ANALYSIS OF RANDOM EQUIVALENCE CLASS SELECTION ALGORITHMS FOR K-ANONYMITY AND ANOTHER $O(K)$-APPROXIMATION OF OPTIMAL K-ANONYMITY BY CELL SUPPRESSION

By

David M. Cerna

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Major Subject: COMPUTER SCIENCE

Approved:

_____

Professor Bülent Yener, Thesis Adviser

Rensselaer Polytechnic Institute
Troy, New York

July 2010
(For Graduation August 2010)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

Other then thanking my advisor Professor Bülent Yener for discussions and guidance through the research process, I would also like to thank Nadezhda Vassilyeva. I would like to thank her not only for proof reading my thesis, but without her I don't think I would be here; thanks for helping me find my potential.

ABSTRACT

In this thesis, we statistically analyze random equivalence class selection algorithms under different constraints and information loss metrics. This is done in an attempt to heuristically improve the algorithms found in [1, 2] by adding a mixing constraint found in each algorithm or designed by us. We test the algorithms on six different distributions and also take 16 different statistics on the resulting k-anonymous databases. As well as analyzing the algorithms, we test our information loss metric based on information entropy. We also present another $O(k)$-approximation algorithm for cell suppression K-anonymity based on the work done in [3].

1. INTRODUCTION

Databases have become an integral part of our society. Every day more information is stored including information for medical records, political party affiliation, and financial records. At the same time as institutions store these massive loads, of data many organizations also want access to this information for analysis and studies. There are many benefits brought by these studies, but at the same time there are many security risk involved. The originator of the databases wants to, or has to by the law, protect the individuals whose data is stored in the database; however, at the same time they do not want to limit the access to the data to the point where data-mining is completely useless. Such problems were often solved by simply removing the information which uniquely identifies users in the database. Such things as SIS, names, maiden names, etc. were removed from the information sent to the third party doing the analysis.

This process of sanitizing the database has been used for awhile, but is nowhere close to being secure. This process removes only the obvious links, but leaves the possibility of making new links with data from public sources. These less obvious links can be found in pseudo-identifying information such as birthdays, sex, zip code, etc. Information such as the latter, when mixed with public databases that actually have data uniquely identifying users, removes the security that sanitizing the data provided. Thus, a method is needed to protect against an adversary trying to find the sanitized parts of a database through inference. Two proposed methods are generalization and perturbation [4].

The first method to discuss (which is also the method least used) is perturbation, or changing the data so that what is given to the third party cannot usefully be matched to publicly available data. This method is, essentially, add noise to the data in the database, but keep the same distribution of attributes and members. Also, perturbation can be achieved by changing the current distribution of attribute values to a new distribution. Thus, the third party will not see the correct birthday for individuals but will see a perturbed date.

The upside to the perturbation method is there is no loss of information. The state of the information is changed, but anything derived from the data in the perturbed database would have existed in the original. However, the problem with this method is quite obvious. If you perturb the data, you are still holding the original database, just in a different attribute name space or a new derivable distribution. It is entirely possible for an adversary to take perturbed information and find out what changes were made or what distribution was used to add the random noise. Thus, once an adversary finds this information, the data can be brought back to the original state and is no longer protected. As a result, this scheme offers very little security. This is why generalization has been the predominant choice in attempting to protect databases. Generalization and the algorithms that are used to implement it will be the focus of this thesis.

1.1 Generalization

Prior to the discussion of generalization, we should discuss the structure of a database. A database has a set of members and a set of attributes. For each set of attributes in the database, there is a set of values which this attribute can hold for any given member. These sets of values can be either continuous or discrete. Generalization is accomplished by taking a subset of values of an attribute and calling this subset a new attribute value. Mattering on the type of generalization (and this will be discussed shortly) either all members which have a value in the subset will have their value changed to the new attribute value or only some of the members will make that change. The problem that generalization is attempting to solve is the uniqueness that can tacitly exist in the pseudo-identifying attributes of a database. This tacit uniqueness is what allows adversaries to match the attributes with data in the public domain.

