Satisfying K-Anonymity:

New Algorithm and Empirical Evaluation

Romeo Issa

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of

Master of Computer Science

Under the auspices of the Ottawa-Carleton Institute for Computer Science

Carleton University

Ottawa, Ontario, Canada
January 2009

Abstract

Nowadays, clinical institutions are increasingly asked to make their raw data electronically available for research purposes. However, the same laws that prevent casual disclosure of such data have also made it difficult for researchers to access the information they need to conduct critical research. Therefore, several algorithms were developed with the purpose of making that information anonymous, hence readily available for researchers.

In this thesis, we present the results of an empirical evaluation of algorithms that aim to achieve k-anonymity under global recoding and hierarchical generalization, namely, Datafly and Samarati's algorithms. We conclude that on average the latter produces better results, but neither produces an optimal solution. Next, we propose a new method to efficiently find the optimal solution, and we illustrate some programming optimizations. Finally, we compare our approach from an efficiency perspective to Incognito, an efficient algorithm that finds the set of all possible solutions.

Acknowledgment

I would like to express my deep and sincere gratitude to my co-supervisors, Prof. Khaled El Emam, Prof. Daniel Amyot, and Prof. Jean-Pierre Corriveau. This thesis would not have been possible without their continuous help, guidance and support.

This research was conducted at the Electronic Health Information Laboratory as part of the Collaborative Health Research Project on *Performance Management at the Point of Care: Secure Data Delivery to Drive Clinical Decision Making Processes for Hospital Quality Control*, funded by the Canadian Institutes of Health Research and the Natural Sciences and Engineering Research Council of Canada. The work was done mainly in collaboration with Dr. Khaled El Emam, with the much appreciated support of Dr. Fida Kamal Dankar. Furthermore, many thanks go to my co-workers and the people who provided the private data sets used in the experiments, namely: Elizabeth Jonker, Elise Cogo, Sadrul Chowdhury, Regis Vaillancourt, Tyson Roffey, Jim Bottomley and Mark Walker.

My sincere thanks also go to the official referees, Dr. Anil Somayaji and Dr. Carlisle Adams, for their detailed review and constructive criticism.

Above all, I want to thank Mr. Peter and Mrs. Siham Irani, and their family, who welcomed me in their home during my stay in Canada so far, and who also provided the kind of unconditional love and support a person would only expect from his own parents and family.

Table of Contents

List of Tables

List of Figures

List of Acronyms

Acronym	Definition
DM	Discernability Metric
DOB	Date of Birth
EC	Equivalence classes
FSA	Forward Sortation Area
GH	Generalization Hierarchy
HIPAA	Health Insurance Portability and Accountability Act
IL	Information Loss
k	The anonymization level
MaxSup	Maximum Suppression allowed (or Suppression limit)
NE	Non-Uniform Entropy
OSC	Optimal Solution Candidates
PHIPA	Personal Health Information Protection Act
PT	Private Table
QI	Quasi-Identifier
SDC	Statistical Disclosure Control
SSN	Social Security Number
UI	Unique Items

Chapter 1. Introduction

1.1. Motivation

Nowadays, clinical institutions are increasingly asked to make their raw, non-aggregated data (also called microdata), electronically available for research purposes. However, since such data may contain private personal information as in the case of medical records, the identity of the entities involved must remain confidential.

A telephone poll has been conducted in the United States in which 88% of the respondents replied that to the best of their knowledge, no medical information about themselves had ever been disclosed without their permission. In a second question, 87% said laws should prohibit organizations from giving out medical information without obtaining the patient's permission. Thus, the public would prefer that only employees and directly-involved people have access to their records and that these people be bound by strict ethical and legal standards that prohibit further disclosure [34].

Nowadays, the disclosure of health information is strictly regulated in many jurisdictions, and institutions are often legally required to apply privacy-enhancing transformations to health data prior to their disclosure to researchers. For example, the Health Insurance Portability and Accountability Act (HIPAA) [19] in the United States, and the Personal Health Information Protection Act (PHIPA) [30] in Canada, are some of the well-known privacy regulations that protect the confidentiality of electronic healthcare information.

In order to protect the privacy of the respondents to which the data refer, released data were at first "de-identified" by removing all explicit identifiers such as names, addresses, and phone numbers. However this de-identified data could still have other implicit identifying characteristics such as race, birth date, sex, and postal code which, when considered all together, can uniquely, or almost uniquely pertain to specific individuals. These sets of characteristics are often called *quasi-identifiers*.

For instance, in one study, Sweeney estimated that 87.1% of the US population can be uniquely identified by the combination of their 5-digit zip code, gender, and date of birth because such records can be linked to publicly available databases such as voter lists and driving records. To prove her point, Sweeney re-identified a series of supposedly anonymous medical records including one belonging to William Weld – the governor of Massachusetts at the time – using a voter list she purchased from the city of Cambridge, Massachusetts for a mere $20 [21][37][38].

To illustrate the concept, consider Table 1, which exemplifies medical data to be released. In this table, data have been de-identified by suppressing names and Social Security Numbers (SSNs) so not to explicitly disclose the identities of patients.

Table 1 De-identified private table (medical data)

SSN	Name	Race	DOB	Sex	ZIP	Marital Status	Disease
		asian	64/04/12	F	94142	divorced	hypertension
		asian	64/09/13	F	94141	divorced	obesity
		asian	64/04/15	F	94139	married	chest pain
		asian	63/03/13	M	94139	married	obesity
		asian	63/03/18	M	94139	married	short breath
		black	64/09/27	F	94138	single	short breath
		black	64/09/27	F	94139	single	obesity
		white	64/09/27	F	94139	single	chest pain
		white	64/09/27	F	94141	widow	short breath

Table 2 Non de-identified publicly available table

Name	Address	City	ZIP	DOB	Sex	Marital Status
….	….	….	….	….	….	….
….	….	….	….	….	….	….
Sue J. Doe	900 Market St.	Utah	94142	64/04/12	F	divorced
….	….	….	….	….	….	….

However, notice that there is only one divorced female (F) born on 64/04/12 and living in the 94142 area. This combination, if unique in publicly available databases such as in Table 2, identifies the corresponding tuple as pertaining to "Sue J. Doe, 900 Market Street, Utah", thus revealing that she has reported hypertension [7].

In order to overcome the potential for a privacy breach, some researchers tried to further de-identify the data by using techniques such as scrambling and swapping values and adding noise to the data while maintaining an overall statistical property of the result. However, this compromised the integrity, or truthfulness, of the information released [1][16][31][41].

In a different direction, intensive research has been directed towards the anonymization of the data. Although guaranteeing complete anonymity is obviously an impossible task, the *k-anonymity* concept has been introduced: "A data release is said to satisfy k-anonymity if every combination of values of quasi-identifiers can be distinctly matched to at least k individuals in that release" [31].

1.2. Research Objective

Several algorithms were developed with the purpose of making de-identified data k-anonymous [7][40], hence readily available for researchers. However, we are only concerned with the methods that aim to achieve k-anonymity through full domain global recoding, hierarchical generalization and minimal suppression, as will be motivated in the next chapter. Mainly, two of the most popular approaches that fall under the former specifications and that were heavily used so far for clinical data [34][35] are Sweeny's Datafly algorithm [34] and Samarati's algorithm [31].

So far no one has empirically evaluated these algorithms in order to recognize which does a better job in balancing satisfactory privacy with minimum information loss, or how their solution compares with respect to the optimal one. More importantly, these approaches always rely on some heuristic that would approximate a "good" solution rather than actually finding the optimal one with respect to any given preference or information loss metrics.

Other existing methods, such as Incognito [23], tend to find all the possible solutions. However, a major drawback of such approaches is that the number of solutions they return is usually very high, and it is impractical to check the information loss of all of them in order to find the optimal one.

Resolving the above issues is very important. Accordingly, by assuring better solutions, researchers will benefit immensely, since the better the quality of the anonymized data, and the less the information loss, the more valuable that data is for their research. Therefore, our objective is to evaluate these algorithms and determine whether a better one can be devised in order to efficiently find an optimal solution.

1.3. Thesis Contribution

The contribution of this thesis is twofold:

- First, we present the results of an empirical evaluation of Datafly and Samarati's algorithms.
- Second, we propose our own approach, a new method to efficiently find an optimal solution.

1.4. Thesis Structure

In this thesis, we start by introducing some preliminary concepts and summarizing previous work related to k-anonymity. Then in chapter 3, after highlighting the methodology and selecting the information loss metrics, we present the results of an empirical evaluation. We show that, with respect to three information loss metrics, Samarati's approach usually gives better results than Datafly's, and that overall, the optimal solution is rarely attained by either algorithm.

Next, in chapter 4, we identify the set of solutions that are the candidates to be optimal with respect to the information loss metrics. Then we propose our own approach, a new method to efficiently find this set and hence the optimal solution. In the process, we illustrate why the two algorithms above, on many occasions, cannot find that solution. Moreover, we discuss some programming optimisations that are used in our implementation.

Then, in chapter 6, we note that Incognito efficiently returns the set of all solutions. The disadvantage is that this set is usually large, and it takes a lot of time to evaluate all its content in order to locate the optimal solution. Nevertheless, it can be argued that this approach can be altered to find the same set of solutions as our algorithm. Therefore, we compare the two from an efficiency perspective and we show that in most cases we outperform Incognito or have similar performance. Finally, in chapter 7, we state our conclusions, recommendations and future work.

Chapter 2. Background

In this chapter we will highlight the definitions and previous work that are relevant to our research.

2.1. Preliminary Concepts and Definitions

In what follows, we assume the existence of an already de-identified private[1] table PT to be anonymized. The rows in PT may be referred to as *tuples*, and the table is assumed to have at least k tuples. Moreover, the columns in the table are the *attributes*, and unless otherwise mentioned, the set of PT's attributes will be strictly considered as the quasi-identifier.

(Def. 1) Quasi-identifier (QI): A set of attributes in PT that, in combination, can be linked with external information to re-identify the respondents to whom information refers [9][31]. Examples of common quasi-identifiers are [6][13][14][15]: dates (such as, birth, death, admission, discharge, visit, and specimen collection), locations (such as, postal codes, hospital names, and regions), race, ethnicity, languages spoken, aboriginal status, and gender.

One of the methods applied in order to satisfy k-anonymity is the generalisation of data so that the tuples in PT can be distinctly matched to at least k other tuples. Because of the nature of clinical data, we are mainly concerned with hierarchical generalization.

(Def. 2) Hierarchical generalization: The act of generalizing tuples in PT to satisfy k-anonymity, with regard to the well-defined hierarchies of each attribute [31].

[1] By private we mean the data belongs to some institution, as opposed to publicly available data.

(Def. 3) Generalization hierarchy (GH): Hierarchies related to each attribute are assumed to exist, where leaves consist of the data that could be found in PT, and the rest of the levels are generalization of these data accordingly [5][7][31].

Examples of generalization hierarchies are given in Figure 1 to Figure 3. The higher the attribute, the more general it is.

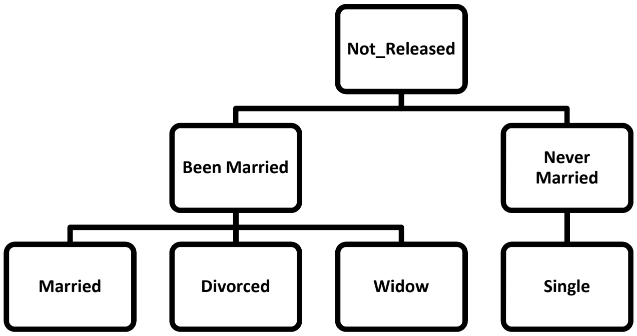

Figure 1 GH for Marital Status

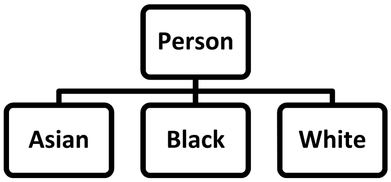

Figure 2 GH for Race

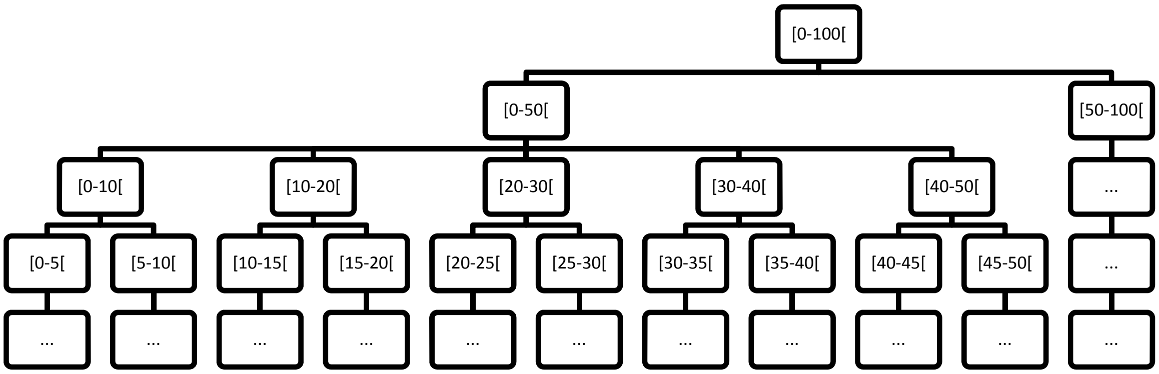

Figure 3 GH for Age

Generalization of PT can happen either by local recoding or global recoding.

(Def. 4) **Global recoding:** The act of generalizing an attribute in PT, through all the tuples, to the same level in the respective generalization hierarchy of that attribute [7].

(Def. 5) **Local recoding:** As opposed to global recoding, this is a cell-level generalization of each tuple in PT independently [7][40].

Table 3 De-identified table

Race	Marital Status	Age
asian	married	47
black	single	21
asian	married	49
white	widow	45
white	married	45
white	married	47
black	single	24
asian	married	49

Table 4 2-anonymized via local recoding

Race	Marital Status	Age
person	married	47
black	single	*[20-25[*
asian	married	49
white	*been married*	45
white	*been married*	45
person	married	47
black	single	*[20-25[*
asian	married	49

Table 5 2-anonymized via global recoding

Race	Marital Status	Age
asian	*been married*	*[45-50[*
black	*never married*	*[20-25[*
asian	*been married*	*[45-50[*
white	*been married*	*[45-50[*
white	*been married*	*[45-50[*
white	*been married*	*[45-50[*
black	*never married*	*[20-25[*
asian	*been married*	*[45-50[*

Table 6 Global recoding with suppression

Race	Marital Status	Age
asian	married	*[45-50[*
black	single	*[20-25[*
asian	married	*[45-50[*
white	married	*[45-50[*
white	married	*[45-50[*
black	single	*[20-25[*
asian	married	*[45-50[*

Table 4, Table 5 and Table 6 are generalizations of Table 3 with respect to the generalization hierarchies found in Figure 1, Figure 2 and Figure 3. They are all considered to satisfy 2-anonymity.

Clearly, we can see in Table 4 that, via local recoding, we have a minimal loss of information. However, we will not consider this method in this thesis for two main reasons: (i) this approach has been proven to be NP-Hard [26] and (ii) the output is not practical to be used for research on clinical data under the existing statistical software. These software mainly expect all the data in a column to have similar format as opposed to the case in Table 4 where, in the age attribute, some tuples use a number while others use an interval. The regression procedure for example, which is fairly popular, cannot be processed within this format.

Therefore, we shift our attention to global recoding. Table 5 and Table 6 are typical examples. In Table 5, we simply generalize one or more variables throughout the table. The reason why we generalized marital status and age (only) once (one level up the hierarchy) with respect to the corresponding GHs is: (a) it now satisfies 2-anonymity, and (b) additional generalizations, e.g. two levels up for the age, could have just as well satisfied 2-anonymity but at the cost of unnecessary information loss. Therefore, the goal is to find the minimal generalization required to satisfy k-anonymity while preserving as much information as possible.

(Def. 6) Information loss metrics: The metrics used to calculate by how much the data in the result table differ from the original table after generalization, therefore how much information we lost. There exist many metrics to give an estimate of such loss [4][10][17][39][44].

(Def. 7) Minimal generalization: Given a table PT, different possible generalizations exist. Not all generalizations, however, can be considered equally satisfactory. For instance, the trivial generalization bringing each attribute to the highest possible level of generalization, thus collapsing all tuples in PT to the same list of values, provides k-anonymity at the price of a huge information loss [31].

(Def. 8) Tuple suppression: The act of removing a tuple from a table PT [6][8][31]. In Table 6, tuple suppression was used as a complementary approach to hierarchical generalization. A suppression of one or more tuples is used to "moderate" the generalization process when a limited number of *outliers* (i.e., tuples with fewer than k occurrences) would force a great amount of generalization [31]. For example, in Table 6, by removing only one tuple, we were able to satisfy k-anonymity without the need to generalize the marital status as in Table 5, thus overall, losing less information.

Obviously, we are not interested in the suppression of more tuples than necessary to achieve k-anonymity at a given level of generalization. Therefore, this approach should be controlled. That is why we suppose that we will be provided a suppression limit.

(Def. 9) Suppression limit: The maximum number of tuples that we are allowed to suppress in order to achieve k-anonymity [22].

Now that the main approaches have been introduced, we will go through additional important concepts and definitions.

(Def. 10) Distance vector: This is the measure of the level of generalizations of each attribute. For example, for Table 3 (for convenience, now partly replicated in Table 7 below), the vector [0,1,1] suggests to generalize, in all rows, the second attribute once regarding to its corresponding hierarchy (Figure 1), and the third attribute once regarding the hierarchy in Figure 3, while the first attribute (Race here) remains intact [31].

Table 7 Hierarchical generalization with regard to the vector [0,1,1]

Race	Marital Status	Age
asian	married	47
black	single	21
asian	married	49
...	...	...

[0,1,1] ➔

Race	Marital Status	Age
asian	*been married*	*[45-50[*
black	*never married*	*[20-25[*
asian	*been married*	*[45-50[*
...	...	...

(Def. 11) Lattice: This is a collection of distance vectors and their interconnections, set up as a hierarchy going from the *null* vector to the *max* allowed generalizations. For example, consider a table similar to Table 3 but without the age column. According to the corresponding hierarchies of the first two attributes in Figure 1 and Figure 2, the maximum allowed generalization is the vector [1,2] and the corresponding lattice is the following [31]:

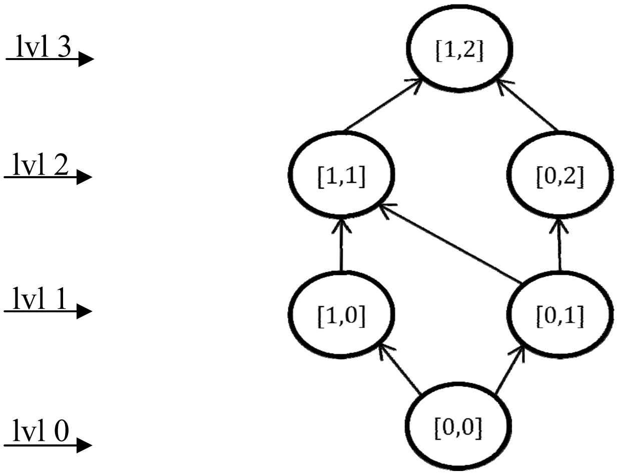

Figure 4 A lattice

(Def. 12) Generalization strategy: Every path in the lattice going from the bottom vector to the max (topmost) vector with respect to the corresponding arrows is called a strategy. In Figure 4, one strategy could be {[0,0] ➔ [0,1] ➔ [1,1] ➔ [1,2]} [31].

(Def. 13) Vector length: The measure of total generalizations implied by a distance vector. For example, the length of the vector [0,2,1] is 3.

(Def. 14) Lattice level: The set of vectors with equal length in the lattice. The lattice in Figure 4 has 4 levels. At level 0, we have the vector [0,0], and at level 3 the vector [1,2]. The other vectors are at levels 1 and 2 respectively.

(Def. 15) Vector height: The level in the lattice where a vector resides.

(Def. 16) Unique items: The distinct data items belonging to an attribute. For example, in Table 3, the unique items of the variable Race are: Asian, Black and White. These unique items are as well leaves of the corresponding hierarchy (Figure 2).

(Def. 17) Equivalence classes (EC): The tuples of QI that are uniquely distinguishable from other tuples. For example, in Table 5, we have 3 equivalence classes: {asian, been married, [45-50[}, {black, never married, [20-25[} and {white, been married, [45, 50[}.

(Def. 18) Frequency Set: The frequency set of PT is a mapping from each EC in PT to the total number of tuples equivalent to this EC in T (the size of EC) [23].

(Def. 19) Optimal solution: The vector used to generalize the table PT in a way that satisfies k-anonymity while having the minimal information loss possible (when compared to all other possible solution vectors) with respect to a given metric.

2.2. Previous Work

Many studies have been conducted towards achieving k-anonymity. Consequently, many strategies and methods were created in order to accomplish that task [7]. Nevertheless, as illustrated earlier, the most practical for medical data are the methods that aim to attain k-anonymity through global recoding (Def. 4), hierarchal generalization (Def. 2), and minimal suppression (Def. 7). These rules stem from a consensus based on the experiences of the authors analysing and de-identifying clinical data. Therefore, the methods that do not comply with these properties, including Bayardo and Agrawal's K-Optimal approach [4] which does not use hierarchical generalization, and the algorithms that uses local recoding [2][11][17][24][43][44], are not of any interest to our research and will be completely disregarded.

Iyengar [20] proposed an approach based on genetic algorithms and solves the k-anonymity problem using an incomplete stochastic search method. As stated by the author, the method does not assure the quality of the solution proposed, but experimental results show the validity of the approach. Moreover, Winkler [42] proposes a method based on simulated annealing for finding locally minimal solutions, which requires high computational time and does not assure the quality of the solution [7]. Therefore, since it is known that there is no guarantee on the quality of their output, we will not discuss these two approaches any further.

LeFevre, DeWitt and Ramakrishnan [23] propose an algorithm for computing k-minimal generalization, called Incognito, which takes advantage of a bottom-up aggregation along dimensional hierarchies and a priori aggregate computation. However, a major drawback of this approach is that it returns the set of all possible solutions in the lattice, and thus it is impractical to check the information loss of all of them in order to find the optimal one. Nevertheless, the method used to retrieve this set is interesting and proven to be efficient. We will revisit this approach in more detail in chapter 6.

Finally, two of the most popular approaches are Samarati's algorithm and the Datafly algorithm.

2.2.1 Samarati's algorithm

Samarati makes the assumption that the best solutions are the ones that result in a table having minimal generalizations. That is, the vector solution(s) with the minimal height possible. Therefore, her algorithm is meant to search the lattice and identify the lowest level on which one or more solution vectors are found (i.e. the generalizations that satisfy k-anonymity with minimal suppression).

The following is a summary of Samarati's algorithm:

1. Consider a table T = PT[QI] to be generalized (takes into consideration only the quasi-identifiers fields).
2. Consider the middle height in the area of search (area of search is initially the whole lattice).
3. Check if at that height there is at least one node that satisfies k-anonymity with minimum suppression (the minimum suppression variable would be already set) then,

   a. If not the minimum, specify the upper half as the new area of search.
   b. If minimum, specify the lower half as the new area of search.
4. If the area of search consists of more than one level in the lattice, repeat step 2. Otherwise, return a solution residing on this level.

In other words, Samarati's approach takes advantage of the fact that if at any level in the lattice a solution can be found, then all the levels above this one must contain solutions and therefore only the lower levels need to be checked. Hence, the algorithm goes through the lattice with a binary search, always cutting the search space in half, going down if a solution is found at a level, or up if not. Eventually, the algorithm finds the solution(s) with the lowest height, thus with the least generalizations.

Afterwards, as Samarati suggests, the best solution on that level (i.e. with the least information loss) with respect to a given preference (i.e. information loss metric) is chosen.

2.2.2 Datafly algorithm

Sweeney considers that the best solutions are the ones that are attained after generalizing the variables with the most distinct values (unique items). The search space is again the whole lattice; however, this approach only goes through a very small number of nodes in the lattice to find its solution. Thus, from a time perspective, this approach is very efficient (hence the name Datafly).

Here is a summary of the Datafly algorithm:

1. Consider a table MT = PT[QI] (takes into consideration only the quasi-identifiers fields)
2. While k-anonymity is not achieved and the count of the remaining rows that do not comply to k-anonymity is more than k:
   a. Get the number of distinct values of each attribute in MT
   b. Generalize the attribute with the most distinct values
3. Suppress the remaining rows

In other words, at each node in the lattice, check in the data which attribute has the most unique items and generalize that attribute one level up according to the corresponding hierarchy (i.e. go up one level in the lattice). Keep doing this until there are fewer than k rows not complying to k-anonymity, then suppress these remaining rows.

It is worth noting that in case the algorithm finds, at any point, many attributes having the same number of unique items, then one is chosen randomly.

It is easy to incorporate the suppression limit *MaxSup* in this procedure. In (2), instead, simply check if the count of the remaining rows that do not comply with k-anonymity is more than *MaxSup*. With this small alteration, the main idea of the algorithm remains intact. In addition, we gain a common ground between Datafly and Samarati's algorithms, which improves the conditions of the comparison.

2.2.3 Visualization

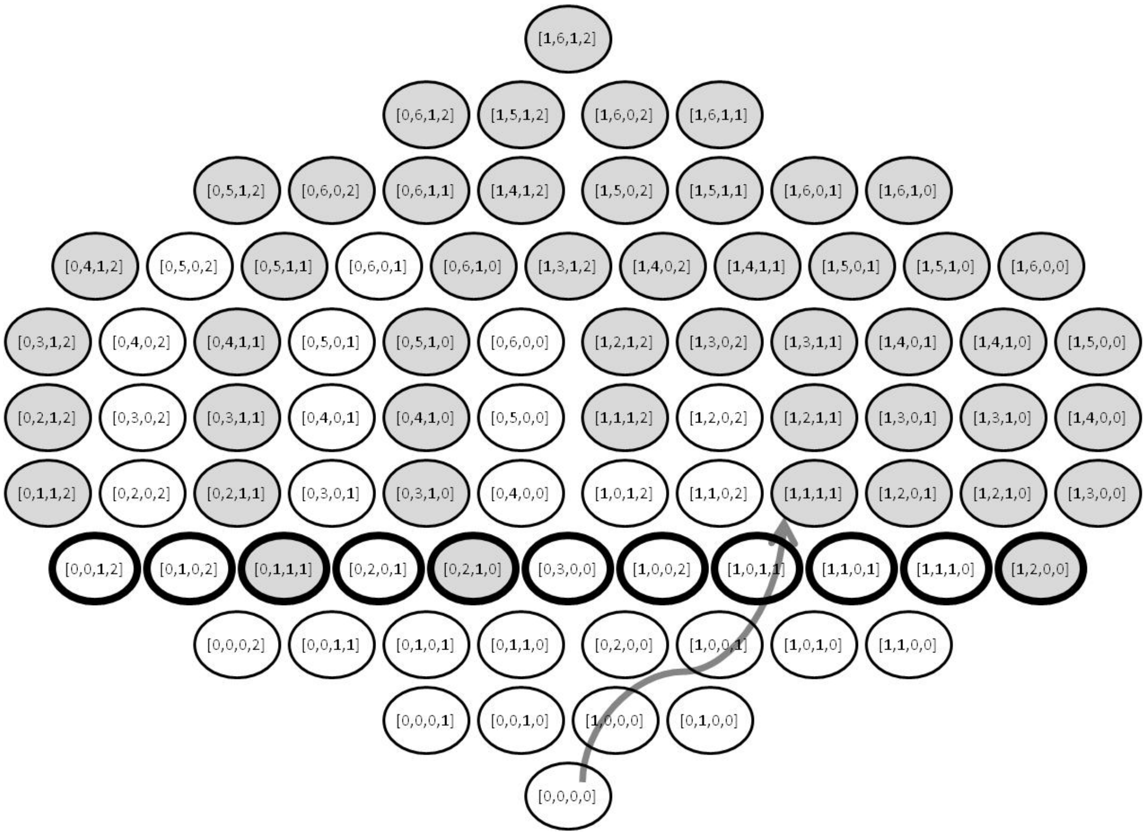

Figure 5 Visual comparison of Datafly and Samarati's algorithms

Figure 5 above illustrates a plausible lattice, where the nodes filled in gray are the possible solutions. Note that these solutions were selected for the sole purpose of emphasizing the output of both algorithms.

