with missing instances deleted. The reason for this was to illustrates that the *Non-Monotonic Reasoning* may be used to combine two novel algorithms and to provide more accurate results than either of the algorithms could achieve by themselves. The results achieved through the various data sets have supported this argument.

As a future work, the employment of different techniques to search for a missed value, different methods to compare/enhance the *Non-Monotonic Engine* such as *Neural Networks* and using the successor of *Plausible Logic* known as *Clausal Defeasible Logic* (CDL) have been displayed as proof that intriguing results may be derived, which paves the way for future research ideas.

## References

1. Y. Bai, F. Wang, and P. Liu. Efficiently Filtering RFID Data Streams. In *CleanDB*, 2006.
2. D. Billington. The proof algorithms of plausible logic form a hierarchy. In *Proceedings of the 18th Australian Joint Conference on Artificial Intelligence, Sydney, Australia, 5-9 December 2005. Lecture Notes in Artificial Intelligence vol. 3809 ISBN 92-990021-0-X, Springer*, pages 796–799, 2005.
3. D. Billington, V. Estivill-Castro, R. Hexel, and A. Rock. Non-monotonic reasoning for localisation in robocup. In *Proceedings of the 2005 Australasian Conference on Robotics and Automation ISBN: 0-9587583-7-9*, 2005.
4. S. S. Chawathe, V. Krishnamurthy, S. Ramachandran, and S. E. Sarma. Managing RFID Data. In *VLDB*, pages 1189–1195, 2004.
5. R. Derakhshan, M. E. Orlowska, and X. Li. RFID Data Management: Challenges and Opportunities. In *RFID 2007*, pages 175 – 182, 2007.
6. G. Gottlob. Complexity results for nonmonotonic logics. *Journal of Logic and Computation*, 2(3):379–412, 1992.
7. S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom. A pipelined framework for online cleaning of sensor data streams. In *ICDE*, page 140, 2006.
8. N. Khoussainova, M. Balazinska, and D. Suciu. Probabilistic rfid data management. *UW CSE Technical Report UW-CSE-07-03-01*, March 2007.
9. S. Liu, F. Wang, and P. Liu. A Temporal RFID Data Model for Querying Physical Objects. Technical Report TR-88 , TimeCenter, 2007.
10. D. Nute. Defeasible reasoning. In *In Proceedings of the 20th Hawaii International Conference on System Science*, pages 407–477, 1987.
11. J. Rao, S. Doraiswamy, H. Thakkar, and L. S. Colby. A Deferred Cleansing Method for RFID Data Analytics. In *VLDB*, pages 175–186, 2006.