We will define generalization as, take a set of values from an attribute and join the values together to make a new attribute value. For example, if an attribute has the values $1, 2, 3, 4, 5$, we can generalize $1, 2, 3$ by making the new value $\{1|2|3\}$. After this generalization the attribute value set will be $1, 2, 3, 4, 5, \{1|2|3\}$. The attribute $\{1|2|3\}$ can be read as "*1 or 2 or 3*".

A problem that arises when generalizing is information loss. The reason pseudo-identifying attributes were left in the database for the third party is that they can be useful in statistical analysis. Studies that try to match diseases or financial information to sex, race, locality, etc. need information about those attributes to make inferences. Thus, any loss of information will hurt analysis of the database. For example, if we where to generalize sex to "*male or female*", we have now suppressed all the information that can be inferred about the members sex. This shows the importance of correctly generalizing a database to not only allow for the minimum information loss, but also to have information loss in attributes that can afford to lose information. The question now is how can we understand the acceptable amount of information loss a database can sustained? Also, how can the security of the database be understood after a generalization procedure? These questions lead to the development of K-anonymous generalization.

1.1.1 K-anonymous generalization

A database is considered to be K-anonymous, if given any member of the database there are at least K-1 other members in the database sharing the same value for the pseudo-identifying attributes[1]. This essentially means that a guess would have to be made by the adversary. The guess that the adversary would have to make is which of the K members represented in the pseudo-identifying attributes is the member they have targeted. The generalizations do not hide the attributes that the members in a given K-anonymous set may have, but generalizations do hide the possible values that each member initially had. Thus, it is entirely possible for an adversary to know exactly which of the K-anonymous sets their target is in, but not which member in that K-anonymous set is the target. Nonetheless, it would be more advantageous to design a set of K-anonymous sets where it is hard for the adversary to even guess which set the target is in, however, information loss would most likely become an issue prior to achieving this level of security.

To get a better understanding of these methods of hiding information, we should consider a set of attributes being separated into three categories: the uniquely-

[1]To make a quick note, K-anonymity is not a completely secure system without the added security of l-diversity and t-closeness [5, 6], but I will leave discussion of these points to 1.1.2.

identifying attributes (UID), the pseudo-identifying attributes(PID), and the data attributes(DATA). Any attribute that is uniquely-identifying will be sanitized from the database prior to giving the database to a third party. Pseudo-identifying attributes are the attributes that the adversary would attempt to match with uniquely-identifying attributes through public resources. The pseudo-identifying attributes will be left in the database when given to a third party, but they will be generalized. Even after generalizing, they can be used by the adversary to find the uniquely-identifying attributes in a public resource. The attributes that represent data stored in the database are the attributes the adversary is trying to get at by piecing together the pseudo-identifying attributes and the uniquely-identifying attributes.

Making a database K-anonymous is quite simple, we can just take the members of the database and put them into groups of size K. Then after we have found all the groups of size K and spread the remaining member that did not fit into a group of size K across the groups, we generalize the pseudo-identifying attributes till each group has only one pseudo-identifying attribute set among all its members. The problem with this process is quite obvious– the amount of information loss is random. Information loss is the most important constraint to place on the database when it comes to K-anonymity. Thus, measuring and minimizing information loss is important if the K-anonymity we get after any anonymizing procedure is going to be usable by third parties.

1.1.2 *l*-diversity and *t*-closeness

As well as minimizing information loss, the distribution of values in the data attributes is also extremely important. For example, we can have K members put together that have very low information loss in their pseudo-identifying attributes, but every member in the set has the same attribute values in the data attributes. The two methods mentioned in a earlier footnote, namely l-diversity and t-closeness [5, 6], were attempts to solve similarities in the data attribute values. In l-diversity, the l stands for the diversity of the information in the data attributes per K-anonymous set. Thus, if the information held in the database was "*does the patient*

have cancer?" and "*does the patient have AIDS?*" a K-anonymous set that does not have at least l of the possible choices for the information will not be allowed. l-diversity can be implemented within most existing algorithms for K-anonymity without major changes. It can actually work on top of a K-anonymity procedure as an extra check accepting the K-set into the database. I will not be discussing l-diversity any further being that it is an addition to the security of K-anonymity and not an integral part of the K-anonymity procedure.