Samarati's solution is on level 3 of the lattice; the level where the vectors are highlighted in bold black lines. Datafly's solution path is the arrow covering [0,0,0,0] → [1,0,0,0] → [1,0,0,1] → [1,0,1,1] → [1,1,1,1]

In this particular example of a lattice and corresponding solutions, Samarati's result is strictly restricted to the solutions found on level 3, namely: [0,1,1,1] , [0,2,1,0] , [1,2,0,0]. On the other hand, Datafly solution could have been attained through any strategy in the lattice, including the ones on level 3, only guided by the number of unique items in each variable. In this case, the solution reached is on level 4.

2.3. Chapter Summary

In this chapter, we defined important preliminary concepts, and we stated the required algorithms' properties that are the most practical when working with clinical data, namely global recoding, hierarchical generalizations and minimal suppression limit. We identified three interesting approaches, and then out of these we highlighted Datafly and Samarati's algorithm with a brief summary and visualisation (Incognito will be detailed in chapter 6).

We showed, as Samarati suggested in [31], that Datafly ultimately walks through a specific generalization strategy in the lattice. From Samarati's perspective, this may be only the local minimum with respect to this strategy, not necessarily the global minimum. By global minimum, she means the solution with the smallest length (generalizations) in the lattice.

Datafly has the advantage of hitting any possible path in the lattice while Samarati's approach has the edge of hitting the solution with minimal length (with the assumption that the optimal solution has more probability to be around that area). It is worthwhile to compare the two and see which is getting better results ultimately, and how their results compares to the optimal one with regard to certain metrics. This will be the topic of the next chapter.

Chapter 3. Empirical Evaluation

In this chapter, we present the results of an empirical evaluation of Datafly and Samarati's approach, comparing their outputs to the optimal one. Although these algorithms do not necessarily locate the optimal solution, if they are found to produce solutions that are sufficiently close to the optimal one, then a case can be made for using them in practice, given that they are simple and well established and understood, not to mention that Datafly can execute very quickly on large data sets.

Incognito finds the set of all solutions, and consequently it always locates the optimal one. Given that this section focuses on optimality rather than on efficiency, Incognito will not be a part of this first experiment.

3.1. The Optimal Solution

The optimal solution with respect to any metric can be computed by running a brute force algorithm (for now). The idea is to go through the whole lattice node by node, check if a certain node is a solution (i.e. it satisfies k-anonymity with minimal suppression), and calculate its information loss with respect to a given metric. Eventually, an optimal solution is the one with the least information loss.

We also specify an "acceptable suppression threshold (*MaxSup*)", i.e. the suppression limit discussed in (Def. 9), provides the maximum number of records which can be suppressed from the data set. This value can be set taking into account the types of suppression techniques that can be applied to the data and compensate any power[2] loss from having missing records [25]. Therefore, as long as the number of suppressed records is below MaxSup, suppression should not play a significant role in deciding the optimality of the solution.

[2] The power to retrieve critical information or conclusions out of the anonymized data.

3.2. Measuring Information Loss

In the literature, there are many definitions of information loss measures. According to Domingo-Ferrer [10], information loss can be obtained by comparing the original data to the masked one, the more similar the data is, the less is the information loss. A similar definition is given by Xu et al. in [44]: the information loss measures “how well the generalized tuples approximate the original ones”.

Nowadays, there is no single information loss metric that is globally accepted by the community. However, when criticising the current metrics, researchers have implicitly pointed out many criteria that a good metric should satisfy [4][10][13][44]. For this reason, we picked three metrics that cover the majority of these criteria.

An information loss metric that takes into account the height of the generalization hierarchy is Precision or *Prec*. *Prec* was introduced by Sweeney [39] as an information loss metric that is suitable for hierarchical data. For every attribute, the ratio of the number of generalization steps applied to the total number of possible generalization steps (total height of the attribute hierarchy) gives the amount of information loss for that particular variable. For example, if age is generalized from a value in years to a value in five year intervals, then the information loss value in that particular cell in the table is ¼ (one step generalization over the number of total generalizations allowed. See Figure 3). Total *Prec* information loss is the average of the information loss across all quasi-identifiers in the data set. As a result, the more a variable is generalized, the higher the information loss. Moreover, variables with more generalization steps (i.e., more levels in their generalization hierarchy) tend to have less information loss than ones with shorter hierarchies.

Another commonly used information loss metric is the discernability metric or DM [4]. DM assigns a penalty to every record that is proportional to the number of records that are indistinguishable from it, and following the same reasoning, DM assigns a penalty equal to the whole data set for every suppressed record (since suppressed records are indistinguishable from all other records).

The DM metric is calculated as follows:

$$DM = \sum_{|EC| \geq k} |EC|^2 + \sum_{|EC| < k} |D||EC|$$

where $|EC|$ is the size of an equivalence class, and D is the total number of records.

Table 8 (a) is a data set, and (b) is its generalization with respect to [0,0,1]

Race	Marital Status	Age
asian	single	18
asian	single	18
asian	single	18
asian	single	13
asian	single	19
black	married	18
black	married	22
black	married	26
black	married	20
asian	single	22

(a)

Race	Marital Status	Age
asian	single	[15-20[
asian	single	[15-20[
asian	single	[15-20[
asian	single	[10-15[
asian	single	[15-20[
black	married	[15-20[
black	married	[20-25[
black	married	[25-30[
black	married	[20-25[
asian	single	[20-25[

(b)

However, DM is not monotonic within a generalization strategy due to the impact of the second term incorporating suppression. The example in Table 8 shows a data set and its generalization with respect to the vector [0,0,1]. For Table 8 (a) to satisfy 3-anonymity, seven out of ten records need to be suppressed, and therefore the DM value is 79. For Table 8 (b), the DM value is 55. This reduction in information loss as we generalize means that we would select the k-anonymity solution with the maximum generalization as the best one, which is counter-intuitive. Moreover, even with the introduction of *MaxSup* where we specify the accepted margin of suppression in order to achieve the least generalizations, DM could still favour less suppression over more generalization. It therefore makes sense not to include the suppression penalty in DM. In other words, we will use a modified version of DM as follows:

$$DM^* = \sum |EC|^2$$

The DM^* value for (a) in Table 8 is 16 and for (b) is 28.

The concept behind the DM has been criticized because it does not measure how much the generalized records approximate the original records [44]. For example, if we have a quasi-identifier such as age and six records with the following age values: 9, 11, 13, 40, 42, and 45, the minimal DM^* value is when all of the records are grouped into three pairs: <9,11>, <13,40>, and <42,45> (to achieve 2-anonymity). The criticism is that this grouping has a very wide range and that a more sensible grouping would have only two equivalence classes: <9,11,13> and <40,42,45>. In our context this criticism is not applicable, since we assume that all data are hierarchical and that the end-user would specify the age grouping in the generalization hierarchy.

The discernability metric has also been criticized because it does not give intuitive results when the distributions of the variables are non-uniform [24]. For example, consider two data sets, the first with 1000 records where 50 are male and 950 are female, and the second with 500 males and 500 females. If the gender is generalized to “Person”, then intuitively losing the 950 females should result in low information loss since the female records dominate the data set, and in this case having “Person” in the data is almost as good as having “Female”. However, the DM^* values show that the information loss for both data sets is the same after generalization.

One information loss metric based on entropy [41] has recently been extended to address the non-uniform distribution problem [13].

$$NE = \sum_{i=1}^{n} \sum_{j=1}^{r} -\log_2 \Pr\left(\frac{R_i(j)}{\bar{R}_i(j)}\right)$$

This calculates the Non-uniform Entropy (NE), where n is the number of rows in the data set, r the number of QIs, and Pr is the probability that a particular value in the row R_i is to be found in the generalized set in $\bar{R}_i$.

Returning to our example, the 50/950 male/female distributed data set has a NE of 286 bits whereas the 500/500 male/female distributed data set has a NE of 1000 bits. Therefore, the information loss in the former data set is much lower, and as stated above, this makes more intuitive sense.

3.3. Data Sets

The data sets used for the comparison are summarized in Table 9. The first three are publicly available for research, where the first (Adult) has been frequently used for similar studies [4][23]. The last three are real hospital and registry data sets and can be considered highly representative. The maximum height of a hierarchy is shown in Table 9 between parentheses near the corresponding attribute (More details in Appendix A).

Table 9 Summary information of the data sets

Description	Quasi-Identifiers	# Rows
Adult The adult data set from the UC Irvine machine learning data repository. This is an extract from the US census: ftp://ftp.ics.uci.edu/pub/machine-learning-databases/adult	• Age (3) • Profession (2) • Education (2) • Marital status (2) • Position (2) • Race (1) • Sex (1) • Country (3)	30,162
CUP Data from the Paralyzed Veterans Association on veterans with spinal cord injuries or disease: http://kdd.ics.uci.edu/databases/kddcup98/kddcup98.html	• ZIP code (5) • Age (4) • Gender (2) • Income (3)	63,411
FARS Department of Transportation fatal crash information: http://www-fars.nhtsa.dot.gov	• Age (4) • Race (1) • Month of Death (3) • Day of Death (2)	101,034
Pharm Prescription records from the Children's Hospital of Eastern Ontario pharmacy for 18 months. This is for inpatients only and excludes acute cases. This data is disclosed to commercial data aggregators.	• Age (4) • Postal code (FSA) (3) • Admission date (6) • Discharge date (6) • Sex (1)	63,441
ED Emergency department records from Children's Hospital of Eastern Ontario for July 2008. This data is disclosed for the purpose of disease outbreak surveillance and bio-terrorism surveillance.	• Admission date (3) • Admission time (7) • Postal Code (7) • Date of Birth (6) • Sex (1)	7,318
Niday A newborn registry for Ontario for 2006-2007: https://www.nidaydatabase.com/info/index.shtml	• Maternal postal code (7) • Baby DoB (4) • Mother DoB (6) • Baby sex (2) • Aboriginal status (2) • Language (2)	124,933

3.4. Methodology

We aim to compare the optimal solution to the output of Datafly and Samarati with respect to the three information loss metrics mentioned in section 3.2.

For each data set, the maximum suppression is set at 1%, 5%, and 10% of the total number of records. Therefore, we basically have 3 different experiments overall.

In practice, a minimal value of k=3 is sometimes recommended [10][12][41], but more often a value of k=5 is used [33][32][28][29]**.** To ensure a reasonable amount of variation in our analysis we use values of k between 2 and 15 inclusively.

As stated earlier, Datafly has a random element in the choice of its solution. When two or more quasi-identifiers have the same maximum number of unique items, one attribute has to be chosen at random [34]. In order to overcome this randomness, Datafly was processed 101 times for every value of k. For every iteration, the information loss is calculated with regard to all metrics. Finally, the average information loss is computed for every value of k with respect to each metric.

3.5. Results

The purpose is to highlight, in a readable manner, how far is the output of the two chosen algorithms from the optimal one. Therefore, the information loss of the optimal solution is considered as the baseline of the comparison, and Datafly and Samarati's outputs are represented as a percentage of that baseline.

For example, in the following charts, if Datafly's solution has an information loss value of 100% it means that the optimal solution and Datafly's output have exactly the same information loss. Alternatively, if it is 200% then that means the information loss of Datafly is twice as large.

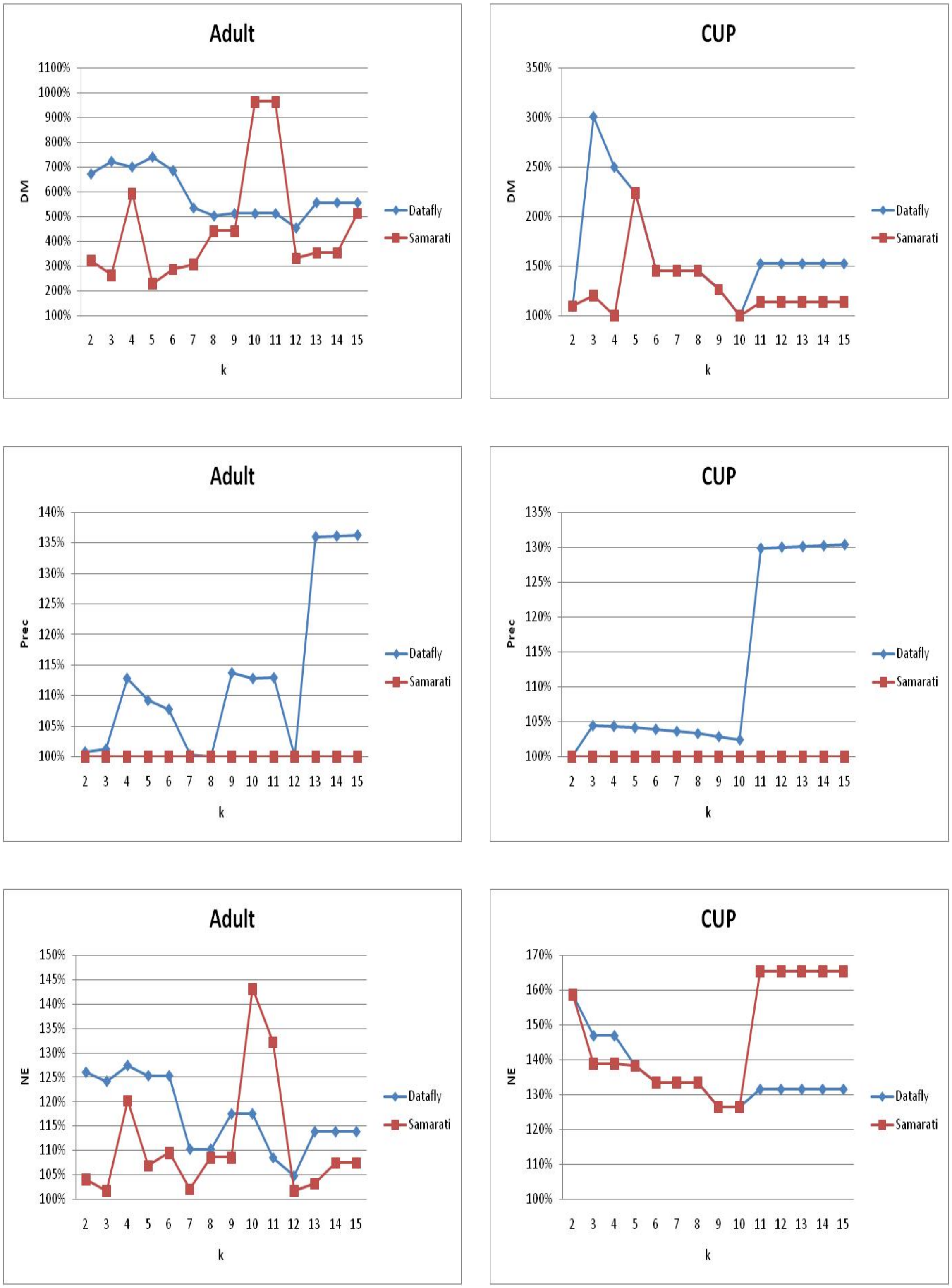

Figure 6 Information loss comparison for Adult and CUP data sets

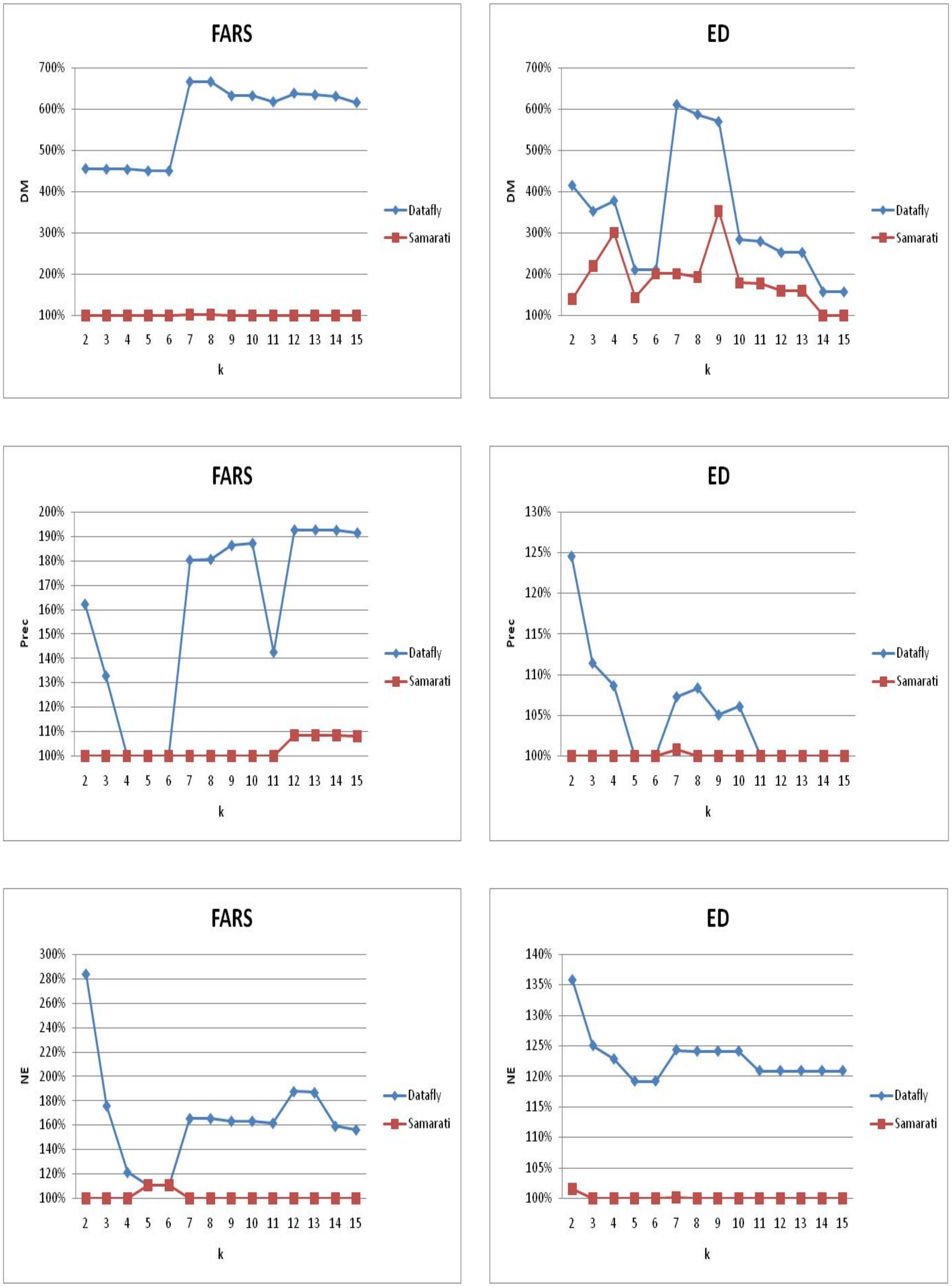

Figure 7 Information loss comparison for FARS and ED data sets

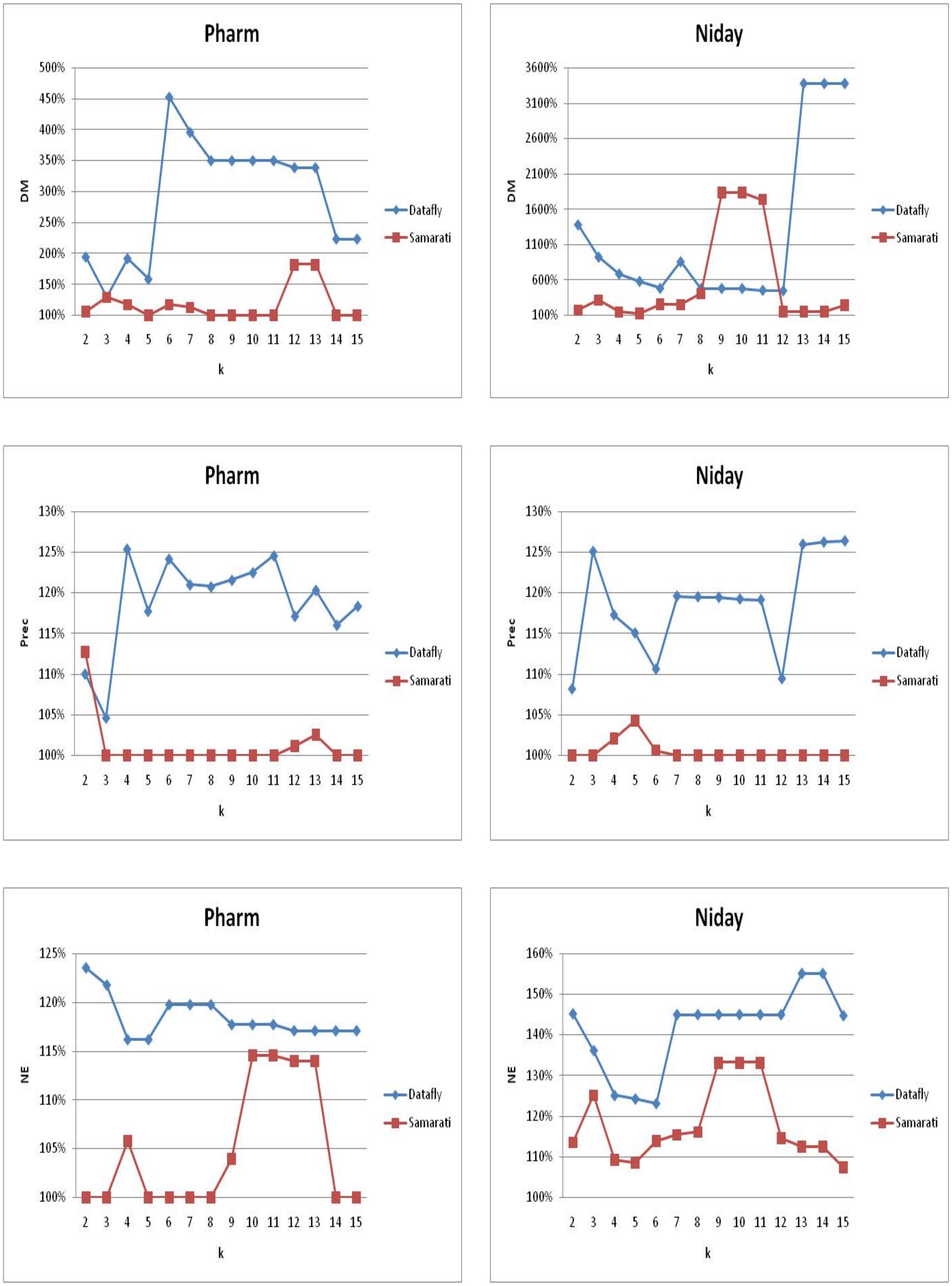

Figure 8 Information loss comparison for Pharm and Niday data sets

The information loss results on all data sets with respect to the aforementioned metrics, for the 5% suppression limit, are presented above in Figure 6, Figure 7 and Figure 8. The x-axis shows the value of k ranging from 2 to 15, while the y-axis is the information loss percentage of the algorithms with respect to each metrics separately.

The remaining information loss results for the 1% and 10% suppression limit are included in Appendix B and are consistent with those shown here.

3.6. Conclusion

It is clear from the graphs above that overall, and across multiple data sets, the performance of Datafly and Samarati did vary significantly depending on the data set, the value of k, and the suppression limit. However, on average, Samarati produced better results than Datafly with respect to all information loss metrics. Moreover, for the *Prec* metric, there is less of a difference between the optimal solution and Samarati's, and this is due to the nature of *Prec* which takes into account the height of the generalization hierarchy.

On the other hand, we also notice that the optimal solution is rarely attained by either algorithm, especially in the case of DM and NE, and in most cases the difference is very significant. Hence, because of the medical nature of the researches conducted on the anonymised clinical data, there is zero tolerance for mistakes and thus a better approach is needed, one that actually searches for the optimal solution as opposed to a mere approximation.

Furthermore, since the search space is limited to the lattice, and since there exist many helpful insights on the nature of this search problem, such an algorithm should be feasible.

Chapter 4. New Algorithm

4.1. Motivation

The results from the previous chapter show that overall, the optimal solution has significantly lower information loss than the output of the algorithms studied so far. However, although a brute force approach can locate the solution needed, the procedure is very slow and impractical, especially when we process large data sets with bigger lattices. Moreover, although other approaches that return the set of all possible solutions might be efficient, the search for the optimal one within this likely huge set is impractical. Hence, there is definitely a need to develop an enhanced algorithm that can efficiently find that optimal solution.

Still, looking at the properties of this problem, we see that it is only a search problem within a very precise search space, the lattice. Nevertheless, due to the cost of evaluating each node in the lattice to see if it satisfies k-anonymity with minimal suppression, any approach that would try to go through a relatively large number of nodes within that lattice while searching for the optimal solution would be considered impractical, time wise. Actually, after observing our implementation of all the above algorithms, we found that the function Satisfy(V), which checks if a vector V satisfies k-anonymity with minimal suppression (where k and MaxSup are already set by the user), consumes almost all of the overall time of the process. Therefore, for any approach to be practical, it needs to deal with the time complexity of that function first. In what follows, we will refer to the evaluation of a node V to see if it satisfies k-anonymity with minimal suppression, as Satisfy(V).

The aforementioned two algorithms deal with this issue in a heuristic manner. That is, they cut large parts of the search space based on pre-made assumptions or heuristics, while unsure at this point whether the rest of the nodes actually contain the optimal solution or not. That way, they ultimately lower the number of calls to that function, and therefore the overall computation time, at the price of a relatively poor output as shown from the results of the previous chapter.

Therefore, before thinking of a practical approach to find the optimal solution, one has to address the above probelm. Our main attempt to solve this particular issue is twofold:

1- Enhance the implementation of Satisfy(V) and make that function as efficient as possible.
2- Minimize the number of invocations of the function Satisfy(V) without cutting any part of the search space and while keeping full confidence in finding the optimal solution.

(1) is the obvious approach and is actually strictly related to the development phase, not to any of the algorithms per se. Nevertheless, it proved to be very critical and allowed us to consider searching the whole lattice. In chapter 5, we show some of the optimizations that were very helpful in our implementation.

On the other hand, (2) at first seems to be an abstract goal. However, in the next sections we show that simple observations, and a bit more insights and knowledge of the overall nature of the problem at hand, can make (2) actually very simple.

4.2. Observations

4.2.1 Prediction

Samarati in [31] proved that: "*the number of tuples that need to be removed to satisfy a k-anonymity requirement can only decrease going up in a strategy. Hence, the cardinality of the table enforcing minimal required suppression to satisfy a k-anonymity constraint can only increase going up in a strategy"*. From this she also proves that: *"if a table Tz with a distance vector DVi,z cannot provide k-anonymity by suppressing a number of tuples lower than MaxSup, then also all tables Tj such that DVi,j < DVi,z cannot*".