t-closeness [6], like l-diversity, also works on the information attributes of the database. Unlike l-diversity, the focus of the t-closeness security metric is to make the distribution of K-anonymized sets t close to the original distribution. The value t represents the percentage likeness between the K-sets data attribute set distribution and the original data attribute distribution; t is a value over $[0, 1]$. This is again, like l-diversity, an addition to the security of K-anonymity and is not an integral part of K-anonymity building procedure. We discussed these points to show where the security guarantees from K-anonymity end and what extra security methods can be used to insure security of the members. However, even if it falls short of securing the information attributes completely, a K-anonymizing algorithm which minimizes information loss sufficiently is absolutely necessary to get a database with low information loss under l-diversity and t-closeness. Both of these constraints guide the original algorithm towards better security, but neither help solve the original problem of reducing information loss. Thus, we focus on the heart of the problem, that is the procedure by which a database is K-anonymized and how well that procedure retains information. Comparison between algorithms were done without considering l-diversity or t-closeness. Results would have shown the effect of these two security procedures on the algorithm rather then the quality of the algorithm's information retention.

1.2 Algorithms for K-anonymity

1.2.1 Generalization applied to algorithms

As we can see, K-anonymity is based on the concept of generalizing the pseudo-identifying attribute set, these are the attributes belonging to the PID. The concept

of generalizing can be broken into Stepped generalization and Suppression. Suppression can also be considered the final step in a Stepped generalization process, generalizing a value till it can no longer be distinguished from any other values in the same attribute. Thus, we will discuss the Stepped generalization process and leave Suppression to inference.

Stepped generalization can take multiple forms, the most common one being a generalization tree which shows how values for a given attribute join together to make more generalized attribute values. One major benefit of using a generalization tree is that information about the distribution can be embedded in the tree. An example of information that could be embedded would be similarity between two attribute values. For example, if we where joining together values from the attribute "*personal wealth*", two possible values might be "*1,000*" and "*100,000,000*". If we were to generalize such that we have "*1,000 or 100,000,000*" as a new attribute value, it is harder for an adversary to guess that the target member is in a certain K-set. This is the case because the variance of values that were generalized over (assuming the target member is poor) is large; however, if we were to generalize to "*1,000 or 1,500*", it is quite easy to see that everybody in this group is poor, and we know where our member is located by the personal wealth attribute (it is at least easier for the adversary to make a guess when similar attributes are joined together). This can be used by the algorithm (inherently without any additional mechanisms) to make it less advantageous for the adversary to pick a particular K-set over another. Generalization trees also help to define simple information loss metrics which we will discuss shortly.

Other than using a Stepped generalization defined in a generalization tree, it can also be defined in the form of a linear progression. This is accomplished by taking the members in a given K-set, and for each attribute union the values for each members together, thus, making a new attribute value. This process is done without considering similarity between the values. The only difference between this method and the generalization tree is that there is no predefined concept of a "*good*" generalization as there is with a generalization tree. To have the predefined concept of a good generalization, we would need to have an ordering or understanding of

similarity for the attribute values. In this case, we accept all $O(n^2)$ possible combinations of attribute values (n is the number of current equivalence classes, thus this is referring to n choose 2) and let the information loss metric decide which to take. This method of Stepped generalization requires the information loss metric to decide what constitutes a good pairing, unlike the generalization tree model. Suppression becomes quite obvious after explaining Stepped generalization. Essentially, Suppression is achieved by changing the attribute value for a given member to the root of the generalization tree, or, to an attribute value comprised of all attribute values the given attribute can hold. However, Suppression can also be used to remove members completely from the database, that is, if a member is an outlier. This type of Suppression is called Member Suppression, and is used when it is hard to fit a member into K-sets without much information loss. We did not use this type of Suppression under the assumption that the database should stay intact. If it is OK to remove a member from the database, there is no reason not to use Member suppression to achieve less information loss.