This means that for any two nodes in the lattice that are directly connected, say V_1 and V_2, where V_2 is a generalization of V_1 (i.e. one step higher in the lattice within the same strategy), if V_1 satisfies k-anonymity with minimal suppression, V_2 must as well be a solution and thus also satisfy these requirements.

The contraposition is also true. That is, if V_2 is a generalization of V_1, and V_2 cannot satisfy k-anonymity with minimal suppression, then V_1 is not a solution either. Formally we get two equations out of this:

a) if ($V_i < V_j$ AND Satisfy(V_j) == false) $\rightarrow$ Satisfy(V_i) = false.

b) if ($V_j < V_i$ AND Satisfy(V_j) == true) $\rightarrow$ Satisfy(V_i) = true.

Where by $V_a < V_b$ we mean that V_b is a generalization of V_a , and where Satisfy(V) is a function that evaluates if a vector V satisfies k-anonymity with minimal suppression and returns true or false accordingly.

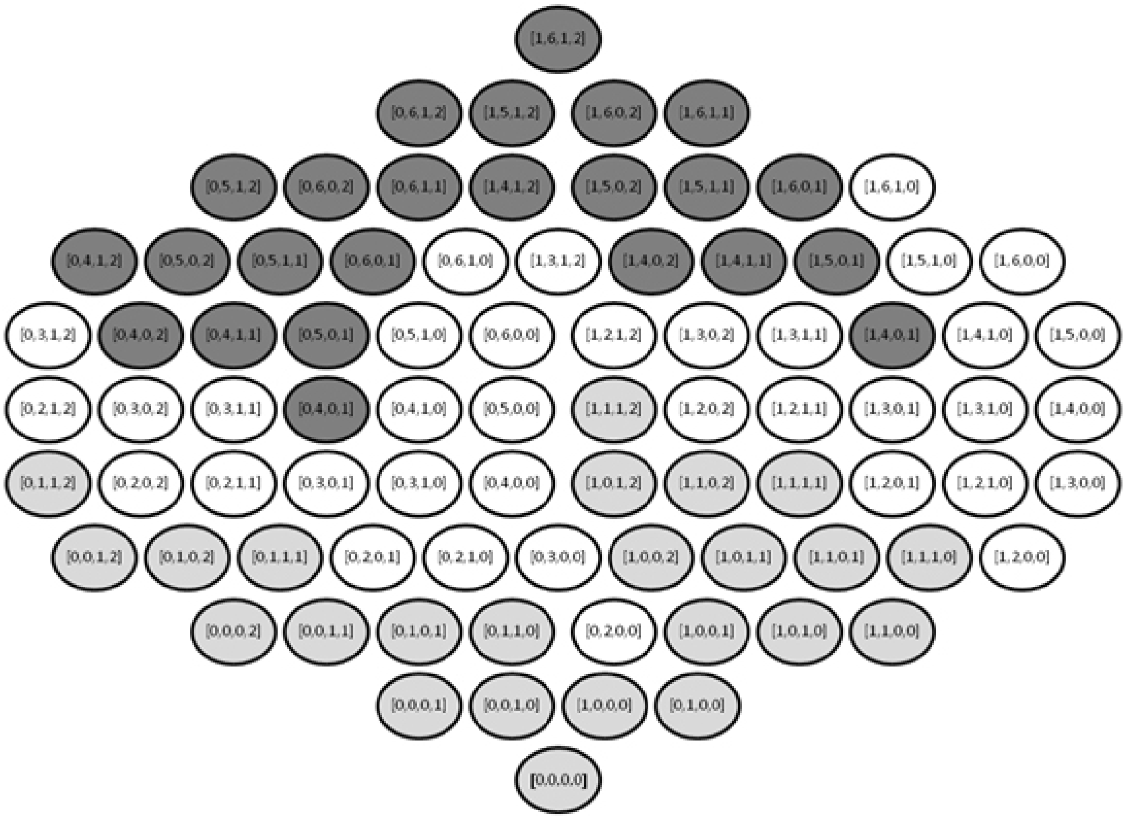

Figure 9 Lattice illustrating "Predictions"

If we correctly use the above formulas, we can end up with a powerful "prediction[3]" procedure. The idea is that by knowing if one vector (or node in the lattice) is a solution or not (satisfies k-anonymity with minimal suppression), we can truly deduce if the vectors of the same strategy are solutions or not without the need to actually them.

To illustrate this, consider the lattice in Figure 9. The vector [1,1,1,2] on level 5 was evaluated and was found to be a non-solution (and highlighted in light-gray, the color pointing to non-solutions in this example). According to equation (a) above, the vectors that are strictly related to this vector and are one level lower in the lattice are as well non-solutions, namely {[0,1,1,2], [1,0,1,2], [1,1,0,2], [1,1,1,1]}. However, since we now know that these are not solutions, we can run the same logic on each one of them. In other words, without reprocessing Satisfy([0,1,1,2]), we know it is a non-solution, and thus all the nodes related to it and one level lower in the lattice are also non-solutions. Accordingly, we end up deducing that all the light-gray nodes in Figure 9 are non-solutions.

In addition, the vector [0,4,0,1] on level 5 in the same lattice was processed by the "satisfy" function and was found to be a solution, and was highlighted in dark-gray. According to equation (b), the vectors that are strictly related to it and one level higher in the lattice are solutions too, namely {[0,4,0,2], [0,4,1,1], [0,5,0,1], [1,4,0,1]}. Again, since we now know that the vectors in this set are all solutions, we can apply the logic of equation (b) to each one of them. We end up predicting all the dark-gray nodes in Figure 9 as solutions.

Note that we did not need to invoke the "Satisfy" function for any of the predictions. We called it only twice and ended up tagging the lattice with 48 solutions and non-solutions. As the whole lattice has 84 nodes, we now know the status of more than 57% of the nodes, and already saved ourselves 46 expensive evaluations of the vectors. This "prediction" mechanism is clearly a powerful tool that will allow us to further investigate the whole lattice and search for an optimal solution while minimizing the number of evaluations.

[3] The word "prediction" is somewhat weak here since there is no real prediction, but logical deduction.

4.2.2 Candidates

The next step is to locate where in the lattice the optimal solution is likely to reside. By optimal solution we mean the solution with minimal information loss with respect to a certain metric.

As argued earlier, a metric that correctly calculates the information loss, after generalization and suppression, needs to take many properties of the data into consideration. More importantly, such an InfoLoss(V) function needs to be monotonic[4] even while giving a penalty for suppressed rows.

c) **if $(V_i < V_j) \rightarrow (\text{InfoLoss}(V_j) \geq \text{InfoLoss}(V_i))$**

The above equation means that if a vector V_j is a generalization of V_i, then the information loss caused by the former is greater than or equal to the information loss caused by the latter. This is the case in all the information loss metrics we chose in this thesis.

The reasoning behind this observation is rather simple. Consider the example in Figure 9. The two vectors [0,4,0,1] and [0,5,0,1], on levels 5 and 6 respectively, are solutions. Since these vectors are connected within the same strategy, the one with more generalizations will definitely cause more (or equal) information loss. Also, since the user specifies the suppression limit, this means any suppression under that limit is accepted and has been compensated for already. Hence, if any penalty should be applied on the suppressed rows, it should be in a way that respects equation (c) above.

Note that, the above logic is not true for unrelated vectors. That is, if two vectors do not belong to the same strategy, even if one has higher generalization with respect to the hierarchies, we cannot infer that it causes more or less information loss.

Based on (c), one would assume that the set of solutions that are candidates to be an optimal one are the set of all the “local minima” in the lattice, where by “local minimum” we mean the first solution (the one with the least generalizations) in a certain strategy.

[4] A monotonic function is always non-decreasing or non-increasing, and it does not oscillate in relative value.

This reasoning is valid; yet, knowing that many strategies interrelate, we can go even further and eliminate any local minimum that is actually a generalization of other local minima. Eventually, we find the set of *Optimal Solution Candidates* (OSC).

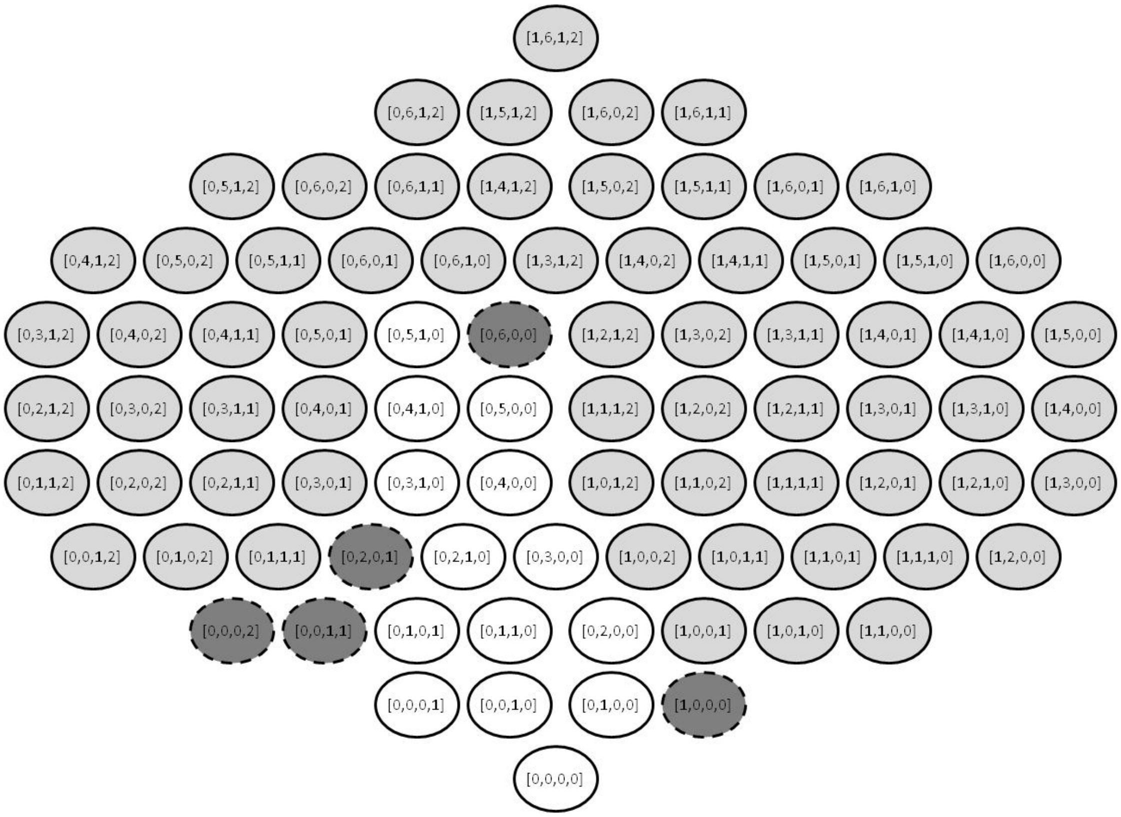

Figure 10 Optimal solution candidates

Figure 10 shows a lattice where the white nodes are non-solutions, the gray nodes are solutions, and the dark-gray nodes are the set of OSC. In this lattice, the OSC vectors are not related to each other, and all other solution nodes in the lattice are generalizations of one or more vector in this set. Therefore, based on equation (c), the optimal solution has to be one of the local minima specified in OSC.

4.3. Limitation of Datafly and Samarati's Algorithms

In order to explain why most of the time, these two algorithms do not agree with the brute force's output in the results obtained, we show that these do not actually search for the optimal solution but an approximation of it.

We know that Samarati searches for the solution with the least generalization. However, as shown in Figure 10, the optimal solution can be one of the OSC, where the solution(s) with the least generalization are only a subset of this search area. Therefore, Samarati's output always has a chance to be an optimal solution, and that is why it had good results when compared to Datafly. However, this chance varies dramatically with regards to many variables such as the data, k, MaxSup, the lattice size, etc., and is almost never 100% unless all the optimal solutions candidates are the solutions with the least generalization, which is rarely the case.

On the other hand, Datafly is basically of a heuristic nature, going through the minimum number of nodes possible in the lattice. This approach does not even assure a solution that is part of the OSC. For example, in Figure 10, if Datafly finds itself at the vector [0,1,1,0], it has three options to go to, namely {[1,1,1,0], [0,2,1,0], [0,1,1,1]} where two of them are solutions that are not even a part of the OSC. Actually, out of the many possible strategies, there is only a very few that overlap with the possible optimal solutions.

Moreover, since Datafly chooses its path, or the next generalization at a given node, based on the variable with the most unique items, it can be compared to a greedy approach solving a "Traveling Salesman" type of problems. These approaches follow the problem solving metaheuristic of making the locally optimum choice at each stage with the hope of finding the global optimum [18]. However, as Bang-Jensen, Gutin and Yeo show in [3], such approaches do not guarantee an optimal solution, or not even a good solution. They also provide a characterization of the cases when such a greedy approach may produce the unique worst possible solution on similar problems.

That is why Datafly has a very slim chance to locate the optimal solution. Nevertheless, the reason why it is sometimes competitive with Samarati is because it can always produce outputs that are near the area where the optimal solution resides in the lattice and thus might have less or similar information loss than the output of Samarati's algorithm.

The main point here is that these algorithms fail to find the optimal solution because they are only searching for an approximation. Therefore, we believe that an algorithm that aims to systematically find the optimal solution needs to locate the set of OSC first and then search within this set.

4.4. New Approach

Having good insight of how to efficiently search the lattice, and knowing exactly where to search in the lattice for the optimal solution, constitutes an ideal approach to successfully and efficiently achieve optimal k-anonymity.

Furthermore, knowing that the optimal solution is nothing but the solution that is considered the best from an information loss (IL) perspective, with respect to a certain metric, we understand that an appropriate approach should be flexible enough to take into consideration the users' choice of information loss metric.

Therefore, our approach takes as input 2 parameters: The data set to be k-anonymized, and an IL metric. The level of anonymization k, and the suppression limit MaxSup, are also parameters but they will be set as global variables.

Accordingly, this method mainly consists of two steps:

1- Find the set of OSC
2- Go through this set of vectors, one by one, check their information loss with respect to the given metric and return the node with the least information loss.

Step (2) is straightforward; a simple loop through the set of vectors where we check each one for its respective information loss, while keeping track of the vector having the least information loss so far. Eventually the vector with the least information loss will be considered as the optimal solution with respect to the given metric.

However, step (1) is somewhat trickier. In order to locate the set of OSC, we will have to identify which of the nodes in the lattice are solutions and which are not. The idea is to find a way to traverse the lattice, minimizing the evaluation of the vectors (i.e. Satisfy(V)) while maximizing the prediction made about the other nodes.

This can be done recursively going vertically through the lattice as a binary search over all the strategies, each possible strategy at a time. That way, based on the location of the solutions, we traverse the lattice up and down while making the respective predictions. The more strategies covered, the more predictions are made and the fewer calls to Satisfy(V) are needed. Eventually, after a very small number of evaluations, we will have most of the lattice already tagged as solutions or non-solutions, and accordingly the search for the OCS would be very quick. The status of all the nodes in the lattice will ultimately be identified.

Table 11 is the pseudo-code for the function that computes step (1). But first, Table 10 shows a list of auxiliary functions used.

GetOSC is a recursive function that takes as parameters the minimal vector of a (sub)lattice *Bnode* (initially the null vector) and the maximum vector *Tnode* (initially the vector with the maximum generalizations). The idea is to go through every strategy in the given lattice and evaluate the nodes for solutions and non-solutions, while keeping track of the local minima found.

This function makes use of the "*Prediction*" procedure highlighted earlier, as well the *"Rollup"* technique, a bottom-up aggregation along the lattice. This is explained in more detail in section 5.1.4, but this is not shown in the pseudo code for the sake of simplicity.

Table 10 Auxiliary functions

Function	**Description**
InfoLoss(*node*)	Computes the information loss for a particular node in the lattice. The information loss should apply to equation (c) above.
Satisfy(*node*)	Evaluates if a given vector is a solution, i.e., if it satisfies k-anonymity with minimal suppression. Returns True or False.
IsTaggedSolution(*node*)	Determines whether a particular node has already been tagged as a solution. Returns True or False.
IsTaggedNotSolution(*node*)	Determines whether a particular node has already been tagged as a non-solution. Returns True or False.
TagSolutions(*node*)	This will tag *node* and all the higher nodes in the lattice along the path of the same generalization strategies as solutions.
TagNotSolutions(*node*)	This will tag *node* and all the lower nodes in the lattice along the path of the same generalization strategies as non-solutions.
Lattice(*bottom-node*, *top-node*)	Creates a lattice with a particular node at the bottom and another at the top.
Height(*lattice*, *node*)	This function returns the height of a particular node in the particular (sub-)lattice.
midLvl (*lattice*)	Returns the set of nodes located on the middle level of the lattice.
CleanUp(*node*)	Removes all nodes in the OSC set that are on the same generalization strategies as *node*, and are not local minima.

Table 11 Pseudo code for getting the optimal solution candidates (GetOCS)

```
Input: Two nodes that represent the bottom and top of a (sub)lattice
Output: Void
Global variables: S the set of OSC, k the anonymity level and MaxSup
the suppression limit (the last two are used in the "Satisfy(node)"
function


GetOSC(Bnode,Tnode)
{
      L=Lattice(Bnode,Tnode)
      $H_H$=Height(L,Tnode)
      If $H_H \neq$ 0 then
            middle = midLvl(L)
            Foreach V in middle
                  If IsTaggedSolution(V) == True then
                        GetOSC(Bnode, V)
                  Else if IsTaggedNotSolution(V) == True then
                        GetOSC(V,Tnode)
                  Else if Satisfy(V) == True then
                        TagSolutions(V)
                        GetOSC(Bnode,V)
                  Else
                        TagNotSolutions(V)
                        GetOSC(V,Tnode)
                  End If
            End For
      Else
            S = S + Tnode
            CleanUp(Tnode)
      End if
}
```

Whenever this function is called, it first checks if the Bnode and Tnode are not the same vector. If that is the case, it creates a lattice with Bnode as its base and Tnode as its top (which is initially the main lattice). Next, it identifies the mid level of that lattice. For every vector V at that level, it first checks if V has been tagged either as a solution or a non-solution. If it was not, it calls Satisfy(V) and processes the prediction with respect to the outcome of that function, by tagging the main lattice accordingly. Either way, if V was identified as a solution, it means that all the local minima of the strategies going through V are within the lattice between Bnode and V. However, if V was identified as a non-solution, it means that all the local minima of the strategies going through V are within the lattice between V and Tnode.

Therefore, it then recursively calls GetOSC with the right parameters according to the status of V. Again and again, it recursively goes through the same process for the sub-lattices until eventually the Bnode and Tnode are the same vector. At that point, it adds V to the set of OSC and then cleans it by removing all the vectors that are generalizations of other vectors in that set. The last step is needed because as stated earlier, knowing that many strategies interrelate, we should eliminate the local minima that are actually generalizations of other local minima and thus eventually find the set of OSC.

Whenever a recursive invocation ends, the process gets back to the next vector at the mid-level of the respective sub-lattice. At that point, it would have made enough predictions that the number of the expected calls of Satisfy(V) will decrease considerably and consequently so will the time to find the OSC.

Figure 11 to Figure 26 illustrate a trace of the algorithm on a plausible lattice. Some obvious steps have been omitted. Note that in these figures, the solutions are highlighted in dark-gray while the non-solutions are in light-gray. The local minima have a dotted line, and the nodes with bold lines are the ones on the midlevel of a (sub) lattice that are still to be processed. The non-transparent nodes are the nodes of the sub-lattice in consideration. The trace is explained in detail afterwards.

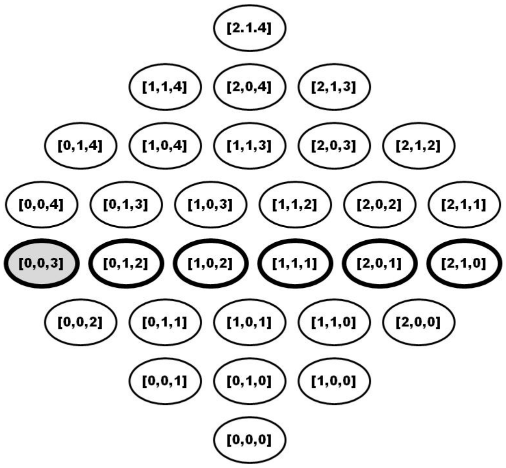

Figure 11 Getting OSC - step A

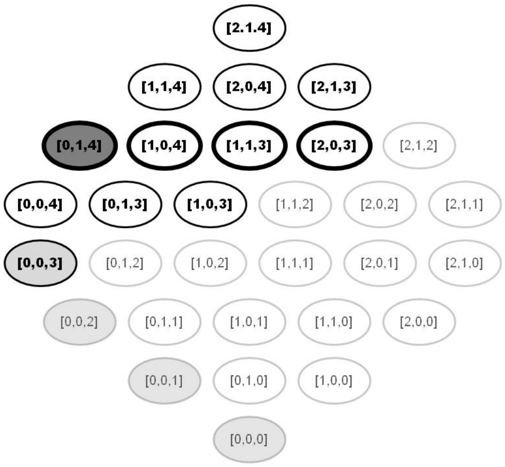

Figure 12 Getting OSC - step B

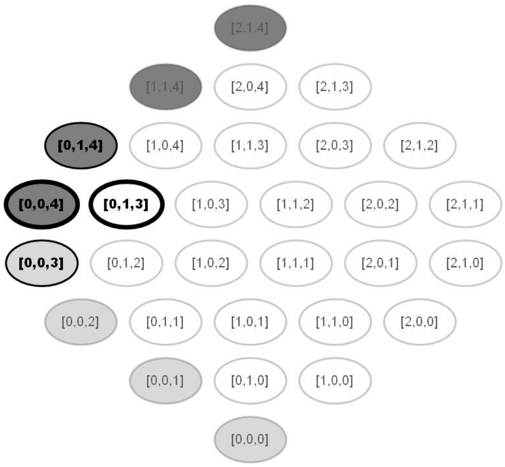

Figure 13 Getting OSC - step C

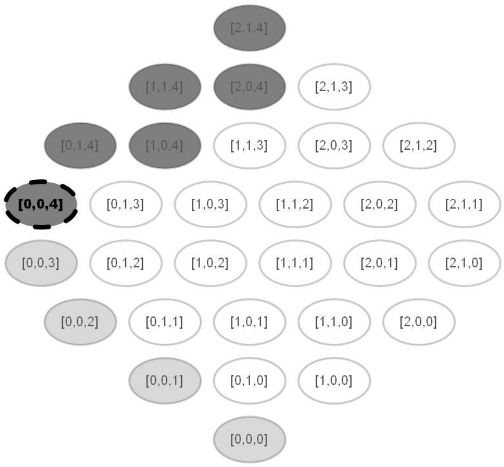

Figure 14 Getting OSC - step D

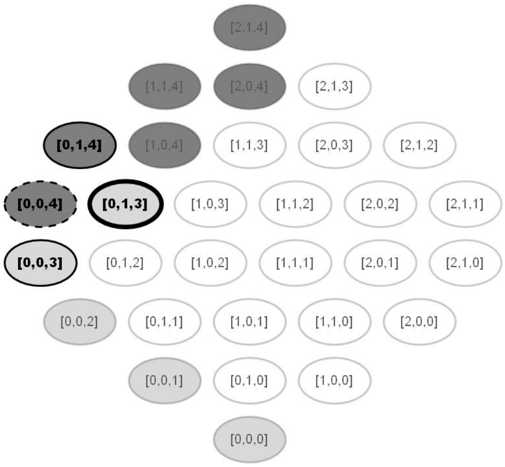

Figure 15 Getting OSC - step E

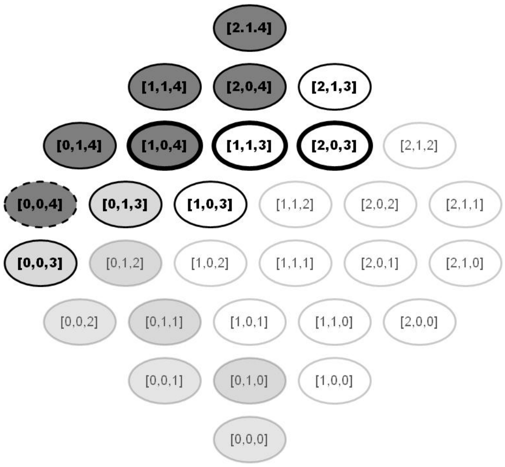

Figure 16 Getting OSC - step F

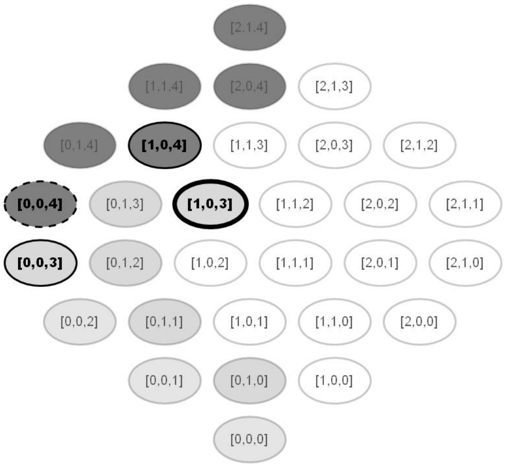

Figure 17 Getting OSC - step G

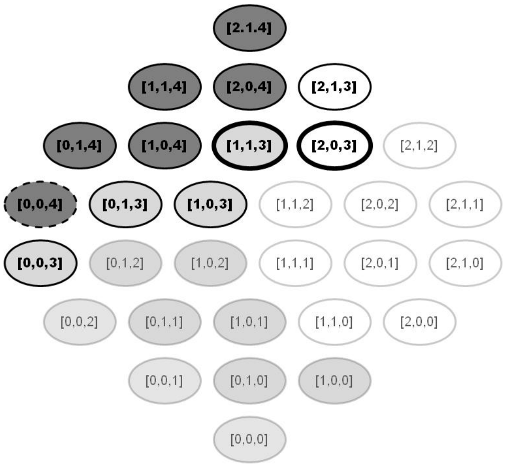

Figure 18 Getting OSC - step H

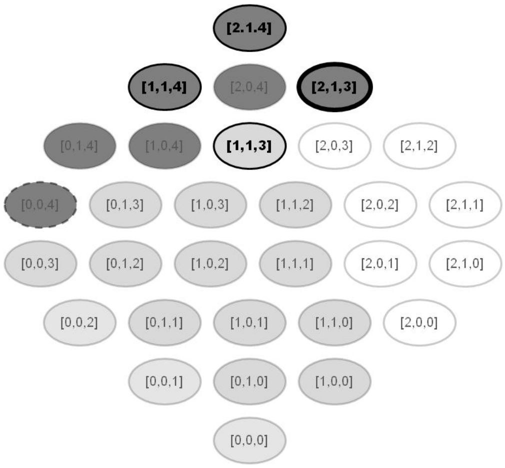

Figure 19 Getting OSC - step I

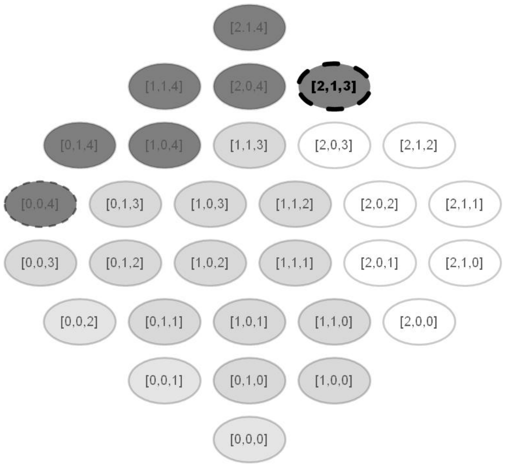

Figure 20 Getting OSC - step J

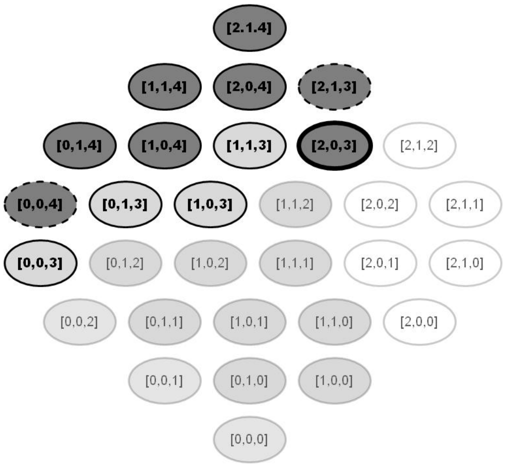

Figure 21 Getting OSC - step K

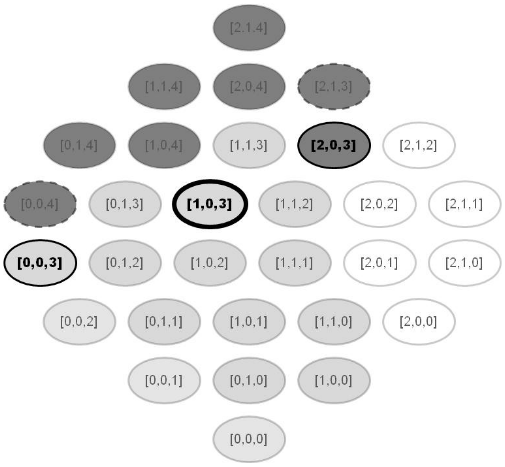

Figure 22 Getting OSC - step L

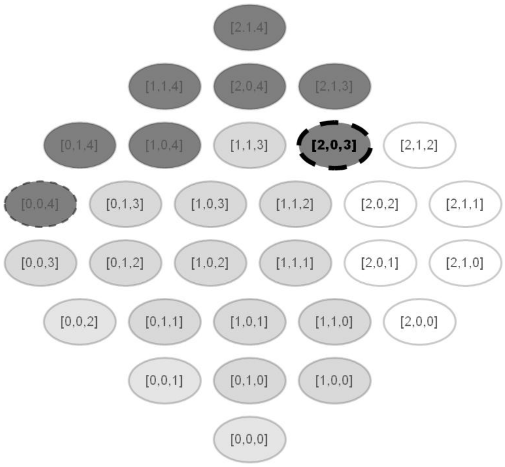

Figure 23 Getting OSC - step M

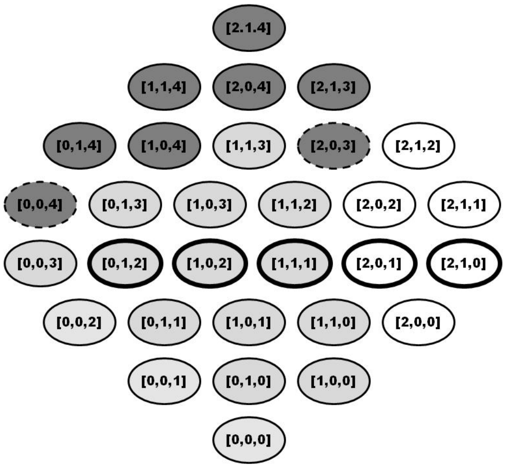

Figure 24 Getting OSC - step N

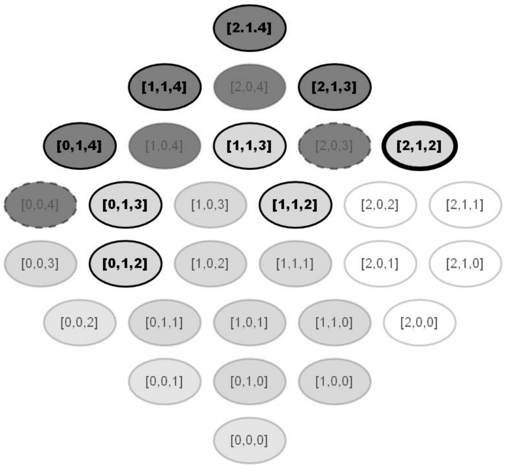

Figure 25 Getting OSC - step O

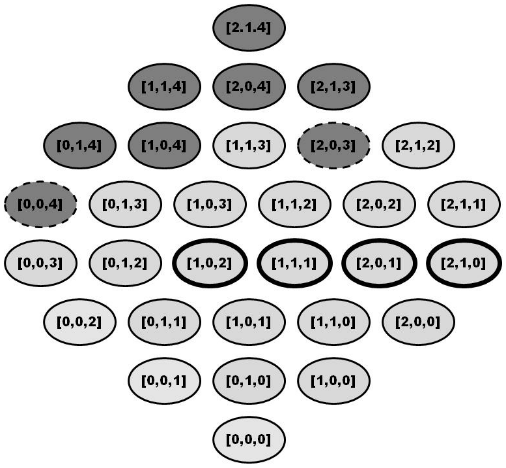

Figure 26 Getting OSC - step P

Here is a brief description of the above example. In step A, the midlevel of the whole lattice is taken into consideration, and the first vector on that level has been evaluated as a non-solution. As our algorithm suggests, the local minima going through that vector reside in the sub-lattice between itself and the top vector. Thus, the respective predictions are tagged, and recursively, the midlevel of that sub-lattice is now taken into consideration, while the first vector on that level has been evaluated as a solution as shown in step B.

At this point the highlighted sub-lattice in step C contains local minima, and again the first vector on the midlevel of that sub-lattice has been evaluated as a solution. Eventually, after going through the same process, this vector is identified and tagged as a local minimum (step D) and more predictions are tagged through the lattice.

Now, since this recursive invocation ended, we get back one level up to the former sub-lattice where the next vector on its mid-level ([0,1,3]) is evaluated as a non-solution (E). The process continues similarly without finding a local minimum. Thus, another invocation ends and we get back one more level up to the sub-lattice shown in step F where more predictions are tagged. At this point, the vector to be processed is already tagged as a solution and does not need to be evaluated. Accordingly, another sub-lattice is processed without finding a local minimum. The algorithm keeps on going through the lattice, tagging the predictions, but not finding another local minimum until step J. However, later on, we remove this local minimum from the OSC list since it is only a generalization of the one found in step M.

Notice that in step N all the secondary recursive invocations have ended and we got back to the initial lattice, where only one vector of its midlevel has been considered. This means that the remaining vectors on that level still have to go through the same process as above. Nevertheless, at this point, the majority of the lattice is already tagged and there are very few evaluations left that are now needed.

Finally, in step P, although there are many more strategies to be checked, all the nodes in the lattice have been already identified as solutions or non-solutions and we have already located the set of OSC.

In this particular trace we had to call the function Satisfy(V) 9 times in a lattice composed of 30 vectors. Therefore, we predicted 21 nodes. This is a total of 70% predictions and 30% checks, where 6% were the optimal solution candidates found.

4.5. Efficiency

In order to further highlight the efficiency of this approach, we used it on all the aforementioned data sets and captured the percentage of the number of evaluations (i.e. calls to Satisfy(V)), and the percentage of the size of OSC, with respect to the lattice size. Table 12 shows the size of the lattice for each data set.

Table 12 Lattice size of the data sets

	Adult	CUP	FARS	Pharm	ED	Niday
Lattice size	5184	360	120	2352	3584	7560

In Figure 27, we show the percentage of the number of nodes evaluated with respect to the total number of vectors in the lattice for k=2 to k=15 inclusively. These results are for a fixed suppression limit of 5%. Additional results for 1% and 10%, consistent with the ones shown here, can be found in appendix B (see Figure 63 and Figure 72).

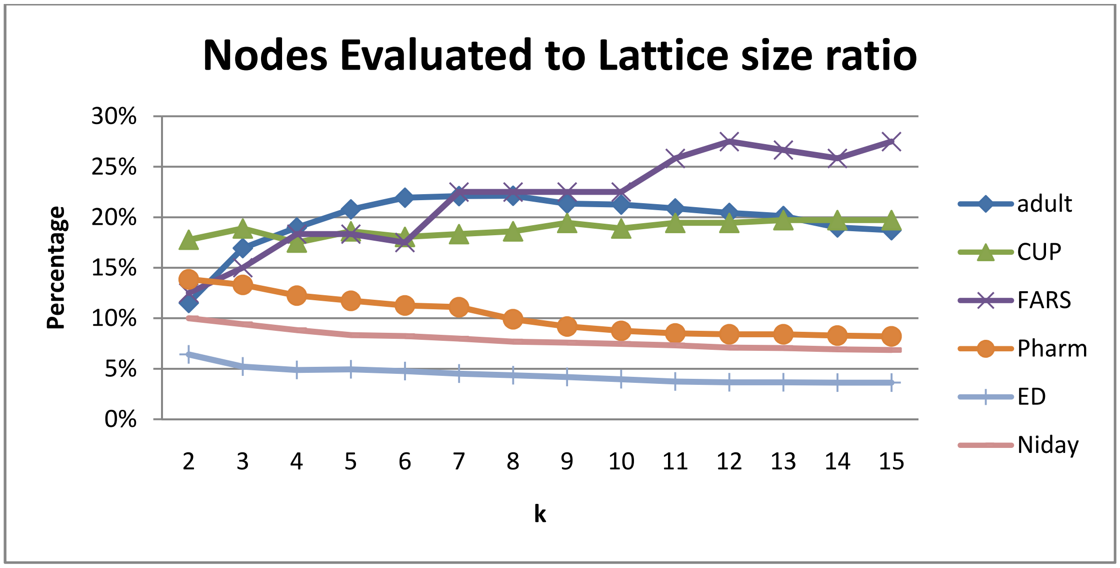

Figure 27 "Number of evaluations" to "lattice size" ratio

Notice that, when dealing with small lattices, the percentage of predictions tends to be relatively small. For CUP and FARS, the approach evaluated on average 20 to 25 percent of the lattice. For the data sets with larger lattices, e.g. Niday, ED and Pharm, with lattice sizes of 7560, 3584 and 2352 respectively, the approach evaluated on average between 4 and 14 percent of the lattice, therefore predicting about 90% of the nodes. It is important to emphasize that these three data sets are real clinical records and are real life examples of what such an approach will have to handle.

Figure 28 highlights the percentage of the size of OSC with respect to the total number of nodes in the lattice for k=2 to k=15 inclusively. It shows that the size of OSC is on average very small compared to the size of the lattice, and these results are consistent with results shown in Figure 27. Again, these are for a fixed suppression limit of 5%, while more results can be found in appendix B (see Figure 64 and Figure 73).

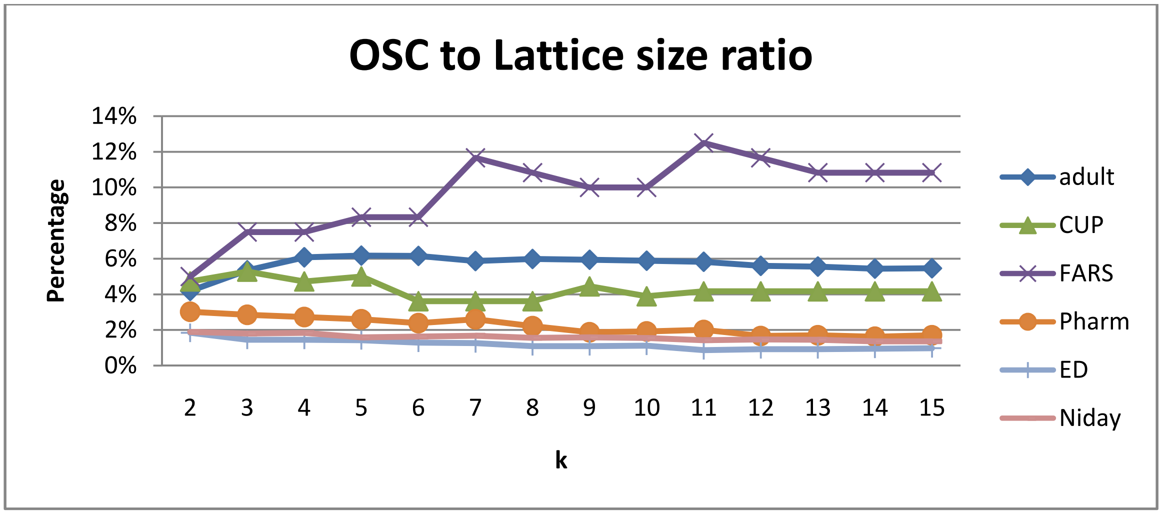

Figure 28 “OSC” to “lattice size” ratio

Since the size of the OSC is relatively small, the step (2) of our approach, which consists mainly of going through this list and checking for information loss, is considered to be straightforward and does not need much attention. The reason we show the percentage of OSC, is to highlight that as opposed to the set of all possible solutions, it is practical to go through this small number of nodes in order to find the optimal solution.

4.6. Summary

This new approach is made possible and pragmatic because of the prediction equations that correctly deduce the status of many vectors, and hence avoid unnecessary computations. Moreover, as opposed to some of the existing approaches, this method does not produce an approximation, but precisely locates the optimal solution. Even so, as a sanity check, we ran our algorithm versus the brute force method on all the aforementioned data sets, and the outputs of the two approaches were absolutely identical.

Furthermore, this approach produces output in a very acceptable time frame (more details about this in section 5.2), and the results in Chapter 6 will suggest that on average it is more efficient than its competitors.

Chapter 5. Optimizations

As stated earlier, the time required to evaluate whether the vectors are solutions or not represents almost all of the total computation time. Therefore, any algorithm that aims to find the optimal solution, which requires the evaluation of a relatively high number of vectors, has to deal with the complexity of this process and make it as efficient as possible.

Consider a straightforward implementation of Satisfy(V) that goes as follows:

1- Generalize the table with respect to V

2- Find the equivalence classes

3- Check if we need to suppress more than MaxSup so that all the remaining equivalence classes have a size >= k

We acknowledge the fact that a better approach than the above might exist. Nevertheless, the optimizations cited below are general programming shortcuts and enhancements, and are not restricted to this method.

5.1. Four Main Optimizations

We tackled the complexity of the process mentioned above through many programming optimizations, out of which we cite four. All of these are somewhat interrelated, while some are strictly dependent on the others.

5.1.1 A: Numbers versus Strings

Comparing strings is always more expensive than comparing integers. This is a very obvious observation.

Therefore, a clean way to make the data strictly consisting of integers is to locate and sort the unique items of every attribute, and then hash each item with respect to its location in its corresponding array. Consequently, this encodes the data set into a matrix of numbers.

Although this is very straightforward, the following example is needed to illustrate further optimizations.

Table 13 Race - unique items

index	Race
0	Asian
1	Black
2	White

Table 14 Marital status - unique items

index	Marital Status
0	Divorced
1	Married
2	Single
3	Widow

Table 15 Original data

Race	Marital Status
asian	married
asian	single
black	married
white	widow
white	divorced
black	single
asian	married
...	...

Table 16 Transformed data

Race	Marital Status
0	1
0	2
1	1
2	3
2	0
1	2
0	1
...	...

At this point, and as shown in Table 16, the data now consists of only integers, and any comparison of the data afterwards will be less expensive.

For example, finding the equivalence classes for the Adult data set (which has 30162 rows), under our implementation, used to take 22 seconds before; now, this takes only 6 seconds. This represents almost a four times speedup for this data set.

5.1.2 B: Flat hierarchies

The hierarchies are usually in a tree-like form such as in Figure 1 and Figure 2 back in chapter 2. When an item in a tuple needs to be generalized, one has to search the leaves of the related hierarchy tree for that item, generalize it as needed by going up the hierarchy, and then return the corresponding value. In a worst-case scenario, this process can have a time complexity of O(n), where n is the number of leaves in the hierarchy. This does not seem very significant in this example since the trees are relatively small. However, when having an attribute with thousands of unique items, and hence a huge hierarchy tree, and when a very large data set is being processed, it becomes critical to have a more efficient way for generalization, such as the one introduced below.

First, we hash the hierarchies with regard to the corresponding hashed values of the unique items. Figure 29 and Figure 30 are the hashed versions of the hierarchies shown in Figure 1 and Figure 2 respectively. Moreover, leaves in both hierarchies are hashed with respect to the hashed unique items in Table 13 and Table 14.

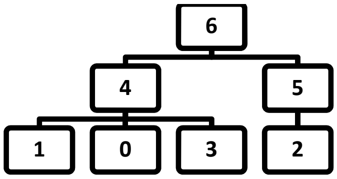

Figure 29 Hashed GH for Marital Status.

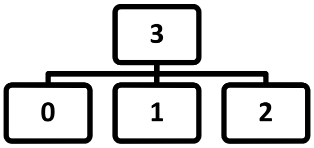

Figure 30 Hashed GH for Race.

Table 17 GH Array – Race – (GHR)

index	G0	G1
0	0	3
1	1	3
2	2	3

Table 18 GH Array – Marital Status – (GHM)

index	G0	G1	G2
0	0	4	6
1	1	4	6
2	2	5	6
3	3	4	6

As shown in Table 17 and Table 18, the hierarchies are then extracted into a two-dimensional array. That way, after optimization A has been applied, the number found in a cell in the data set, and the number of generalizations required, is the index of the new value.

For example, if the second tuple {0,2} in Table 16 above needs to be generalized once on the marital status attribute, the new value would be GHM[2][1] = 5.

Therefore, the time complexity for any generalization is now O(1), regardless of the data set or its hierarchies.

5.1.3 C: Sorting always helps

Finding the equivalence classes is the essence of the efficiency issue we face when trying to evaluate whether a vector is a solution or not. This is the procedure that consumes the majority of the computation time.

In order to find the equivalence classes, a first intuitive approach would look like this: For each row in the data set, if we did not identify this row as an equivalence class yet, then add it to the equivalence classes array, else update the size of this equivalence class.

However, since for each and every row, we always need to check if it is contained in the equivalence classes array, then the time complexity is $O(n^2)$. In addition, we are comparing full arrays that represent the rows, which is an expensive procedure by itself.

Therefore, we rather propose the following: For each row in the data set, hash the row into a string, then SortInsert the hash value in an array. Afterwards, simply go the through the resulting array and count the equivalence classes and their frequencies.

For example, the rows in Table 19 have been hashed into strings, which enable a perfect hash function, and sorted as shown in Table 20. Notice that at this point, we only need to go through the table once and count the sizes of the equivalence classes which can be clearly distinguished without the need for any kind of search.

Under this scheme, finding the equivalence classes will have the complexity O(n log n). Moreover, at this point comparing strings is less expensive than comparing a full row as before.

Table 19 Original data

Race	Marital Status	Age
1	2	2
1	3	0
1	2	1
1	2	2
1	2	2
0	0	2
1	2	0
0	0	2
1	2	1

Table 20 The same data hashed and sorted

Hashed and sorted results
“0-0-2”
“0-0-2”
“1-2-0”
“1-2-1”
“1-2-1”
“1-2-2”
“1-2-2”
“1-2-2”
“1-3-0”

Furthermore, using this approach, we will not always need to go through the whole table in order to figure out if a data set satisfies k-anonymity with minimal suppression. For example, in Table 20, if k = 3 and MaxSup = 4, the first three ECs have a frequency less than k, and thus they need to be suppressed if this data set is to satisfy k-anonymity. However, at this point there is already a suppression of 5 rows, which is greater than MaxSup. Therefore, there is no need to continue the verification, and we can declare this as a non-solution right away.

For the Adult data set, under our implementation, when both A and C are applied, it takes only 0.3 seconds to find the EC rather than the original 22 seconds, a 73x speedup.

5.1.4 D: Rollup

Another shortcut is to compute Satisfy(V) on a frequency set, rather than to compute it on the whole data set.

If we only calculate the frequency set of the original data set and evaluate all other nodes that need to be evaluated, with respect to that set, the speedup will be the ratio of the number of rows in the original data set to the number of ECs in the frequency set. For example, the Adult data set has 30,162 row and 12,005 equivalence classes. Generalizing 12,005 rows and finding the corresponding ECs while adding up their frequencies, is much faster than doing so on the whole data set. The speedup is then around 3x.

Since we traverse the lattice in a seemingly random manner as opposed to a breadth-first bottom-up search, we cannot take advantage of this shortcut to the fullest extent by always using the frequency set of the parents of the current node that is being evaluated. However, we can still benefit a lot from this shortcut by always using the frequency set of the closest node in the same strategy. That is, whenever we evaluate a node in a strategy and we find that it is not a solution, we save the frequency sets of this node. Then, since we have to go up the lattice, all the nodes that need to be evaluated herein can use the closest saved frequency set.

Eventually, the number of rows that need to be processed in order to find the overall equivalence classes within all the evaluated nodes diminishes dramatically. Thus, as shown in the next section, since this is a crucial criterion that defines the efficiency of the process, the speedup tends to be significant.

5.2. Overall Time Complexity

We incorporated the above optimizations, and then processed our approach on the aforementioned data sets on a 3.2 GHz Pentium D processor with 3GB of RAM.

The execution times for the proposed algorithm with MaxSup = 5% are shown in Figure 31 below. Considering the large sizes of the data sets, and the sizes of the lattices with regards to the proposed QIs, the execution time can be considered practical. Note that the time registered in this chart is the time to find the set OSC and then go through all the nodes in this set and find the optimal solution with regard to an information loss metric.

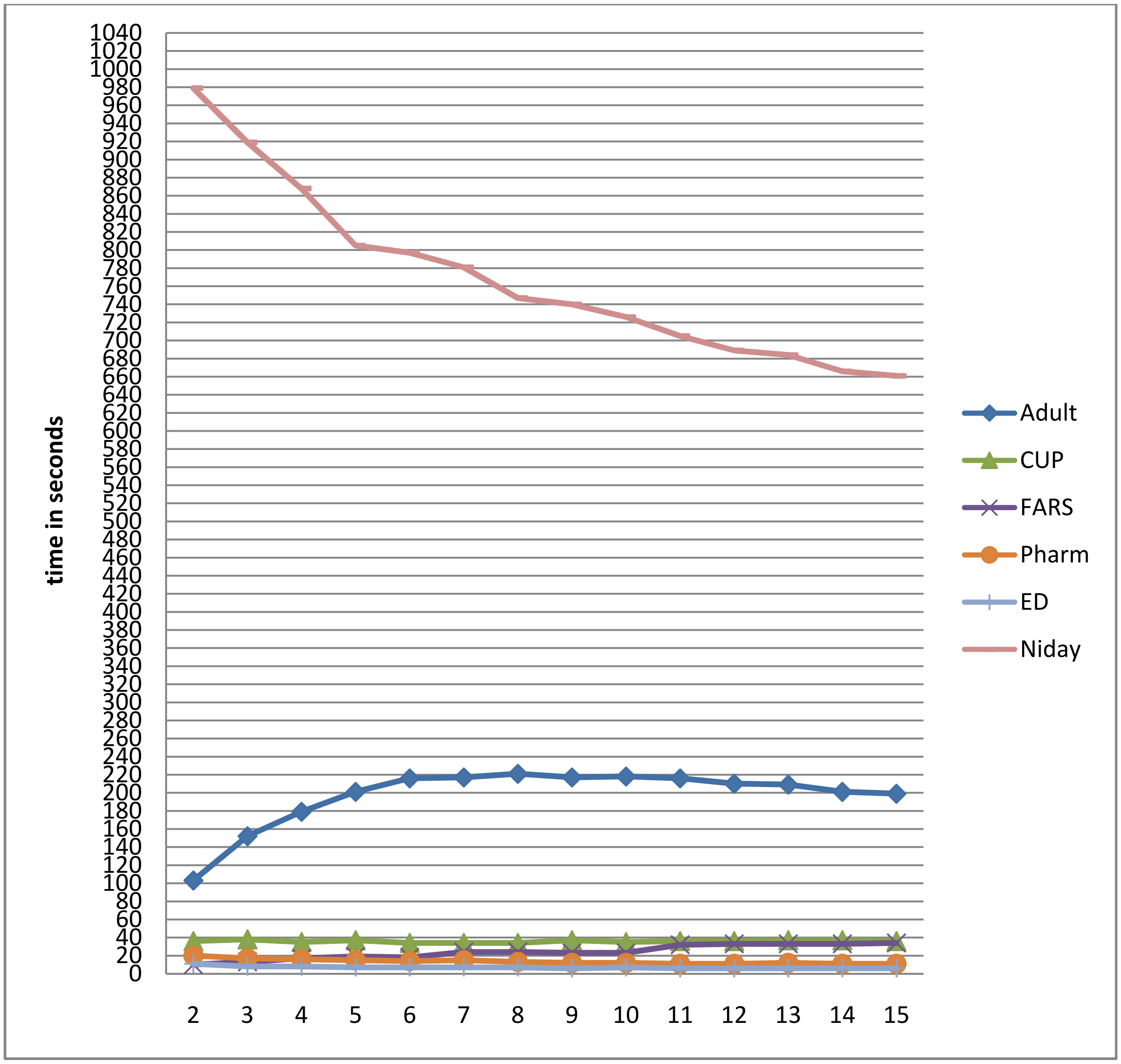

Figure 31 Execution time in seconds. Suppression limit of 5%.

Notice that even for large data sets such as FARS (101,030 rows), the computation time was below 40 seconds. The worst case was the Niday data set (124,933 rows) with a computation time of, on average, around 12 minutes. This was due to the relatively large lattice corresponding to the numbers of QIs we chose for this data set, which required the evaluation of a larger number of vectors before finding the OSC.

Comparing the graph in Figure 31 to the one in Figure 27, we notice that the curves are very much consistent. The most noticeable ones are the Adult and Niday curves. In both graphs the shape of these curves is similar, which means that the higher the percentage of nodes being evaluated, the more time is taken by the algorithm to execute. If the graph in Figure 27 had plotted the exact number of vectors evaluated instead of the percentage of vectors with respect to the lattice size, the two curves would have looked exactly the same.

Eventually, the execution time depends mainly on the number of vectors in the lattice that the algorithm needs to evaluate. Consequently, since we use the optimization D above, the time complexity will depend on the overall number of rows processed.

To further discuss this, here is again the implementation of Satisfy(V) that was mentioned in the beginning of this chapter:

1- Generalize the table with respect to V
2- Find the equivalence classes
3- Check if we need to suppress more than MaxSup so that all the equivalence classes will have a size >= k

Based on this implementation, the time to evaluate a single vector is the sum of the times of every step:

1- As shown in step B, any generalization is of complexity O(1), and the time complexity of generalization with respect to a vector is O(n * |QIg|), where n is the number of rows in the current data set (or frequency set) and |QIg| is the total number of QI that were generalized. Hence, n * |QIg| is the number of generalized cells. In a worst-case scenario, we would need to generalize all the cells in the data set.
2- As shown in optimization C, the complexity of finding the equivalence classes is O(n log n).
3- Again in C we show that in a worst-case scenario we need to pass once through the resulting sorted array to determine whether it satisfies k-anonymity with minimal suppression or not. Therefore, its complexity is O(n).

Therefore, the complexity of the evaluation of one node in the lattice would be:

$$O(n \log n)$$

where n is the number of rows in the current data set (or frequency set[5]). |QIg|. the total number of QI that were generalized, is not taken into consideration in the complexity measure since the dominant part in this scheme is finding the equivalence classes.