Now that we have discussed what Generalization is when used in an algorithm, we can discuss the two types of Generalization commonly used: local re-encoding and global re-encoding. Local re-encoding is defined such that if a value is generalized, it is generalized only for a specific member (or K-set) in the database. In other words, a generalization is an expansion of the value space for a given attribute. Thus, if we were to select a set of members to generalize, the only members in the database to have the new generalized attribute values would be the set we selected. Global re-encoding, on the other hand, changes the set of possible values for an attribute, in this case, generalization occurs on the attributes and not on a set of members. In global re-encoding we choose attribute values to join together that will cause the least generalization (i.e., the least members will be effected by that generalization). Thus, global re-encoding is a contraction of the attribute value space, because, we replace the old attribute values with a new attribute value.

These two methods of generalization implementation give very different results in terms of information loss. For example, if the database has large number of members with "*1000*" for the "*personal wealth*" attribute, then generalizing this

	member	attribute	cell	none
attribute	AG MS	AG AS	AG CS	AG
cell	?	?	CG CS	CG
none	MS	AS	CS	?

Table 1.1: This table is an edited version of a table included in [7]. The left most column represents the types of generalization and the top most row represents the different types of Suppression. Question marks indicate mixtures which are not possible. The abbreviations in the cells are constructed as follows: G stands for generalization and S for suppression. The first letter of the word in the left most column is put before G, and the first letter of the word in the upper most row is put before S.

value at all will lead to many members being in the same equivalence class. This can lead to drastic loss of information. Out of the two methods, local re-encoding will lead to much more fine tuned information retention, but not necessarily a better result than global re-encoding pertaining to the amount of generalization necessary. The use of both local/global re-encoding and generalization/suppression can be summed up in Table 1.1. As we will discuss in a later section, the focus of our work was on cell generalization and cell suppression being that it is the most specific type of Generalization in Table 1.1. We were looking for the least information loss.

1.2.2 Formal problem description

Now that we have discussed the preliminaries, we can formally describe the problem of K-anonymizing a database under a set of constraints. By constraints we are referring to requirements such as, minimization of information loss, minimization of the maximum equivalence classes size, or not allowing equivalence classes larger than a given size.[2] Constraints such as minimizing the maximum equivalence class size might be of importance when there are many similarities between members. When using greedy algorithms for K-anonymity, all closely related members can end up in one group; this is one such situation where minimizing maximum equivalence class size is useful. Minimizing maximum equivalence class size might not appear useful (other than the special cases mentioned above), but for iterative methods,

[2] l-diversity and t-closeness are also examples of constraints, but we did not test either, as was mentioned earlier.

not using this constraint can mean losing out on a better solution; as with greedy algorithms, it could mean losing out on a solution that does not put more than half the members in the same equivalence class. Being that we are using iterative algorithms in particular, minimizing maximum equivalence class size is one of the constraint we found important enough to test.

Definition 1. *Given a database $DB(M, A)$ where M is the set of members and A is the set of attributes, a generalization method G, an information loss metric $d(\cdot,\cdot) : M \times M \to [0,1]$ where 0 is no information loss, and a set of constraints C, find a new database DB' where the constraints are satisfied and no equivalence class in DB' have a cardinality smaller than K. The members of DB are labeled as $m_i \in M$ and the attribute set is broken into 3 parts UID, PID, and $DATA$. The equivalence classes are specifically referring to the set PID. Each $A_i \in A$ is part of one of these three sets, and the constraints are labeled as $c_i \in C$.*

This formal description of the problem is different from other descriptions presented in related work, such as [8, 2, 9, 6, 5]. We used this definition, because it generalizes the problem of K-anonymity and most accurately describes our situation. As an example of what is now considered part of the K-anonymity problem, consider what we mentioned in an earlier footnote, as well as, in the section on l-diversity and t-closeness, specifically, problems in the relationship between equivalence classes in the PID and values in the $DATA$ attributes. These problems were the focus of l-diversity and t-closeness. Both l-diversity and t-closeness are considered constraints under this formal definition.