Hence, the total complexity to find the set of OSC, after evaluating a certain number of nodes, is roughly:

$$O(logN)$$

where $N = \prod_i (n_i^{n_i})$, that is, the product of the number of rows processed for every node evaluated in the lattice, to the power itself . For example, if we evaluated two nodes in the lattice where the number of rows in the first is X and in the second is Y, then their complexity becomes:

$$XlogX + YlogY = logX^X + logY^Y = log(X^X * Y^Y)$$

The remaining execution time results for the 1% and 10% suppression limits can be found in Appendix B (see Figure 65 and Figure 74), which also show a consistent execution time for all data sets.

5.3. Summary

In this chapter, we highlighted a number of programming optimisations used for implementing our approach, which give the algorithm a substantial speed improvement. It was also shown that this algorithm finds the optimal solution in a practical time with respect to the size of the data sets and the size of the lattices.

Moreover, we identified the time complexity of finding the OSC and more precisely the complexity of evaluating whether a vector satisfies k-anonymity with minimal suppression. This complexity analysis will be useful for comparing the efficiency of our approach with other existing approaches.

[5] When taking advantage of the rollup optimization, we generalize over an already generalized dataset that has been transformed into a frequency set. Therefore, the number of rows evaluated for a specific node in the lattice would be the number of EC it has at that point.

Chapter 6. Efficiency

LeFevre, DeWitt and Ramakrishnan [23] proposed an efficient algorithm for computing k-minimal generalization, called Incognito, which takes advantage of a bottom-up aggregation along dimensional hierarchies and a priori aggregate computation. However, a major drawback of this approach is that it returns the set of all possible solutions in the lattice, and thus it is impractical to check the information loss of all of them in order to find the optimal one.

Regardless, the method used to retrieve this set of all solutions is interesting and proven to be efficient. Incognito was compared to several algorithms, including Samarati's, and was shown to be the fastest in locating the corresponding set of solutions [23]. Moreover, one can argue that Incognito can be altered to return the same set of OSC as our approach, and thus it would be interesting to compare this approach to our algorithm in order to identify which of these two is more efficient.

6.1. Incognito

The Incognito algorithm generates the set of all possible solutions, i.e. all the generalization vectors that satisfy k-anonymity with minimal suppression.

Based on the subset property[6], the algorithm begins by checking single-attribute subsets of the quasi-identifiers, and then iterates for i = 0 to |QI|, checking k-anonymity with respect to increasingly large subsets. Each iteration consists of two main parts:

1- Every iteration considers all the nodes in a set S constructed from subsets of the quasi-identifiers of size i. Then it goes through these nodes in a breadth-first bottom up search taking advantage of the rollup phenomenon presented above in section 5.1.4, and of the generalization property stated in equation (b) in section 4.2.1.

[6] Let T be a dataset, and let Q be a set of attributes in T. If T is k-anonymous with respect to Q, then T is also k-anonymous with respect to any set of attributes P such that $P \subseteq Q$ [23].

2- The algorithm then constructs the set of candidate nodes S with quasi-identifiers of size i + 1, taking advantage of the subset property by pruning the nodes that cannot be solutions when the set of attributes is larger.

The authors mention three versions of this approach, each one applying different programming optimizations to the base approach. Then they show that the "Super Root Incognito" is the fastest of all three. This is the version where they simply take more advantage of the rollup property by computing the frequency sets of the parent of the candidate nodes in S after each iteration, and then use this set as a base to evaluate the nodes and before continuing in (1) normally.

In what follows, we will be comparing our approach to the "Supper Root Incognito". However, for convenience, we will refer to this approach simply as Incognito.

6.2. Comparison

We do not compare our algorithm with Incognito in terms of "seconds to produce a result" for two reasons. First, the exact timing will be dependent on the implementation details, and second, such a comparison would be perceived as inherently biased because we would be expected to put more effort optimizing the implementation of our algorithm. Therefore, we measure elements that are inherent to the algorithms rather than their implementations. Note however that the actual time comparison was very much consistent with the following results.

As argued earlier, the most time-consuming activity in all of the aforementioned algorithms is evaluating a node in the lattice to determine whether it is a solution or not. An obvious way to compare the algorithms is to count the number of nodes that need to be evaluated. Figure 32 shows the original search space of Incognito with respect to the original search space of our approach (the lattice). Similar to the charts in the comparisons of chapter 3, the charts herein are the ratio of Incognito's output to the output of our approach. Moreover, the following graphs (Figure 33 to Figure 35) are for MaxSup = 5%, and additional consistent graphs for MaxSup = 1% and 10% can be found in Appendix B.

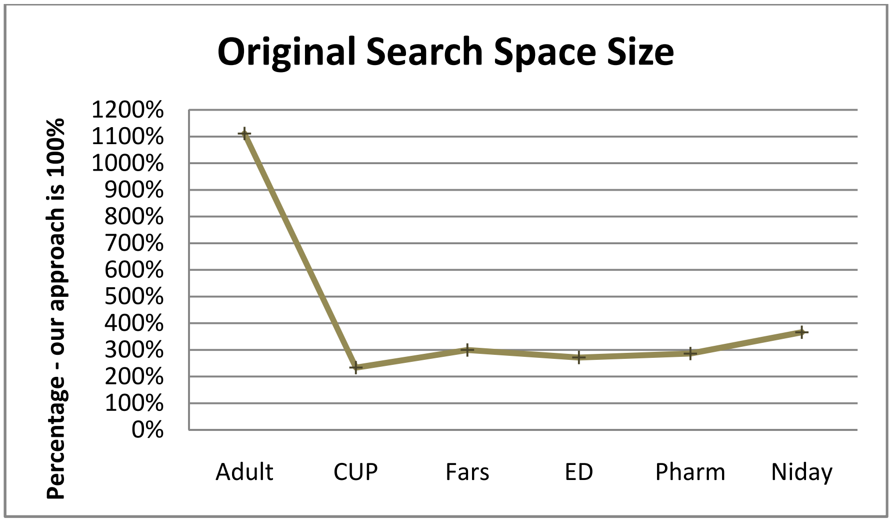

Figure 32 Original search space size.

We see that the search space considered by Incognito is always larger, which is normal since it conceptually generates all the lattices for all the subsets of QI for i = 1 to |QI|. This is not a performance measure per se since Incognito will eventually prune a very large number of nodes, but it is important to see the difference in the search space size, which mainly explains the difference in the number of nodes evaluated.

Figure 33 shows the ratio of nodes evaluated. We see that on average Incognito always evaluates many more nodes than our approach.

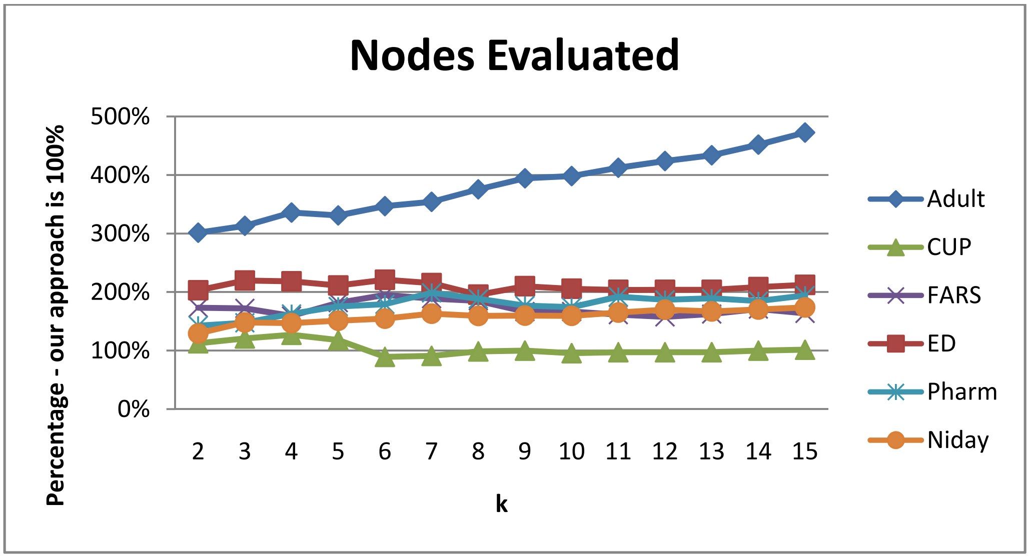

Figure 33 Nodes evaluated ratio.

However, the complexity of nodes evaluation is not the same because the nodes vary in terms of the number of records being evaluated (as a result of the different frequency set attained by the rollup property) and the number of QI being generalized. The number of nodes evaluated is not sufficient to capture the efficiency difference between the two approaches.

Therefore, since both algorithms can use the same function in order to evaluate a node, we make use of the complexity measure of the Satisfy(V) function introduced in section 5.2 above.

The different number of QI in Incognito is strictly related to the frequency set produced in the dataset. The smaller the number of QIs, the fewer ECs will be produced, and thus the number of rows considered. Therefore, this is taken into account, along with the rollup optimization that Incognito takes advantage of, by counting the overall number of rows *N* as stated earlier.

We can then compare the performance of our algorithm and Incognito's by computing this score across all evaluated nodes. Note that *N* will be different for both algorithms since they evaluate a different number of nodes, as shown in Figure 33, and the number of equivalence classes on each of these nodes varies.

One can argue that a different implementation of Satisfy(V) can be more efficient, however, the same implementation can always be used on both algorithms and the results will be similar.

The performance comparison with Incognito over all the aforementioned data sets is shown in Figure 34. Incognito performed better on the CUP data set across all values of k. Otherwise, the performance is more or less the same for the Pharm and Niday data set, and our algorithm performs better for the remaining three data sets. In the case of the Adult data set and FARS, the difference in performance is significant.

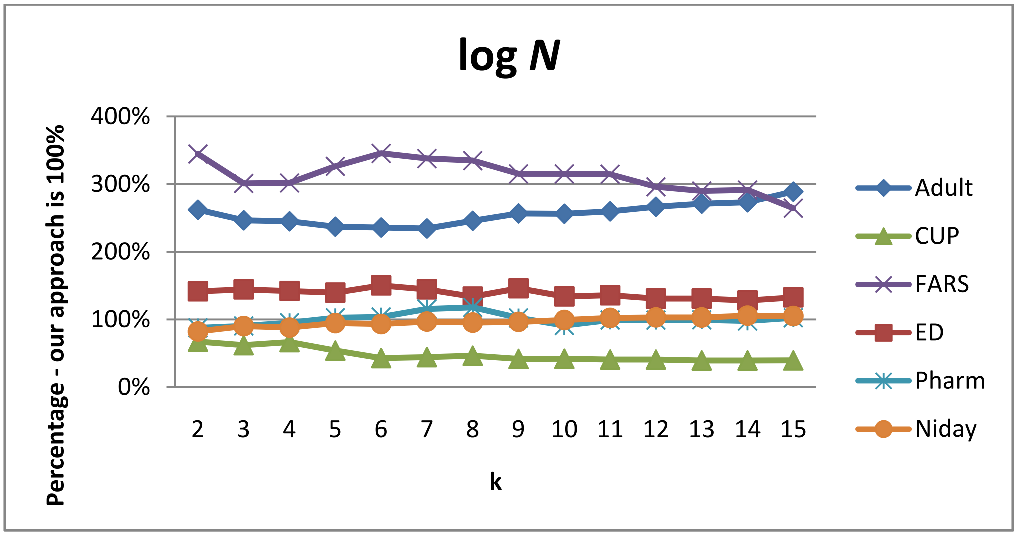

Figure 34 Performance score of Incognito with respect to our approach.

Note that this performance comparison is only for the process of getting the respective sets of solutions, in our case OSC and in Incognito's case the set of all possible k-anonymous solutions. Figure 35 shows that the size of the set of solutions that Incognito returns, and that need to be checked for information loss in order to get the optimal solution, is at least 5 times larger than the set of OSC located by our approach. This means that our approach will be at least 5 times faster than Incognito in this phase of the process.

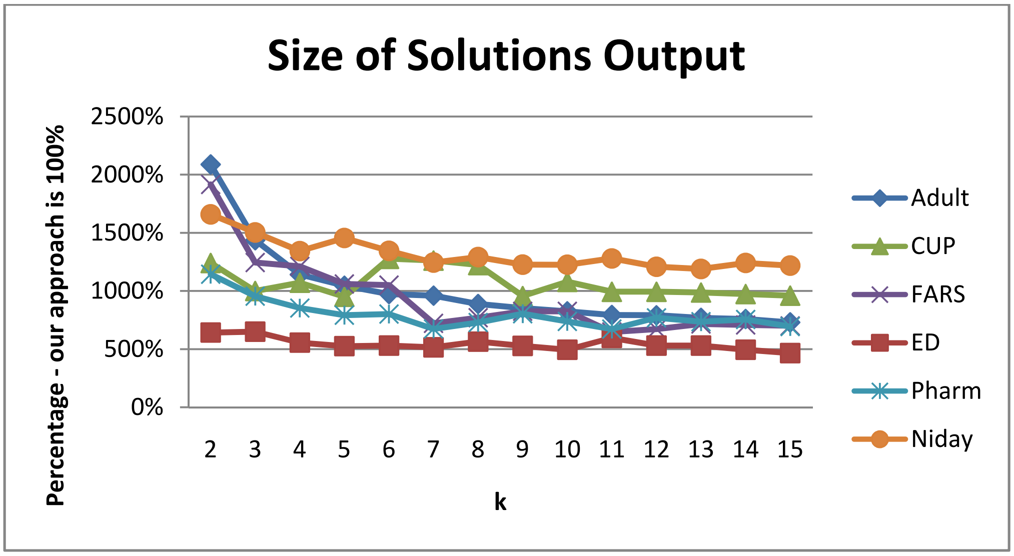

Figure 35 Size of solutions output by Incognito.

However, as argued earlier, Incognito can probably be altered to find the set of OSC instead, and then both algorithms will perform similarly in the second step (the process of finding the optimal solution). Also note that since in our approach we tag all the possible k-anonymous solutions, we can return the set same as Incognito.

6.3. Summary

In this chapter we compared the performance of Incognito to our approach. Only the first step, which consists of finding the respective set of solutions, was considered. The empirical results showed that our approach outperformed Incognito in many cases. However, in some cases both algorithms perform the same more or less, and in rare cases Incognito outperforms our approach.

Then we show that the number of solutions returned by Incognito is very high and is at least 5 times more than the set of OSC. This is a major drawback of the current setup of Incognito since all the vectors in that set need to be evaluated from an information loss perspective and then the one with the least information loss must be returned. Therefore, in this phase of the process, our approach will outperform Incognito. It is worth to note that Datafly is faster than both our approach and Incognito. However, in the context of clinical data, the trade-off between speed and optimality is not possible since researchers needs to work on an anonymised data set with minimal information loss.

Chapter 7. Conclusions

7.1. Contributions

In this thesis, we presented the results of an empirical evaluation of Datafly and Samarati's algorithms versus a brute force approach that extracts the optimal solution with respect to a given metric. The comparison in chapter 3 showed that the performance of Datafly and Samarati did vary significantly depending on the data set, the value of k, and the suppression limit. However, on average Samarati's solution was better than Datafly's. Moreover, it was clear that these approaches give on average a result of poor quality when compared to the one provided by the brute force method.

Therefore, after showing the need to find an optimal solution rather than a mere approximation, we proposed our own approach, a new method to efficiently find an optimal solution with regards to any given information loss metric that is monotonic. In the literature this property is argued to be important in order to produce generalizations that are most practical for research and data mining. Afterward, we highlighted some programming optimizations that were used in our implementation, and showed that our approach executes in a satisfactory time. Finally, we compared the performance of our approach to Incognito and concluded that in many cases our approach outperforms Incognito, especially in the current setup where the latter returns the set of all possible k-anonymous solutions in order to locate the optimal one.

One limitation with our study is that we compared our proposed algorithm with only three other approaches. However, we contend that these are the most suitable for the de-identification of clinical data because they work with hierarchical variables and use global recoding. Furthermore, they are foundational k-anonymity algorithms that are often cited in the literature.

Another limitation might be the assumption of monotonic information loss metrics. However, and stated earlier, this is a very important feature that needs to be present in a metric in order to assure better solutions. Moreover, the information loss metrics we used did not take into account different variable weights. We assumed that all quasi-identifiers were equally important. However, this should not change the outcome of the comparisons.

Nevertheless, regardless of the above limitations, the proposed approach appears to be sound and practical. Moreover, as opposed to the heuristic-based approaches, by insuring an optimal solution that can be located efficiently, researchers will benefit immensely, since the better the quality of the anonymized data, the more valuable that data is for the research.

7.2. Future work

The future work we foresee is mainly twofold:

1- The subset property that Incognito uses is very interesting. However, one main drawback of the way Incognito goes through the search space is the breadth-first bottom up search. In the case that the solutions are high in the lattice, and when not much pruning was introduced to that same lattice, it will have to evaluate a very large number of nodes. Even with the rollup advantage, the complexity of these nodes adds up to an inefficient result. Therefore, we might consider the case of using the subset property and our approach and dropping the breadth-first search. It would be interesting to see the outcome efficiency of the resulting algorithm.

2- Since there is not one globally accepted information loss metric, and since many current metrics capture different logical aspects on how the information is being changed and how much information we are approximately losing, we see a good chance of proposing a new metric that mainly joins these aspects and provides a better approximation.

References

[1] Adam, N. R. and Wortman, J. C. (1989) Security-Control Methods for Statistical Databases: A Comparative Study*, ACM Computing Surveys*, vol. 21, no. 4.

[2] Aggarwal, G., Feder, T., Kenthapadi, K., Motwani, R., Panigrahy. R., Thomas, D., and Zhu, A. (2005) Approximation algorithms for k-anonymity. *Journal of Privacy* Technology, 1–18.

[3] Bang-Jensen, J., Gutin, G. and Yeo, A. (2004) When the greedy algorithm fails. *Discrete Optimization* 1, 121–127.

[4] Bayardo, B. and Agrawal, R. (2005) Data privacy through optimal k-anonymity. In *Proc. of the 21st Int'l Conference on Data Engineering.* IEEE CS, 217–228.

[5] Canadian Institutes of Health Research (2004). *Guidelines for protecting privacy and confidentiality in the design, conduct and evaluation of health research.*

[6] Canadian Institutes of Health Research (2005) *CIHR best practices for protecting privacy in health research*.

[7] Ciriani, V., De Capitani di Vimercati, S., Foresti, S., and Samarati, P. (2007) k-Anonymity. *Secure Data Management in Decentralized Systems. Advances in Information Security*, Vol. 33*,* Springer*,* 323–353.

[8] Cox, L. H. (1980) Suppression methodology and statistical disclosure analysis. *Journal of the American Statistical Association*, 377–385.

[9] Dalenius, T. (1986) Finding a needle in a haystack - or identifying anonymous census record. *Journal of Official Statistics*, 329–336.

[10] Domingo-Ferrer, J.T.V. (2003) Risk Assessment in Statistical Microdata Protection via Advanced Record Linkage*. Journal of Statistics and Computing, 13(4).*

[11] Du, Y., Xia, T., Tao, Y., Zhang, D., and Zhu, F. (2007) On Multidimensional k-Anonymity with Local Recoding Generalization. *IEEE 23rd International Conference on Data Engineering*, IEEE CS, 1422–1424.

[12] Duncan, G., Jabine, T., and de Wolf, S. (1993) Private Lives and Public Policies: Confidentiality and Accessibility of Government Statistics. *National Academies Press.*

[13] El Emam, K., Brown, A., and Abdelmalik, P. (2008) Evaluating Predictors of Geographic Area Population Size Cutoffs to Manage Re-identification Risk. *Journal of the American Medical Informatics Association* (accepted).

[14] El Emam, K., Jabbouri, S., Sams, S., Drouet, Y., and Power, M. (2006) Evaluating common deidentification heuristics for personal health information. *Journal of Medical Internet Research;* 8(4):e28.

[15] El Emam, K., Jonker, E., Sams, S., Neri, E., Neisa, A., Gao, T., and Chowdhury, S. (2007) *Pan-Canadian De-Identification Guidelines for Personal Health Information*. Report prepared for the Office of the Privacy Commissioner of Canada. Available from:
http://www.ehealthinformation.ca/documents/OPCReportv11.pdf. Archived at: http://www.webcitation.org/5Ow1Nko5C. (last access: January 2009)

[16] Federal Committee on Statistical Methodology. (2005) *Report on statistical disclosure limitation methodology*. 2005; Office of Management and Budget.

[17] Gionis, A. and Tassa, T. (2007) k-anonymization with minimal loss of information. Algorithms – ESA 2007, LNCS 4698, Springer, 439–450.

[18] Greedy Algorithm, *available online at*
http://en.wikipedia.org/wiki/Greedy_algorithm (last access: December 2008)

[19] Health Insurance Portability and Accountability Act, *Available online at*
http://www.hhs.gov/ocr/hipaa, (last access: December 2008).

[20] Iyengar, V. (2002) Transforming Data to Satisfy Privacy Constraints. *Eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, Edmonton, Canada, ACM, 279–288.

[21] Katirai, H. (2006) *A Theory and Toolkit for the Mathematics of Privacy: Methods for Anonymizing Data while Minimizing Information Loss.* M.Sc. thesis, Dept. of Electrical Engineering and Computer Science, MIT, USA.

[22] Kim, J. and Curry, J. (1977) The treatment of missing data in multivariate analysis. *Social Methods & Research; 6:215-240.*

[23] LeFevre, K., DeWitt, D.J., and Ramakrishnan, R. (2005) Incognito: Efficient Full-domain k-anonymity. *Proc. ACM Management of Data, Baltimore*, Maryland, USA, ACM, 49–60.

[24] Li, T. and Li, N. (2006) Optimal k-anonymity with flexible generalization schemes through bottom-up searching. *Sixth IEEE Int. Conf. on Data Mining Workshops*, Hong Kong, China, IEEE CS, 518–523.

[25] Little, R. and Rubin, D. (1987) Statistical Analysis With Missing Data. *John Wiley & Sons.*

[26] Meyerson, A. and Williams, R. (2004) On the complexity of optimal k-anonymity. *Proc.of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on the Principles of Database Systems*, Paris, France, ACM, 223–228.

[27] Ochoa, S., Rasmussen, J., Robson, C., and Salib, M. (2001) *Reidentification of individuals in Chicago's homicide database: A technical and legal study*. Massachusetts Institute of Technology.

[28] Office of the Information and Privacy Commissioner of British Columbia (1998)*, Order No. 261-1998*. Available at:
[http://www.mser.gov.bc.ca/privacyaccess/Summ_IA_Order/Orders/order_261.htm] (last access: January 2009)

[29] Office of the Information and Privacy Commissioner of Ontario *(1994) Order P-644.* Available at: [http://www.ipc.on.ca/images/Findings/Attached_PDF/P-644.pdf].

[30] *Personal Health Information Protection Act*, available online at http://www.e-laws.gov.on.ca/html/statutes/english/elaws_statutes_04p03_e.htm (last access: January 2009)

[31] Samarati, P. (2001) Protecting Respondents' Identity in Microdata Release. *IEEE Transactions on Knowledge and Data Engineering,* 13 (6), 1010–1027.

[32] Statistics Canada (2007) *Therapeutic abortion survey*. *Available from:* [http://www.statcan.ca/cgi-bin/imdb/p2SV.pl?Function=getSurvey&SDDS=3209&lang=en&db=IMDB&dbg=f&adm=8&dis=2#b9]. Archived at: [http://www.webcitation.org/5VkcHLeQw].

[33] Subcommittee on Disclosure Limitation Methodology - Federal Committee on Statistical Methodology (1994) *Working paper 22: Report on statistical disclosure control.* Office of Management and Budget, USA.

[34] Sweeney, L. (1997) Guaranteeing Anonymity When Sharing Medical Data, The Datafly System. *Proc. AMIA Annual Fall Symposium*, 1997:51-5.

[35] Sweeney, L. (1997) *Computational Disclosure Control for Medical Microdata: The Datafly System. Record Linkage Techniques.* National Academy Press.

[36] Sweeney, L. (2000) *Uniqueness of Simple Demographics in the US Population.* Carnegie Mellon University, Laboratory for International Data Privacy.

[37] Sweeney, L. (2002) *Comments of Latanya Sweeney, Ph.D., to the Department of Health and Human Services On Standards of Privacy of Individually Identifiable Health Information*. Available at: [privacy.cs.cmu.edu/dataprivacy/HIPAA/HIPAAcomments.html] (last access: January 2009)

[38] Sweeney, L. (2002) k-anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems,* 10(5): 557–570.

[39] Sweeney, L. (2002) Achieving k-Anonymity Privacy Protection Using Generalization and Suppression. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems,* 10(5): p. 18.

[40] Truta, T.M., Campan, A., Abrinica, M., and Miller, J. (2008) A Comparison between Local and Global Recoding Algorithms for Achieving Microdata P-Sensitive K-Anonymity. *Acta Universitatis Apulensis,* Alba Iulia, Romania, No. 15, 213–233.

[41] Willenborg, L and DeWaal, T. (1996) *Statistical Disclosure Control in Practice.* Lecture Notes in Statistics, Vol. 111, Springer.

[42] Winkler W. (2002) *Using simulated annealing for k-anonymity*. Research Report Series Number 2002-07, US Census Bureau Statistical Research Division, Washington, DC, USA.

[43] Wong, R., Li, J., Fu, A., and Wang, K. (2006) {alpa,k}-Anonymity: An enhanced k-anonymity model for privacy-preserving data publishing. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Philadelphia, USA. ACM Press, 754-759.

[44] Xu, J., Wang, W., Pei, J., Wang, X., Shi, B. and Fu, A.W. (2006) Utility-Based Anonymization Using Local Recoding. *12th ACM SIGKDD international conference on Knowledge discovery and data mining*, Philadelphia, USA, ACM, 785–790.

Appendix A: Data Sets Details and Hierarchies

In this appendix, we provide some details about the data sets used for the evaluations, including the hierarchies for every QI (The QIs chosen are quite typical of what is seen in realistic situations [14][15][27][36]). We have six data sets: Adult, CUP, FARS, Pharm, ED, and Niday, introduced in Table 9. We will go through them in that order.