1.2.3 Information loss metric

We have yet to describe an important part of K-anonymity, how information loss is measured. There are many ways to compute information loss. The few we will discuss here represent the state of the art methods. Information loss metrics cannot always be universally used, as some algorithms required a specific method to compute the information loss. In some cases, using a certain method can actually hurt performance; for instance, all the algorithms we have implemented and studied are iterative methods, and thus, require an information loss metric which can be

computed on an incompletely anonymized database. A metric such as the discernibility metric [9] would be too coarse to get a good measure of the information loss at each iteration. The discernibility metric assumes the algorithm anonymizing the database is finished prior to calculating the metric. It works better on clustering algorithms that return a completely clustered database under some heuristics. In the equation for discernibility metric (1.1), each E is an equivalence class in the database, and D is the database. The second summation is referring to the members of the database which needed to be suppressed, thus, these members are given a cost of $|D|$, the size of the database. All other members are given a cost based on the size of their equivalence class. Two metrics which work well with the class of iterative algorithms are global certainty penalty (GCP)[10] and ancestral metric, which is referred to as cluster loss metric in [11]. Global certainty penalty (GCP)[10] uses normalized certainty penalty(NCP)[2] as a base.

$$C = \sum_{|E|\geq k} |E|^2 + \sum_{|E|<k} |D||E| \tag{1.1}$$

$$An(PID_{m_i}) = |PID_{m_i}| \cdot \sum_{j=1}^{t} \frac{height(Gen(PID^j_{m_i}))}{height(Gen(A_j))} \tag{1.2}$$

The PID_{m_i} is the entire pseudo-identity for a given member m_i, the summation is iterating through the attributes belonging to the PID, and $Gen(\cdot)$ is the generalization tree function which gives the position in the tree of a given attribute value. Using $Gen(\cdot)$, it is possible to find the height of the attribute value. The value of $Gen(A_j)$ is the root of the generalization tree, or in other words, the completely generalized attribute value (think of suppression). This is a modified version of the equation for cluster information loss from [11]. We use a slightly finer metric than cluster information loss. We take into account which leaves have the $Gen(PID^j_{m_i})$ as an ancestor, because different leaves have different heights mattering on the configuration of the tree and we want the one with the greatest height. Choosing the leaf with the greatest height from $Gen(PID^j_{m_i})$ was implemented in our version of cluster information loss.

$$GCP(D) = \frac{\sum_{PID \in D} |PID| \cdot NCP(PID)}{d \cdot N} \tag{1.3}$$

$$NCP(PID) = \sum_{i=0}^{d} w_i \cdot NCP_{A_i}(PID) \tag{1.4}$$

$$NCP_{A_i}(PID) = \begin{cases} 0 & |PID_{A_i}| = 1 \\ \frac{|PID_{A_i}|}{|A_i|} & otherwise \end{cases} \tag{1.5}$$

The above three equations are used for computing the global certainty metric. In (1.3) the value passed to the function is D, the database. The summation iterates through each of the equivalence classes $PID \in D$, and then, finds $NCP(PID)$. The denominator in (1.3) is equal to $|PID|$ multiplied by $|D|$ (we left the syntax chosen in [10]). In (1.4) the summation iterates through each of the attributes. The weight given to each attribute represents the importance of the attribute; the weights should be chosen such that $\sum w_i = 1$. In our work we made each weight $\frac{1}{|PID|}$. In (1.5) we are essentially giving each value a number based on the level of generalization, or in other words, how close to suppression is the given value. The closer the value is to suppression, the higher the value returned by $NCP_{A_i}(PID)$. These equations are modified versions of the equations given in [10]. Our implementation of GCP is exactly as stated above. We will discuss entropy when discussing our implementation.

1.2.4 Hardness of K-anonymity

The problem of K-anonymizing a database optimally under given constraints has been shown to be $NP - hard$ [3]. In [3], $NP - hardness$ was shown for the global re-encoding problem over the PID attribute set. The way K-anonymity was described was by minimizing the number of changes to the database, in other words, keeping the number of changes under a given threshold. For the problem of local re-encoding we are adding more variability, and thus, making the problem even harder. In local re-encoding, attribute generalization is done on individual members. Local re-encoding was also shown to be $NP - hard$.

1.2.5 Contribution

The main contribution of this thesis is an analysis of iteration based, random equivalence class selection algorithms for K-anonymity, specifically, the algorithms from [1, 2]. We have worked with the original constraints from [1, 2], as well as, additional constraint we have developed. Also, we have developed a new metric for information loss that uses information entropy. Entropy is used in many applications where information is very important to retain and/or measure. The metric of entropy is compared against the two metrics, Ancestor tree and GCP.

Other than the above contributions we have also evaluated different ways of viewing the problem of K-anonymity. The current popular way of viewing K-anonymity is as a cluster procedure [1, 2, 12]. For example a few of the approaches are, comparing the members by an information loss metric and then build equivalence classes [13] and finding a clustering genetically [14]. The idea we are proposing in this thesis was used to find another approximation algorithm for K-anonymity by local re-encoding under cell suppression K-anonymity. The idea is to start off with some base information and build a K-anonymous database using this assumption. The base information that we started with is, for each member of the database, find the set around this member of cardinality $k-1$ with the minimum pairwise metric sum out of all sets with cardinality $k-1$. Essentially ,we are finding the members which are closest together and close to the original member. By close we are referring to generalizing such that the two members are in the same equivalence class would require minimal information loss. This concept, in conjunction with [3] lead to another $O(K)$-approximation of optimal K-anonymity by cell suppression. So in a nutshell, the contributions of this thesis are

- Analysis of algorithms based on those proposed in [1, 2].
- Development of a new information loss metric based on entropy, as well as, comparing it to existing state of the art information loss metrics.
- Development of a new algorithm using the bi-criteria location finding problem. This algorithm is an $O(K)$-approximation of optimal cell suppression K-anonymity. It uses concepts discussed and developed in [3]. An $O(K)$-

approximation has been found twice prior to this thesis, we will leave discussion of the algorithms to the historical review section.

2. HISTORICAL REVIEW

In this section, we will discuss the related work done in the subject of algorithm design. We have already mentioned the state of the art information loss metrics, thus, we will not discuss them further. Also, there has not been much work on the problem of cell generalization, specifically, cell suppression problem. However, many of the state of the art algorithms can be changed slightly to accept this type of generalization. Changing to the local re-encoding methods, however, will be a significant loss of speed, but this is expected.

2.1 Optimal algorithms

The quest for a "*good*" optimal K-anonymity solution has stopped since it was shown that optimal K-anonymity problem is at the easiest, $NP - hard$. However, there are many optimal algorithms designed earlier in K-anonymity research, which help elucidate the problem. The first optimal algorithm was published in [15]. This algorithm worked by creating a generalization structure, essentially a graph structure, whose edges represent generalizations. Each node in this structure was a database, and from each database is an edge for each of the possible generalizations which can be done. All possible generalizations exist in this structure, thus, the optimal solution must be in the generalization structure as well. Using this structure, a binary search can be done on the levels of the data structure to find the best choice for the generalization of the database. This algorithm is extremely elegant. However, it is very expensive, because there is a level in the data structure which has an exponential number of possible generalizations. Thus, for large databases this algorithm will not be feasible. There are, of course, many other possible optimal algorithms for K-anonymity; however, this happens to be one of the faster algorithms without approximation. We developed an optimal algorithm which has a running time of $|P_A|^{|M|}$ where $|P_A|$ is the cardinality of the permutation set of the attribute set (including all generalizations) and M is the number of members. This algorithm just iterates through all the possible combinations of members at

each level of generalization. The algorithms in [15] was meant to be used on global re-encoding, in our case, the algorithm was design for local re-encoding. Thus our algorithm has a much worst running time, but that is to be expected.

2.2 State of the art algorithms

There are three state of the art algorithms that we would like to discuss. None of the algorithms mentioned here are the algorithms that we have tested. These algorithms are well developed and tested thoroughly in other works, Instead of working with already developed concepts, we decided to evaluate less pronounced methods for K-anonymity, which might yield better results under the right constraints. Also, these algorithms use very specific methods and do not generalize well to the local re-encoding problem.

The first algorithm we will discuss is *Incognito* [16]. This algorithm took advantage of the generalization tree hierarchy which gives a necessary condition for K-anonymity with respect to the current $PIDs$. This necessary condition only works if we are generalizing over the attribute space and not over the cell space. What they realized is that any K-anonymity in a subset of a PID (the tree underneath this new level of generalization) implies K-anonymity in the new generalization; and thus, they could scrap parts of the tree that did not hold this property. This result lead to a simple and fast algorithm by pruning. The generalization tree that is used in *Incognito* is similar to the one used in [15], but so many of the branches are cut that it does not take as long as finding the optimal solution.