Adult

This is a machine Learning Database from the US Census Bureau, provided by Bren School of Information and Computer Science at the University of California, Irvine. This data set has 31062 records and 15 attributes. More information can be found at: http://archive.ics.uci.edu/ml/datasets/Adult

Eight quasi-identifiers were chosen for this experiment, we list them with their corresponding hierarchies:

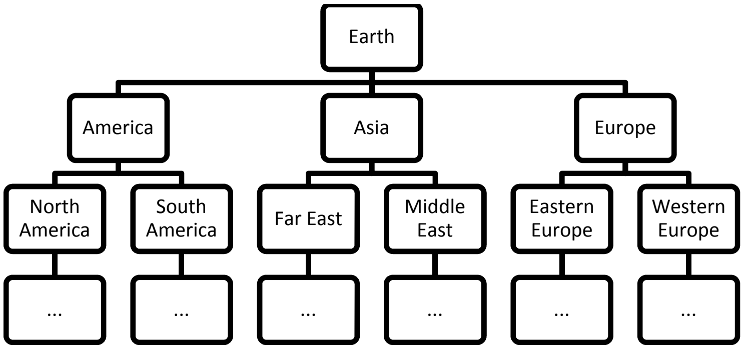

Figure 36 GH of Native-Country

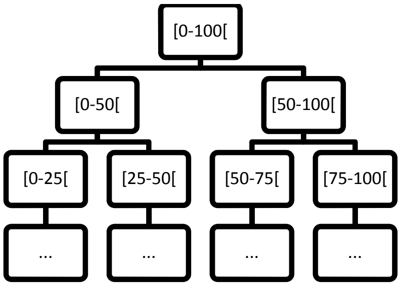

Figure 37 GH of Age

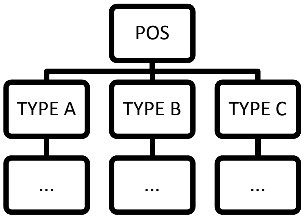

Figure 38 GH of Occupation

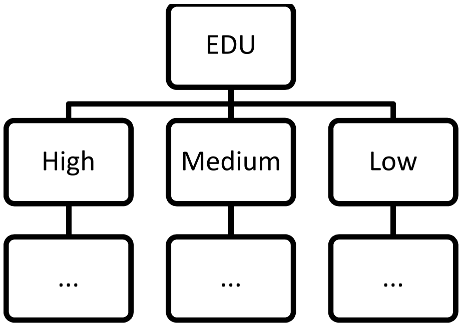

Figure 39 GH of Education

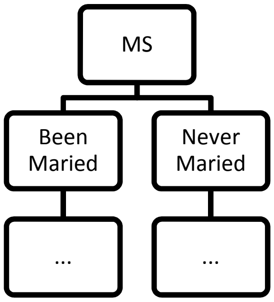

Figure 40 GH of Marital Status

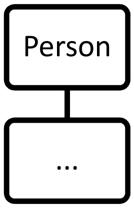

Figure 41 GH of Race

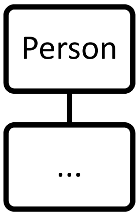

Figure 42 GH of Sex

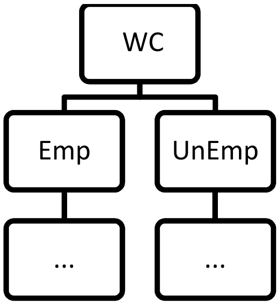

Figure 43 GH of Work Class

With respect to these hierarchies, the lattice size is equal to 5184 nodes.

The above hierarchies were chosen as such because of the nature of the unique items in each attribute. As well, the hierarchies for the remaining data sets were chosen related to the corresponding unique items.

CUP

The data set for CUP has been provided by the Paralyzed Veterans of America (PVA). PVA is a not-for-profit organization that provides programs and services for US veterans with spinal cord injuries or disease. With an in-house database of over 13 million donors, PVA is also one of the largest direct mail fund raisers in the country. For more information, see http://kdd.ics.uci.edu/databases/kddcup98/kddcup98.html

This data set has 63,411 records and 479 variables. The quasi-identifiers are: Zip code, Age, Gender and Income.

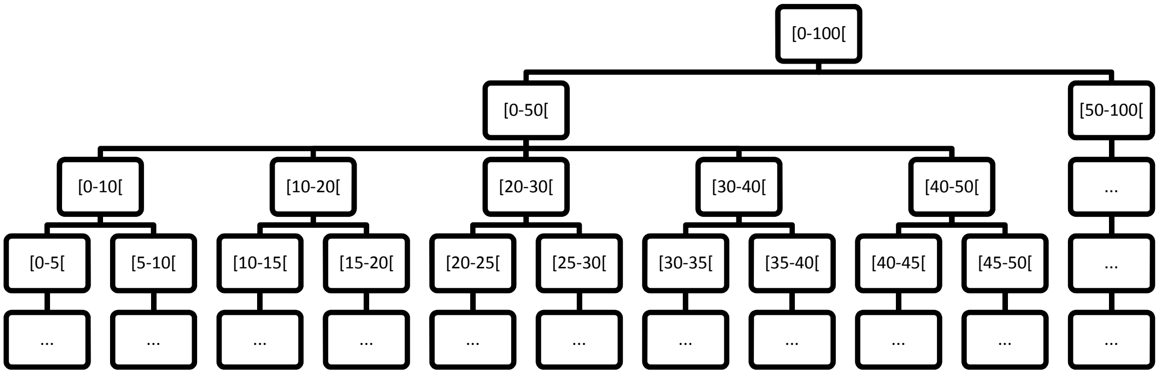

Figure 44 GH of Age

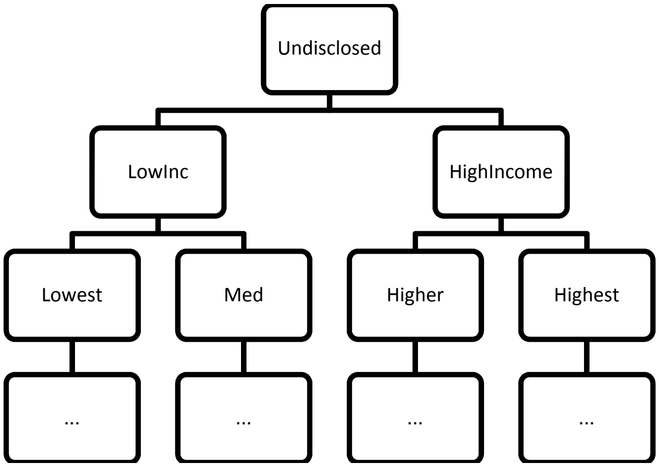

Figure 45 GH of Income

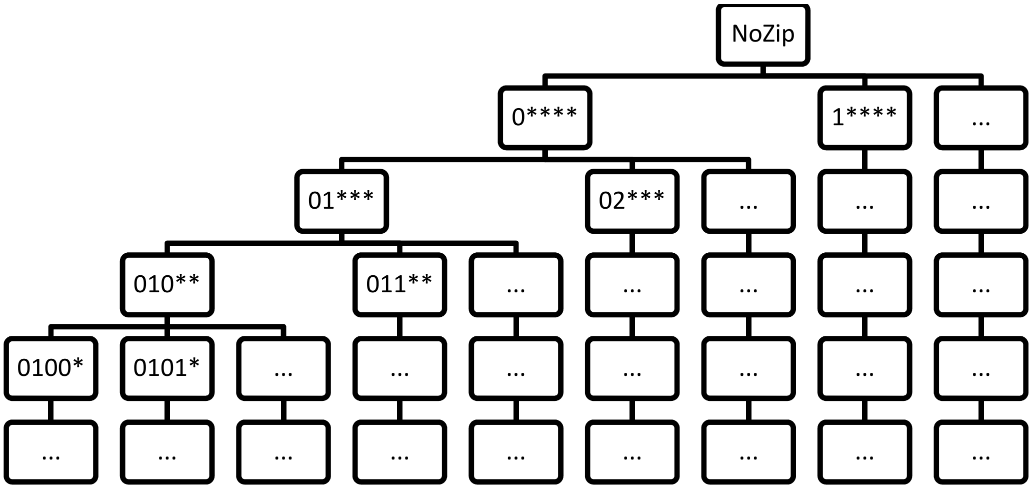

Figure 46 GH of Postal Code

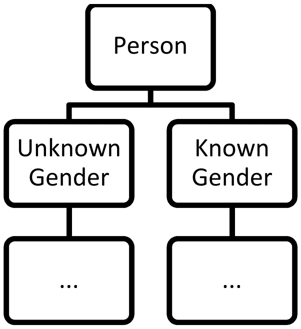

Figure 47 GH of Gender

The gender hierarchy has one more level because the unique items for this variable are composed as follows:

M = Male

F = Female

U = Unknown

J = Joint Account, unknown gender

FARS

FARS stands for Fatality Analysis Report System. The National Highway Traffic Safety Administration (NHTSA) decided in 1996 to make FARS data easier to obtain by using Internet technologies. This FARS Web-based Encyclopedia offers a more intuitive and powerful approach for retrieving fatal crash information. For more information about the Web site, visit http://www-fars.nhtsa.dot.gov/

This data set has 101,034 records and 120 attributes of which 4 are chosen as quasi-identifiers: Age, Race, Month of Death and Day of Death. The hierarchy for Age is similar to Figure 44, and the hierarchy for Race is a straightforward hierarchy with one level where all races are generalized to person.

Month of death:

The unique items of this attribute are the months, between 1 and 12 for January till December respectively. The hierarchy splits the year into two halves and each half into two quarters in the year. Moreover, there are some other numbers that indicate that the month was unknown or blank or undisclosed. The following is the corresponding hierarchy:

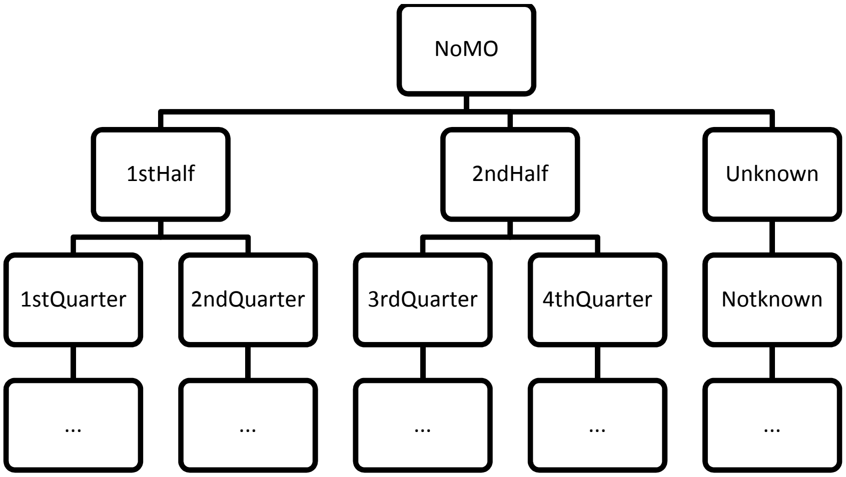

Figure 48 GH of Month of death

Day of death:

The days of the month are split into 5 weeks and an unknown.

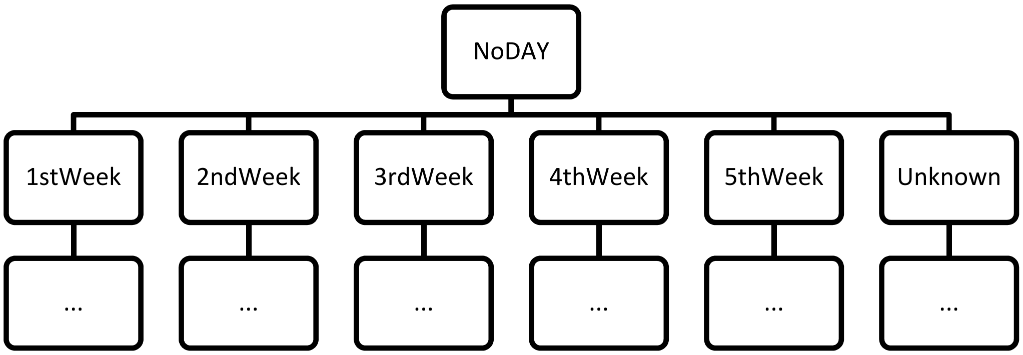

Figure 49 GH of Day of death

Pharm

These are prescription records from the Children's Hospital of Eastern Ontario pharmacy for 18 months. This is for inpatients only and excludes acute cases. This data set is disclosed to commercial data aggregators. It has 63,441 records and 10 columns.

Five quasi-identifiers where chosen for this data set: Age, Postal Code (Forward Sortation Area, FSA), Admission Date, Discharge Date, and Sex. The Sex hierarchy is the same as Figure 42. Admission Date and Discharge Date have similar hierarchies.

This is the age of the individuals in weeks.

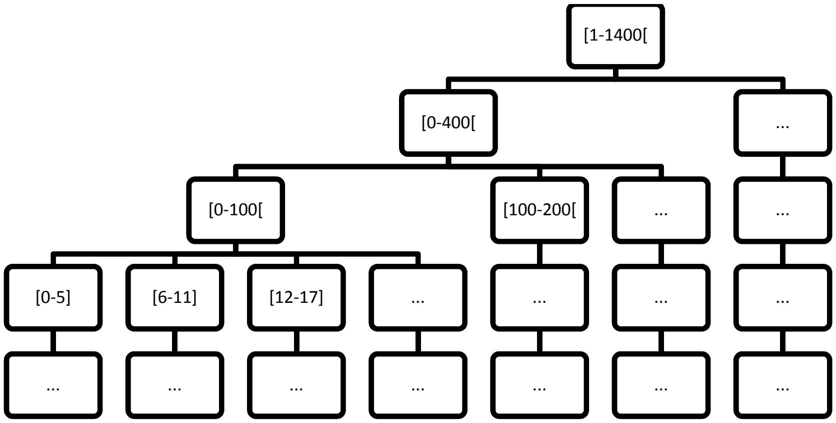

Figure 50 GH of Age

This is the postal code (FSA) made of first three letters of the full postal code.

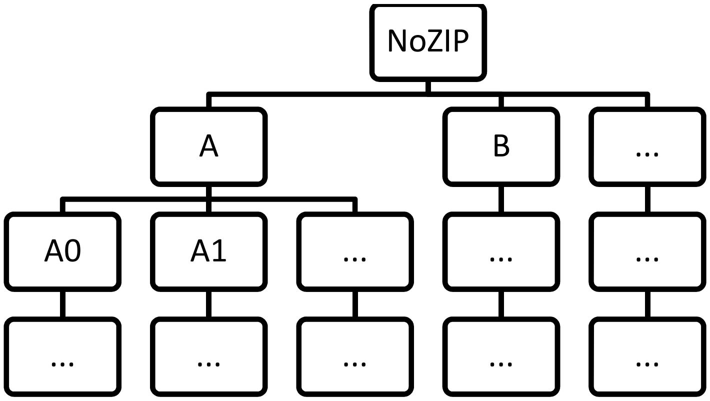

Figure 51 GH of Postal Code

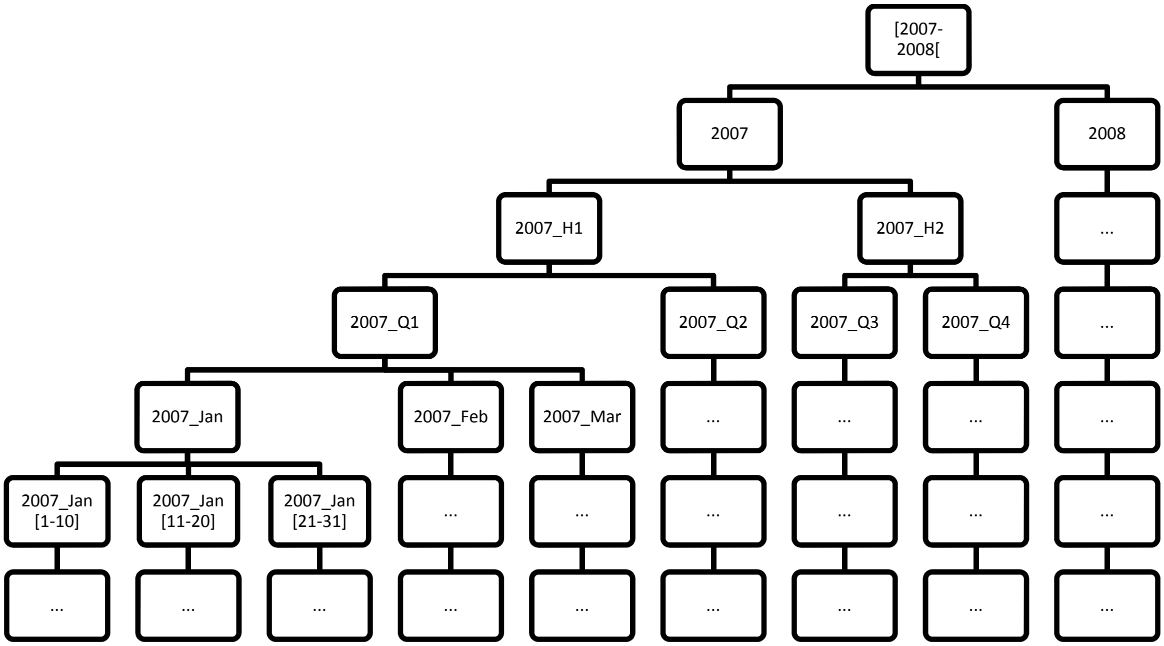

Figure 52 GH of Admission date

ED

These are emergency department records from the Children's Hospital of Eastern Ontario for July 2008. This data set is disclosed for the purpose of disease outbreak surveillance and bio-terrorism surveillance.

This data set has 7,318 rows and 15 columns. The quasi-identifiers are: Admission date, Admission time, Postal Code, Date of Birth (DOB) and Sex. The latter is similar to the above.

The date was in July 2008, thus that month was split into 5 weeks, and each week split into two, Monday to Wednesday and Thursday to Sunday.

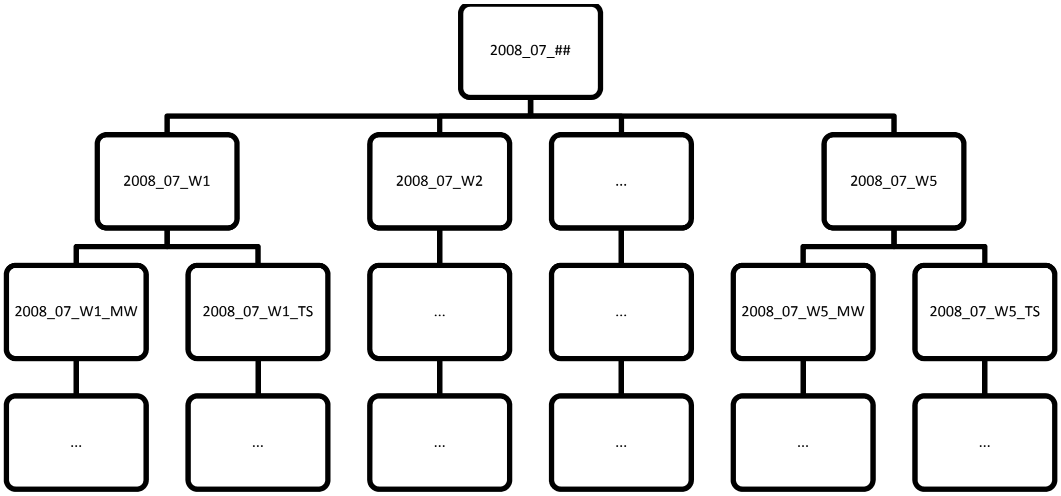

Figure 53 GH of Admission date

The admission time is in 24-hour format starting from 0000 and ending with 2359.

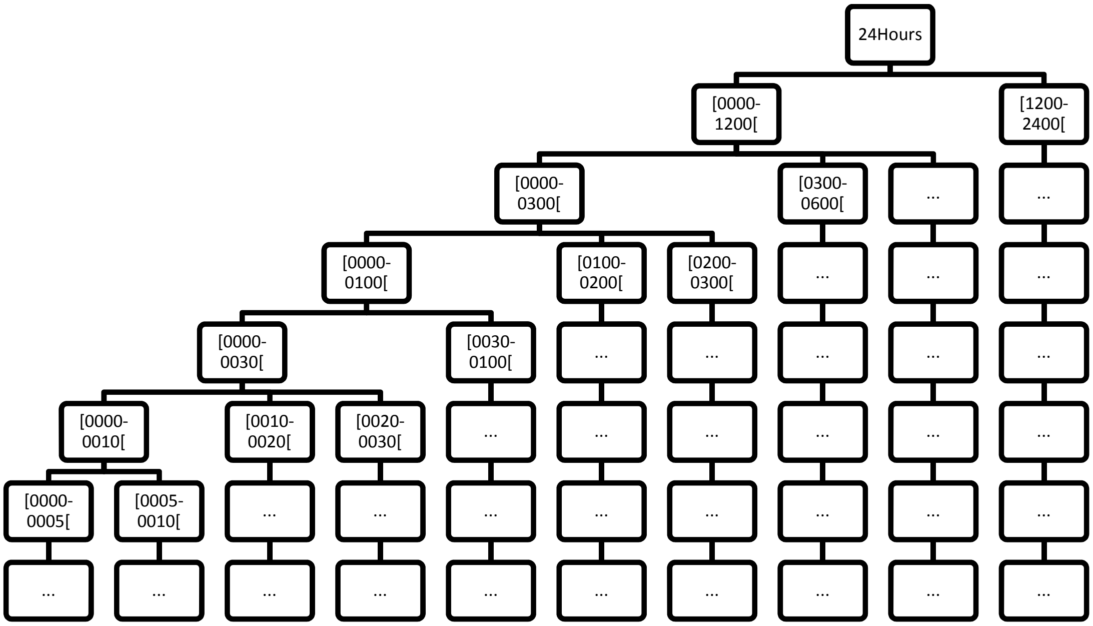

Figure 54 GH of Admission time

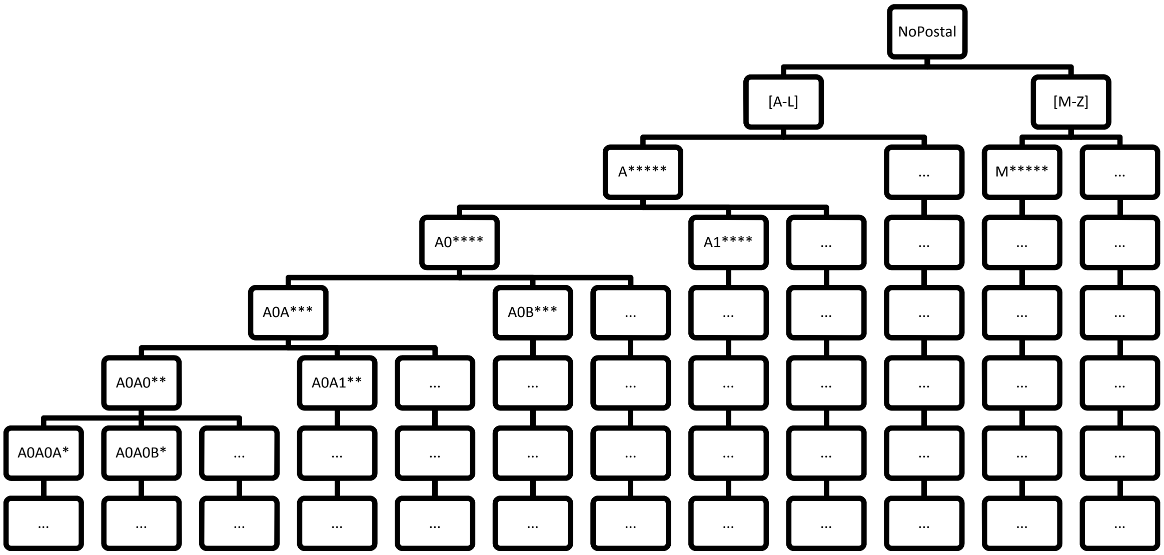

Figure 55 GH of Postal Code

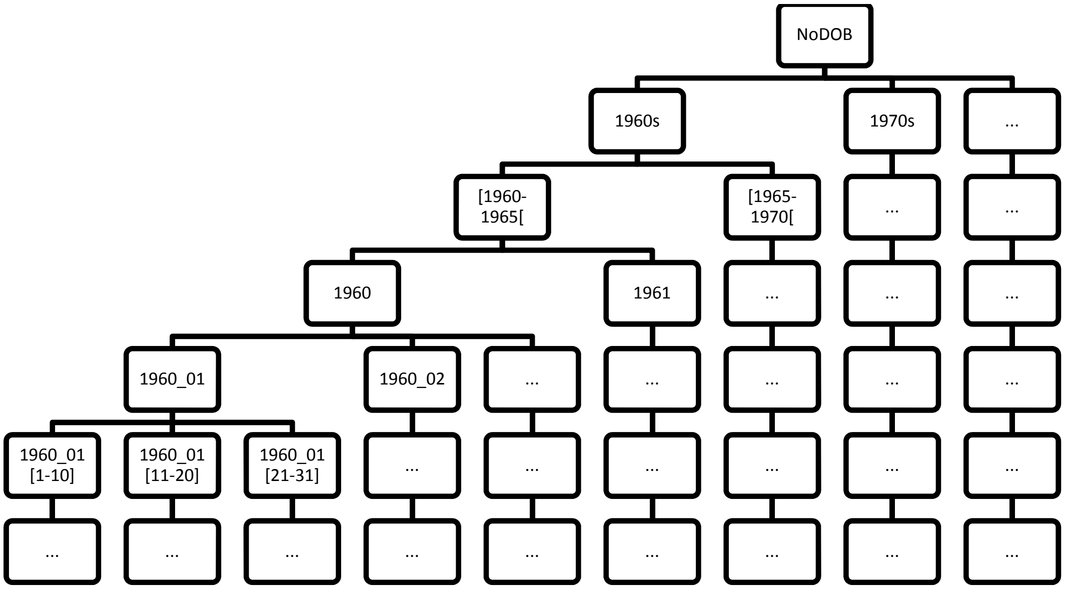

Figure 56 GH of DOB

Niday

This is a newborn registry for Ontario for 2006-2007. For more information visit https://www.nidaydatabase.com/info/index.shtml

This data set has 125,017 records and 116 columns, out of which we chose 4 quasi-identifiers.

Maternal postal code: Similar to Figure 55.

Baby Sex: The sex was sometimes undisclosed or unknown when entering the record to the data set. Therefore, we have the following hierarchy:

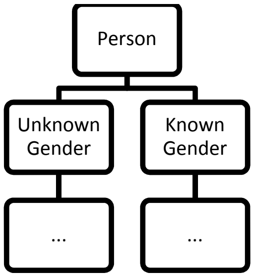

Figure 57 GH of Baby Sex

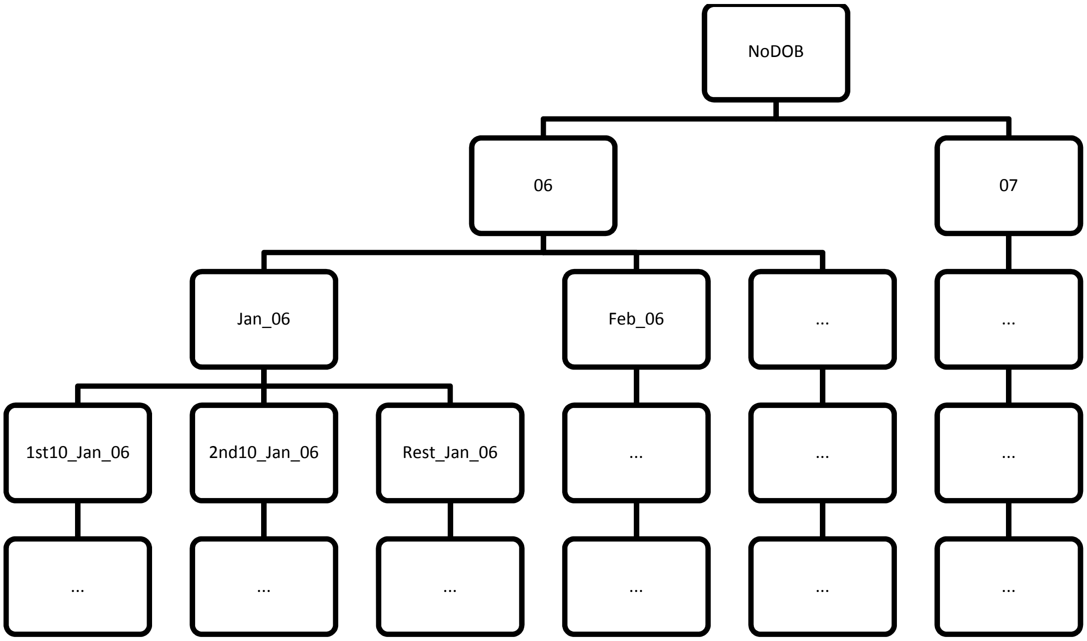

Figure 58 GH of Baby's Date of Birth

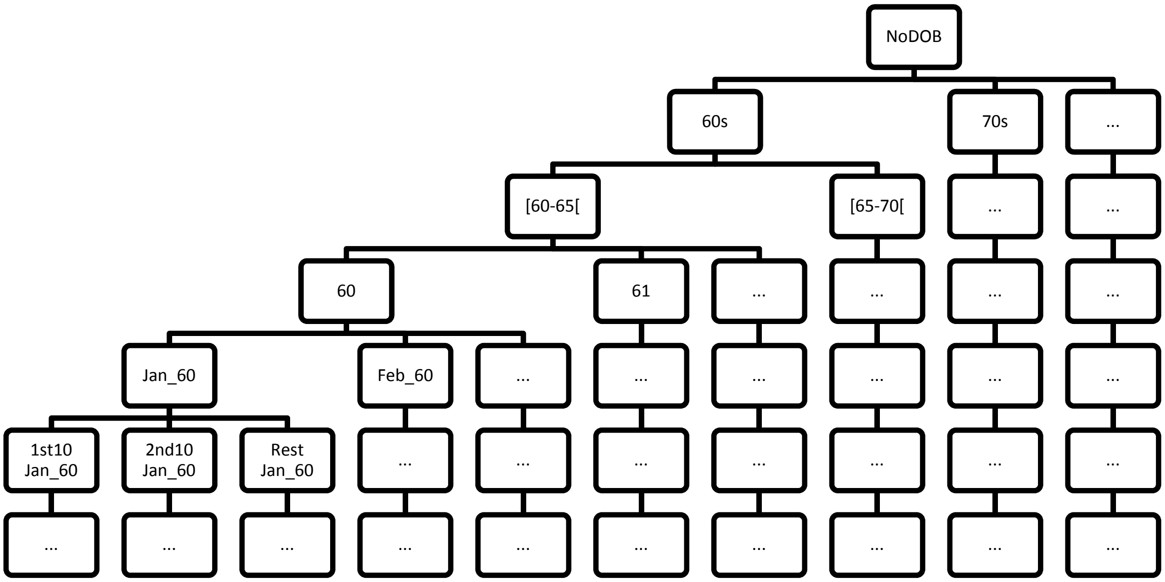

Figure 59 GH of Mother's Date of Birth

Appendix B: Additional Results

This appendix contains additional results corresponding to every aspects of empirical evaluation mentioned in this thesis. The results are split into two experiments, one with MaxSup = 1% and the other with MaxSup = 10%.