The next algorithm is *Mondrian* [17], which is a clustering algorithm cutting a multi-dimensional space (this space represents the permutation space of the attribute set) into pieces. Each attribute is considered a dimension of the space. The algorithm continues to cut the space (at each iteration) into smaller and smaller pieces until any new cut will cause a set to be smaller than K in size. This method works well and returns good results in terms of amount of generalization and average set size. However, it relies on global re-encoding because the change that occurs when an attribute is cut affects all members. Another nice result of this algorithm is that the number of points in any given subspace can be proven to be smaller

than $2d(k-1)+m$, where d is the size of the possible $PIDs$ and m is maximum number of members with the same PID prior to running the algorithm. The results of *Mondrian* are much better than those of *Incognito* in terms of information loss.

The last algorithm in the state of the art algorithms section is a genetic programming algorithm. Interestingly enough, many dismissed genetic algorithms because they converged slowly, and the space of possible choices (referring to the pool for each generation) is so large that it is infeasible to wait for convergence. However, when considering the problem from a different perspective (as done in [14]), it is possible to reduce the permutation space and prune possible results which are unnecessary or maligned. What was done to make genetic programming a viable algorithm choice for K-anonymity, was to reduce the permutation space by focusing on the attribute permutation space and not on all possible combinations of attribute values. The numbers of combinations of attribute values can be huge, and therefore, by focusing on the attribute permutation space, i.e. global re-encoding, they were able to solve a more tractable problem. The work in [14] is based on [18], where their genetic algorithm used enough bits, such that, each attribute value was allocated a bit. This representation can have many illegal permutations, any of which must be pruned from the generations (referring to the iterations of a genetic algorithm). Thus, by reducing the amount of information to just the attribute permutation space and pruning generations lead to a significant speed up. This algorithm outperformed all the other genetic algorithms designed for K-anonymity, but, as of yet, there has not been a comparison to clustering algorithms such as *Mondrian*.

2.3 Approximation algorithms

As mentioned earlier, optimal K-anonymity is a $NP-hard$ problem, thus there has been a lot of work on approximating the optimal solution. The first approximation we will mention is the one shown in [3]. Here, they took advantage of the set cover problem, and its relationship to the members of the database. The idea is, if we can find a set covering which covers all the members of the database, such that the average diameter (to be formally defined shortly) is minimized, then such a cover is close to the K-optimal solution. This problem already has an approx-

imation ratio. The only problem with the set covering problem is sets are allowed to overlap (K-sets in an optimal K-anonymous solution must be disjoint). The way this problem was solved was allowing sets to have a cardinality of at most $2k-1$. If two sets overlap, join the sets together if they are the same size, else if one is larger than the other, move some of the points from the larger set to the smaller one. This algorithm has an approximation ratio of $O(k \ln k)$. The procedure for finding the K minimum diameter sum is $O(\ln k)$-approximation, and the reduction step (the step outlined in the previous sentence) is $O(k)$-approximation. All of these concepts where proven in [3]. The only problem with this approximation is as K increases so does the running time of K minimum diameter sum as it is super polynomial in running time, however, in [3] a polynomial running time methods was shown. Also, this approximation is only defined for cell suppression, which is a sub-problem of local re-encoding. Local re-encoding allows for cell generalization, not suppression.

A better approximation for cell suppression was found in [13], where they developed a weighted graph out of the members of the database. Then they, computed a forest using the graph, where each tree has at least K vertices. Also, the maximum number of nodes per tree was at most $3k-3$. The approximation ratio for this algorithm is $O(K)$, a much better result than the previous approximation ratio. This algorithm can be upgraded to handle cell generalization instead of just cell suppression. The algorithm is completely described in [19]. This algorithm was also shown to be an $O(K)$-approximation for cell generalization. The cell generalization algorithm works by changing the weight metric put on each edge in the graph to allow for more variation. The weight metric they used is similar to the ancestor metric. Also, in [19] it was shown that the best this method of approximation can do is $O(K)$, thus,they gave an imapproximability result for this specific construction. The way that this was shown, was assume the graph structure, as defined in [19], can be optimized to be better than the initial weighted graph. They were able to prove the solution still converges to the same forest. This is roughly what is described in [19].