MaxSup = 1%

Empirical evaluation of Datafly and Samarati with respect to the optimal solution, with information loss metrics: DM, Prec and Non-uniform Entropy (NE).

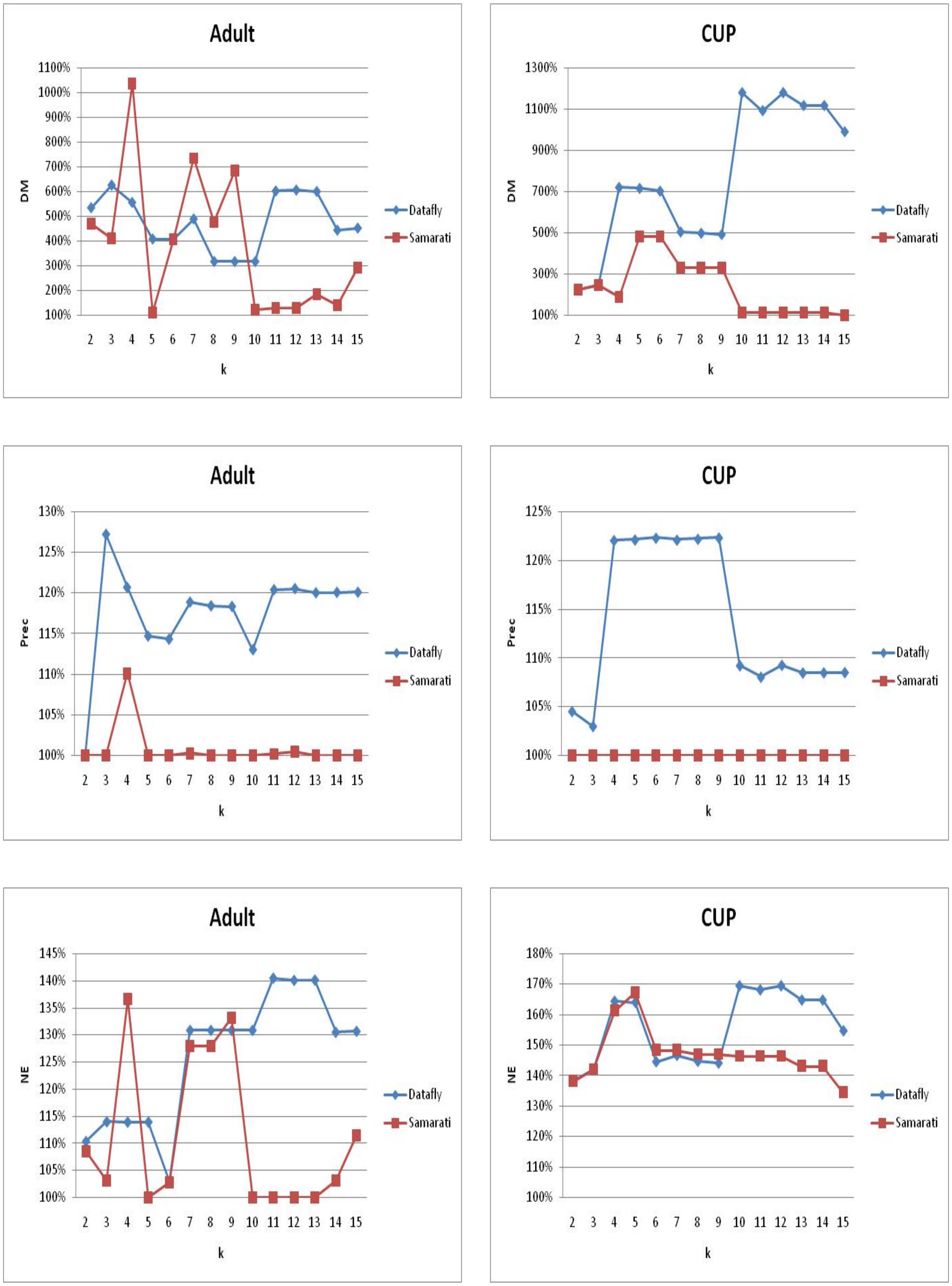

Figure 60 Information loss comparison for Adult and CUP data sets. MaxSup = 1%

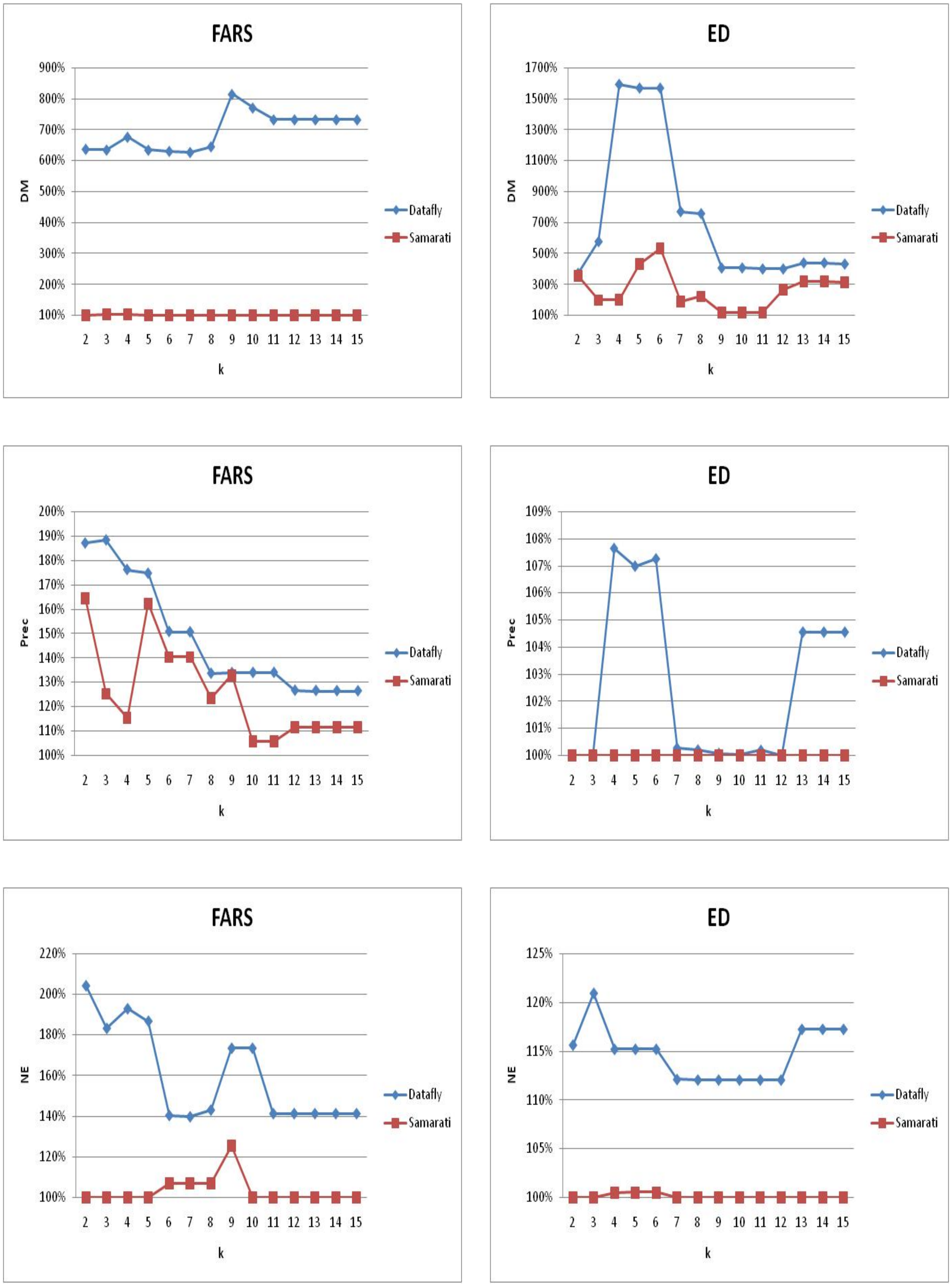

Figure 61 Information loss comparison for FARS and ED data sets. MaxSup = 1%

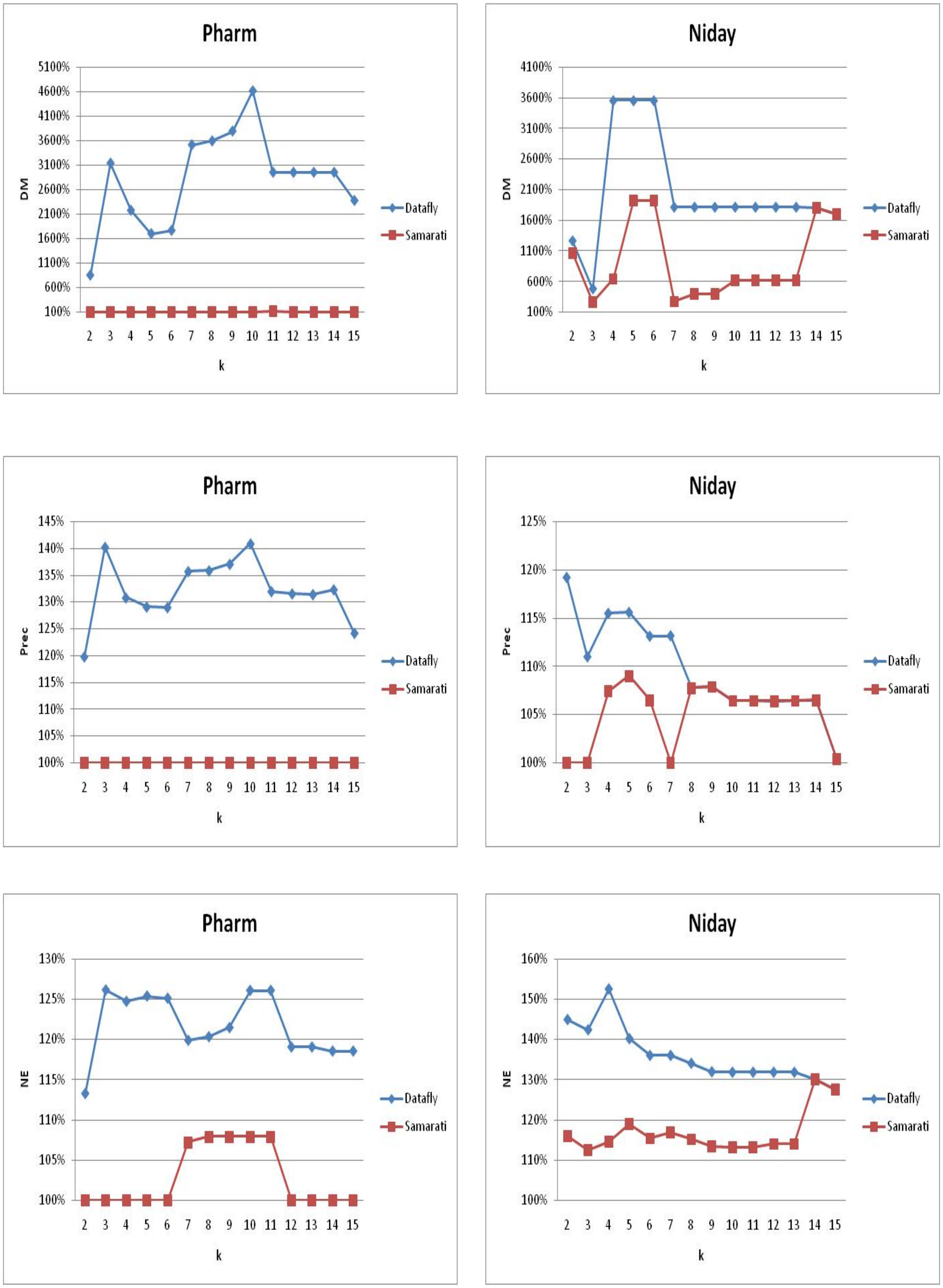

Figure 62 Information loss comparison for Pharm and Niday data sets. MaxSup = 1%

Efficiency related graphs of our approach

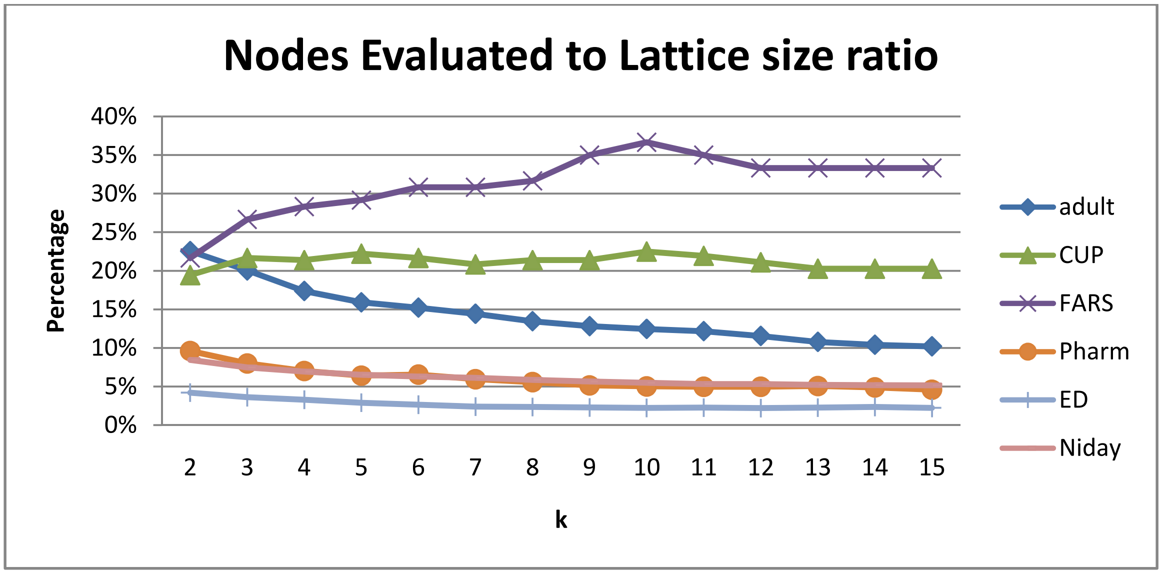

Figure 63 “Number of evaluations” to “lattice size” ratio. MaxSup = 1%

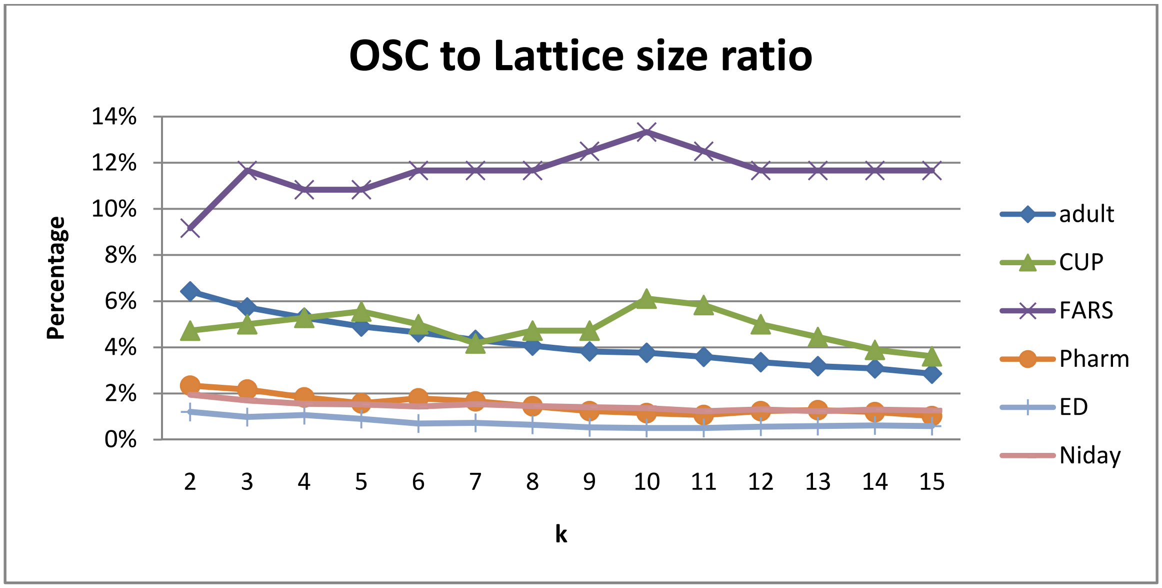

Figure 64 “OSC” to “lattice size” ratio. MaxSup = 1%

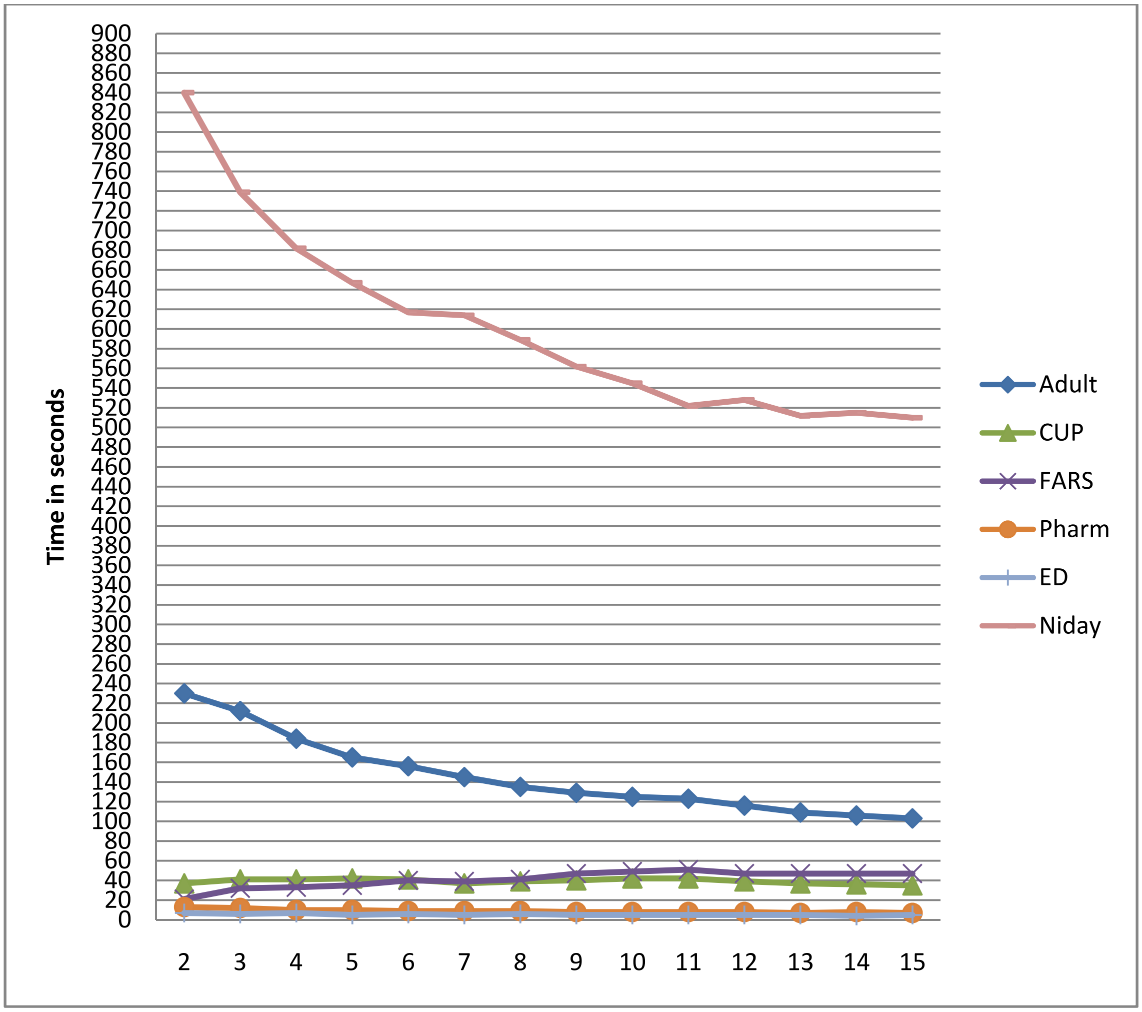

Figure 65 Execution time in seconds. MaxSup = 1%

Comparison with Incognito

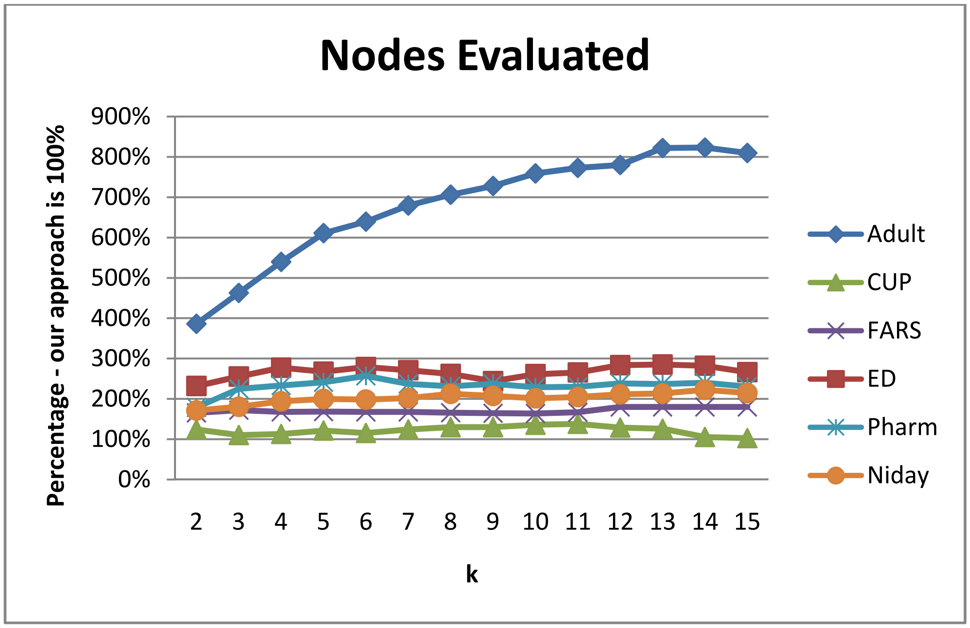

Figure 66 Nodes evaluated ratio. MaxSup = 1%

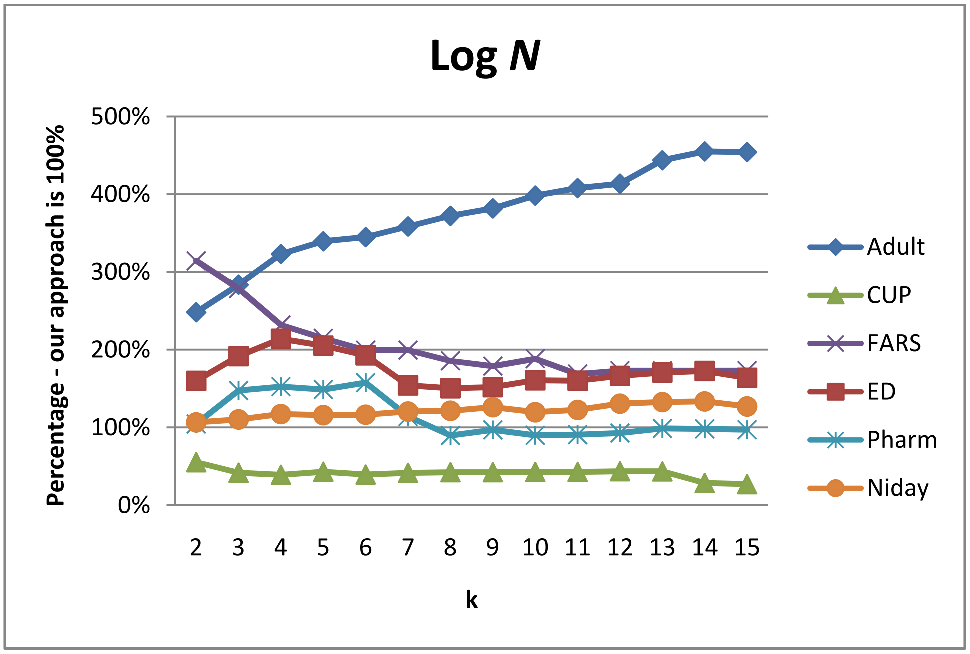

Figure 67 Performance score of Incognito with respect to our approach

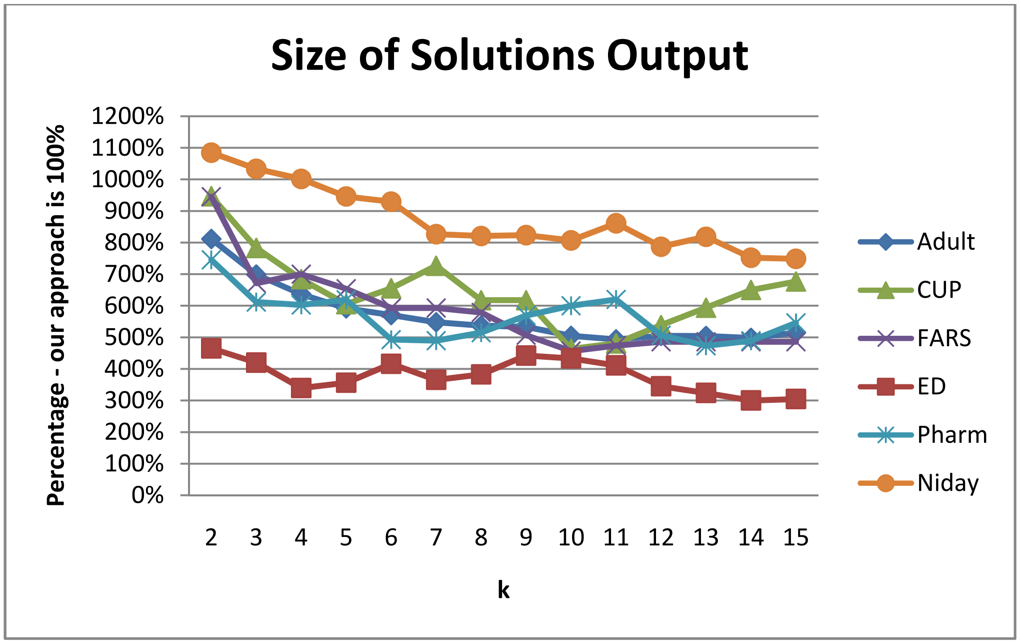

Figure 68 Size of solutions output by Incognito. MaxSup = 1%

MaxSup = 10%

Empirical evaluation of Datafly and Samarati with respect to the optimal solution, with information loss metrics: DM, Prec and Non-uniform Entropy (NE).

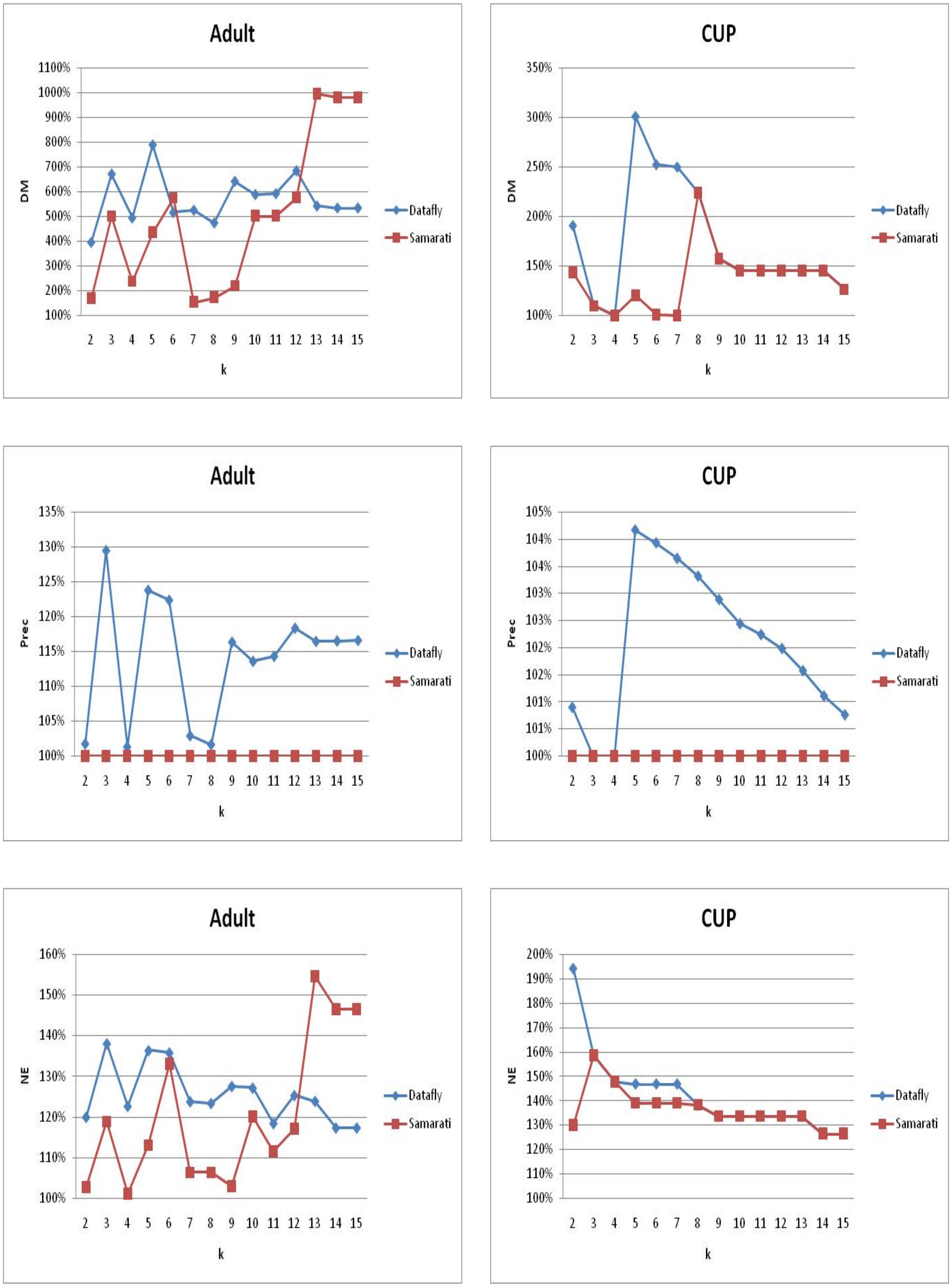

Figure 69 Information loss comparison for Adult and CUP data sets. MaxSup = 10%

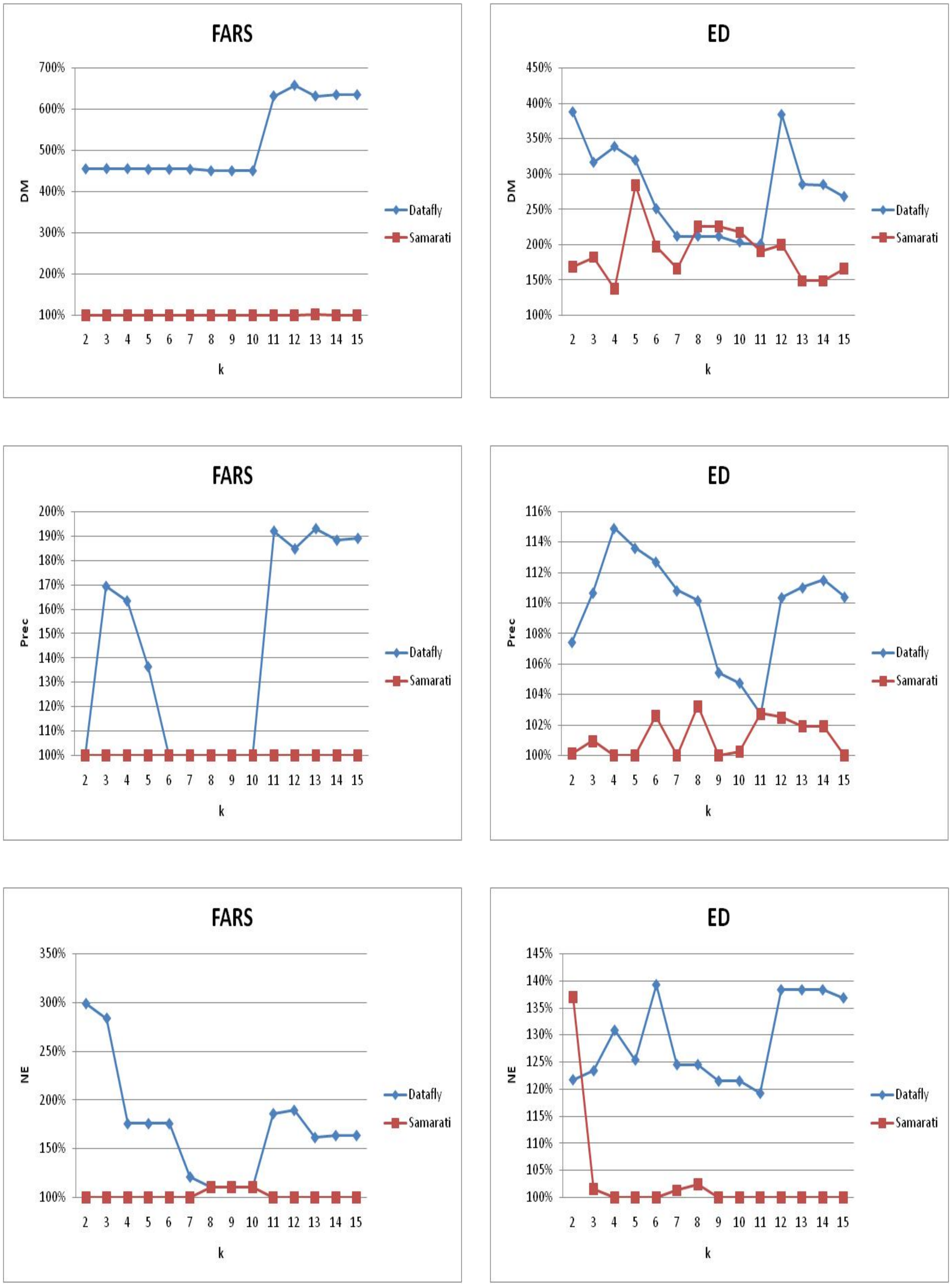

Figure 70 Information loss comparison for FARS and ED data sets. MaxSup = 10%

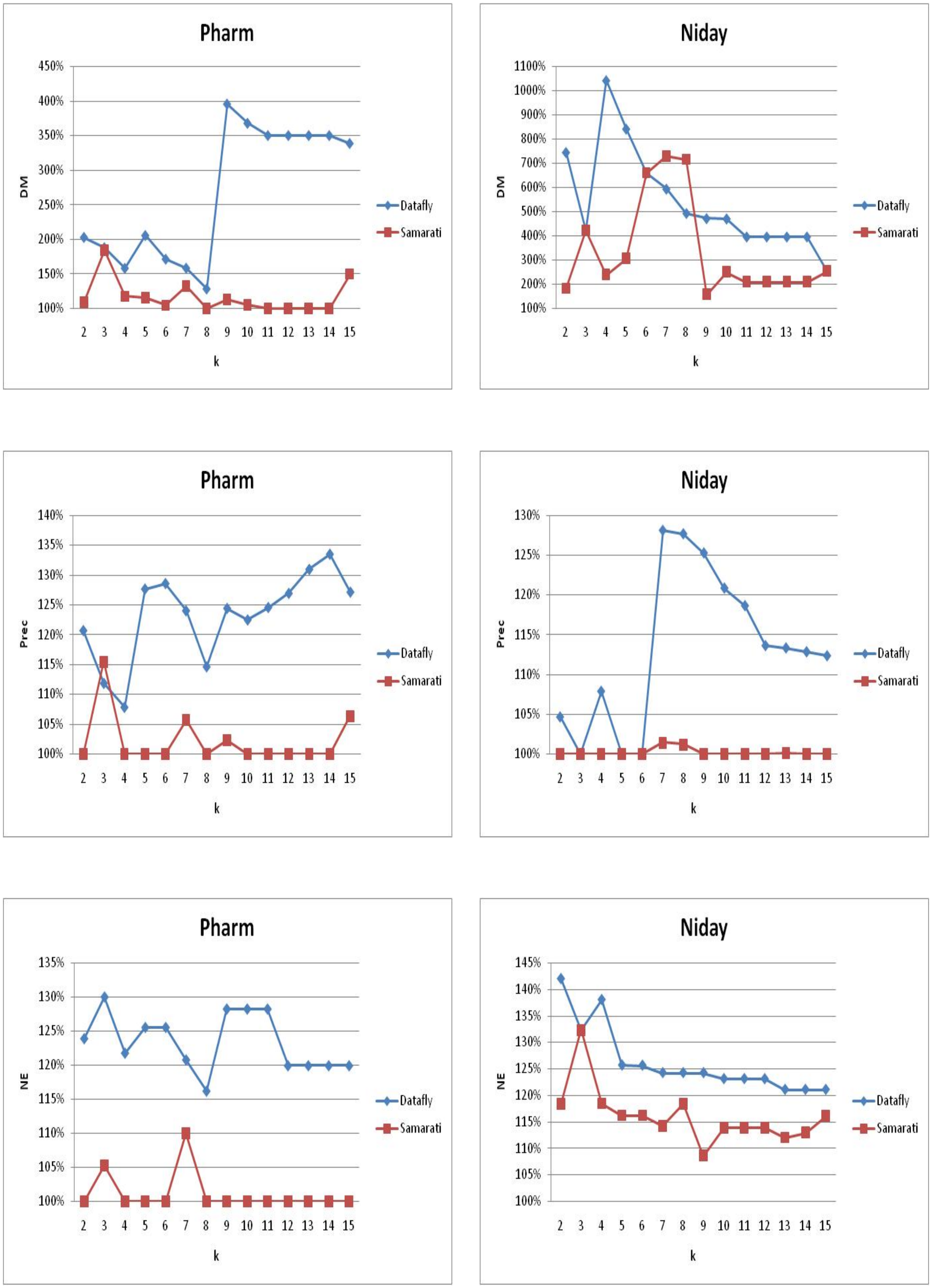

Figure 71 Information loss comparison for Pharm and Niday data sets. MaxSup = 10%

Efficiency related graphs of our approach

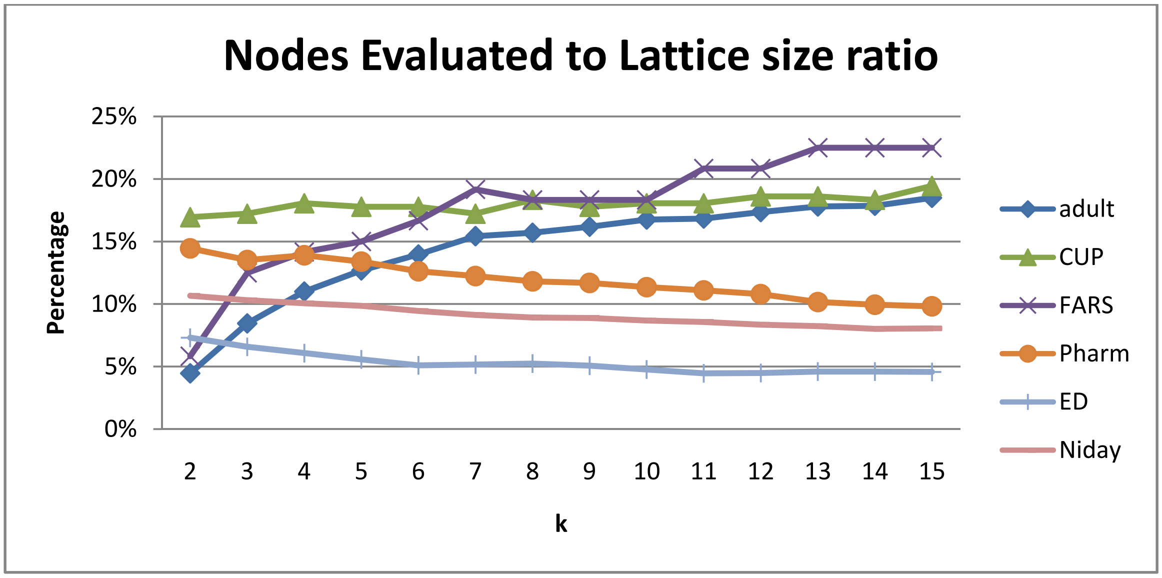

Figure 72 "Number of evaluations" to "lattice size" ratio. MaxSup = 10%

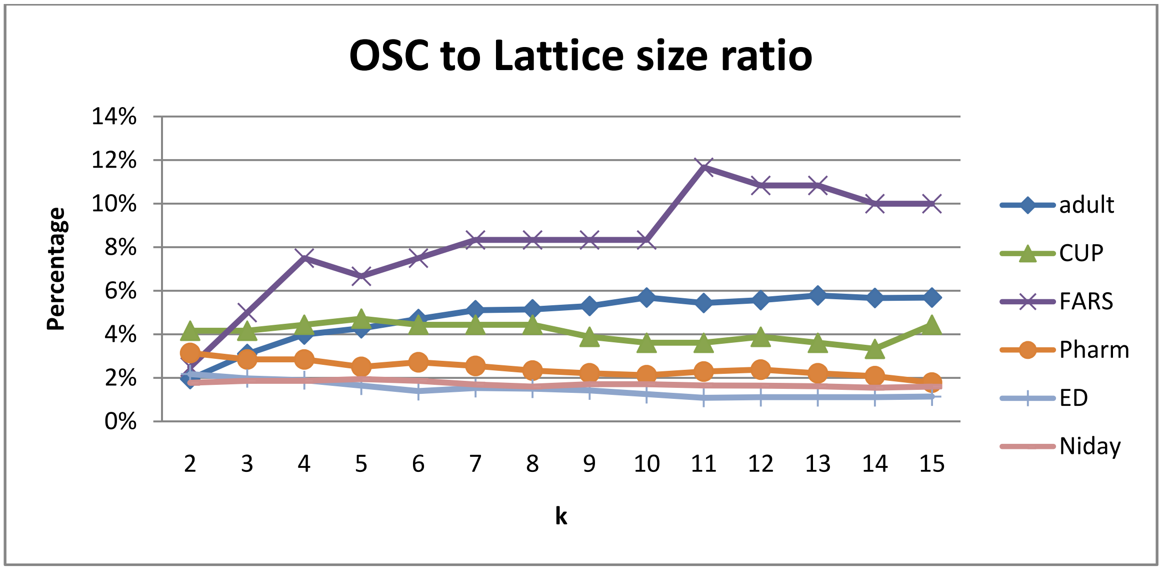

Figure 73 "OSC" to "lattice size" ratio. MaxSup = 10%

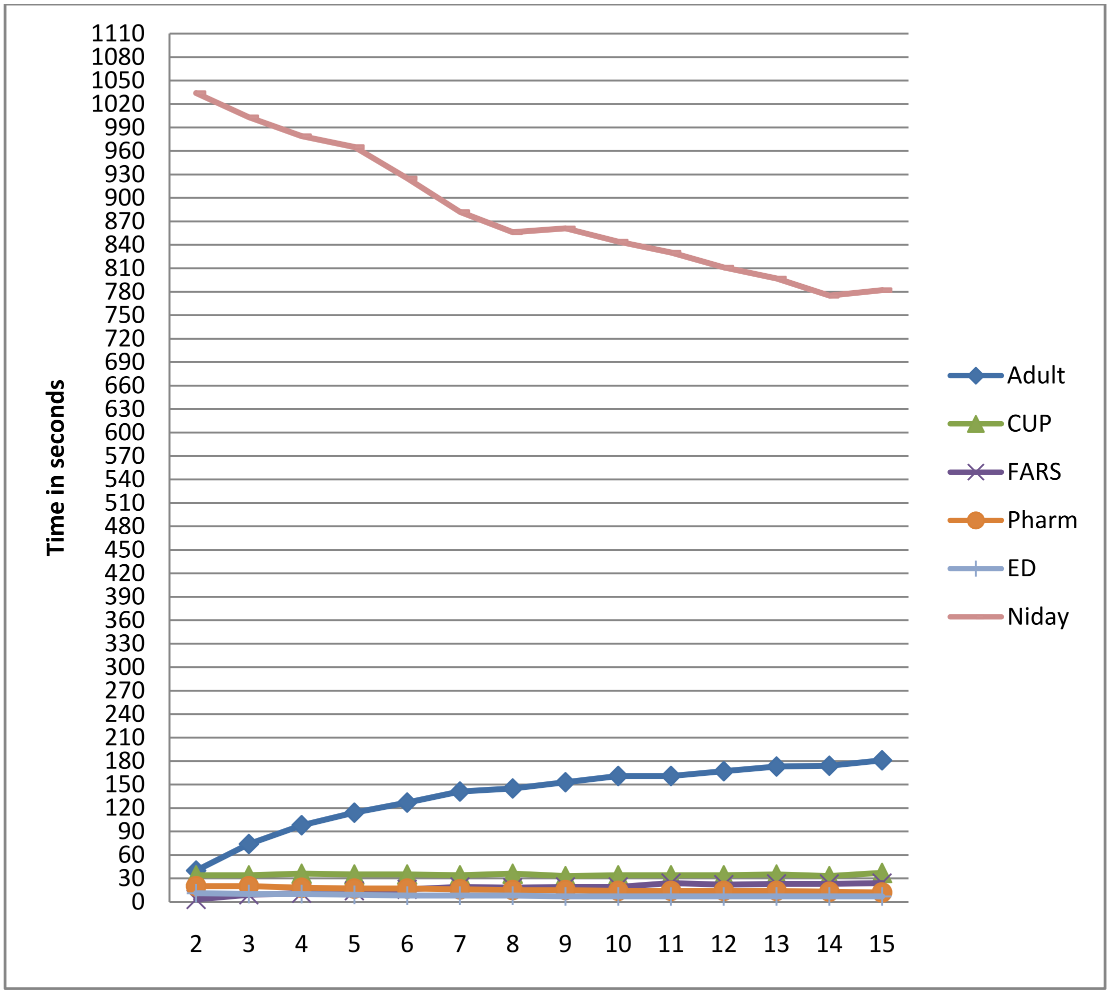

Figure 74 Execution time in seconds. MaxSup 10%

Comparison with Incognito

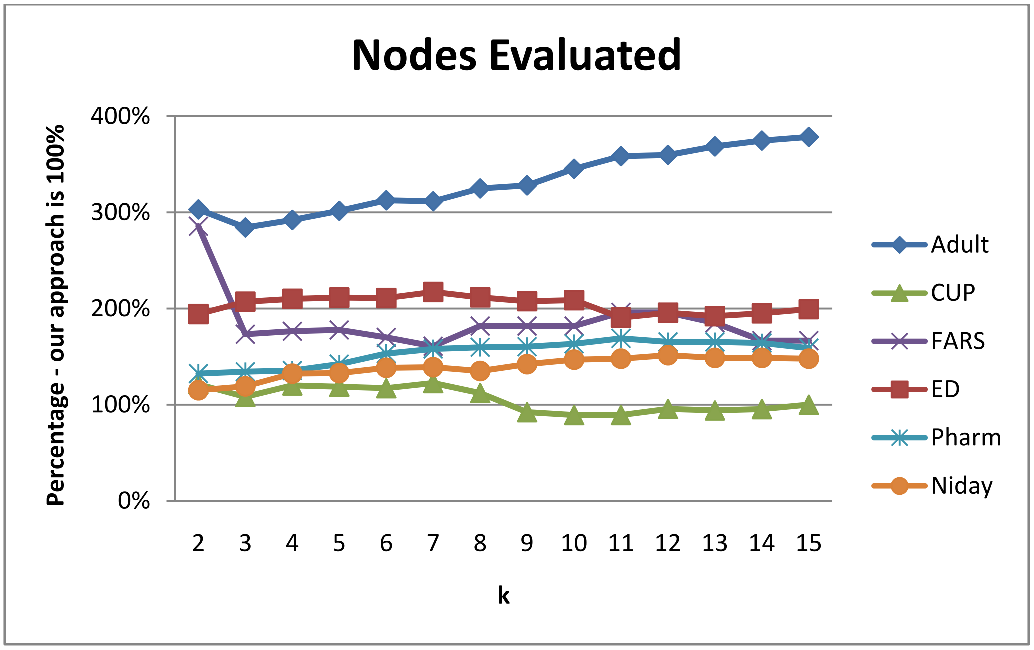

Figure 75 Nodes evaluated ratio. MaxSup = 10%

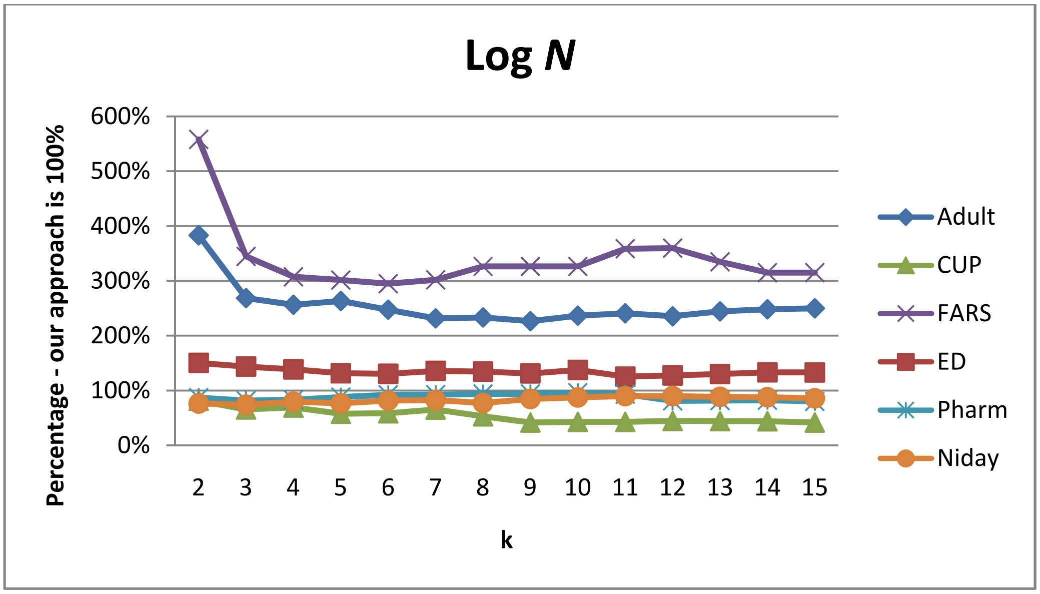

Figure 76 Performance score of Incognito with respect to our approach

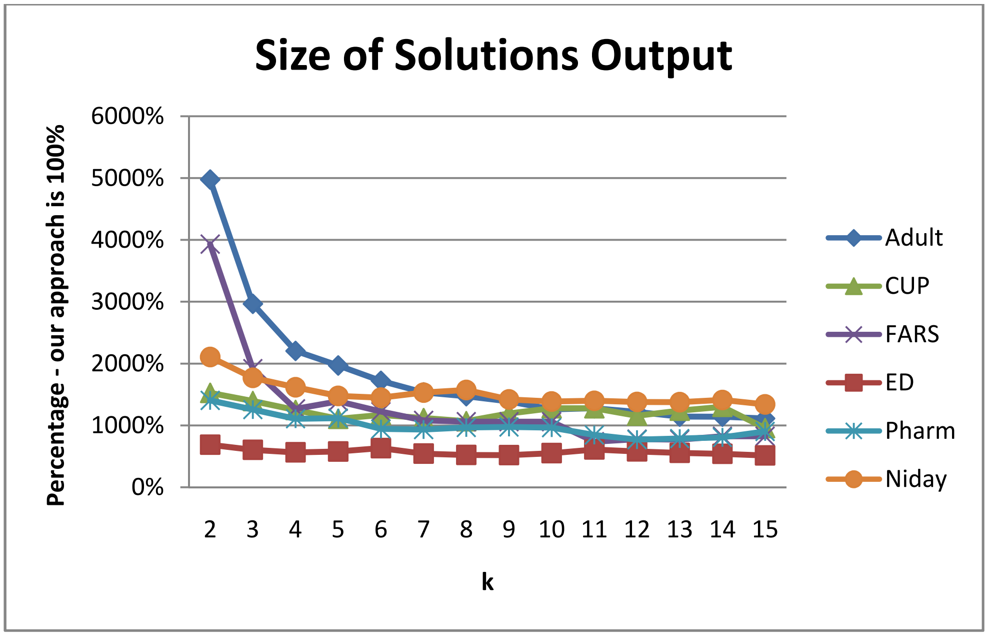

Figure 77 Size of solutions output by Incognito. MaxSup = 10%