Algorithm: K-anonymization by clustering in attribute hierarchies (KACA)

1: Form equivalence classes EQ from $M \in D$
2: **while** $\exists\ eq_i \in EQ \mid eq_i < k$ **do**
3: Randomly choose an $\{eq_i \in EQ | eq_i < k\}$
4: Find $d(eq_i, eq_j)$ for each $eq_j \in EQ$ where $i \neq j$
5: Find $eq_j \in EQ$ where $i \neq j$ s.t. $d(eq_i, eq_j) = \min_{k \neq i} d(eq_i, eq_k) \forall eq_k \in EQ$
6: Generalize eq_i and eq_j into eq'_i
7: **end while**

Figure 2.1: This is one of the algorithms we have analyzed, taken from [1]

2.4 Implemented algorithms

Now that we have described the state of the art algorithms, we can discuss the algorithms we will be using for our test. The choice of these algorithms, as stated before, was for their simplicity and potential. Neither algorithm was implemented as they were written in [1, 2]. We changed their design, such that we could add additional constraints and used entropy as an information loss metric. We are including the pseudo-code for both of the algorithms, allowing one to use the pseudo-code as a reference for the rest of this thesis.

2.4.1 Algorithm I from [1]

The algorithm in Figure 2.1 is the algorithm included from [1]. It simply chooses a random pivot and first checks to see if this pivot is of cardinality less than K. If it is, then it moves to the next step, else repeat. Then, it checks the pivot against all other equivalence classes to see which one has the cheapest metric cost. Once it finds the best choice, the two classes are generalized, such that they are now one class. We removed the constraint from the following line of the algorithm, " *Randomly choose an* $\{eq_i \in EQ | eq_i < k\}$", to see how it would affect its performance. We allowed any equivalence class to be chosen as the pivot instead of limiting the pivot to classes of size less than K. As you can see, this is an extremely simple algorithmic frame work. The running time of this algorithm is $O(n \log n + |E|^2)$.

2.4.2 Algorithm Bottom Up Method from [2]

Algorithm: Bottom up method
Input: A database D, parameter K, weights on attributes, hierarchies on categorical attributes.
Output: A K-anonymous database D

1: Form equivalence classes EQ from $M \in D$
2: **while** $\exists\ eq_i \in EQ \mid eq_i < k$ **do**
3: **for** $\forall eq_i \in EQ$ s.t. $eq_i < k$ **do**
4: Find $eq_j \in EQ$ s.t. $eq_i \neq eq_j$ and $d(eq_i, eq_j) = \min_{i \neq k} d(eq_i, eq_k) \forall eq_k \in EQ$
5: Generalize eq_i and eq_j into eq'_i
6: **end for**
7: **for** $\forall eq_i \in EQ$ s.t. $eq_i > 2k$ **do**
8: split eq_i into $\lceil \frac{eq_i}{k} \rceil$ equivalence classes s.t. each has a size of at least k.
9: **end for**
10: **end while**
11: Generalize the remaining equivalence classes.

Figure 2.2: This is the second algorithm we have analyzed and have taken parts of to use for creating new constraints [2]

The general idea behind the algorithm in Figure 2.2, is: at each step, generalize equivalence classes that are less than K in size, and then, break equivalence classes which are larger than 2K in size into smaller classes. We adjusted this algorithm by adding the random pivot concept found in [1]. The first for loop relied on the ordering of the equivalence classes which we found to be unnecessary. Thus, we not order and got rid of the first inner for-loop. This allow for more variance run after run. We could then run the algorithm multiple times and take the best result. Also, this algorithm suffers from equivalence classes building up in size rapidly; as it is possible for the classes to double in size every iteration. Making the equivalence class choice random would help mitigate the speed at which equivalence classes grow.

3. Theory

Before we discuss the algorithms that we analyzed, or details of the algorithm that we designed, we will discuss some of the additional theory used for our $O(K)$-approximation based on the work of [3]. In [3] the technique used to find a K-anonymous solution by cell suppression consist of finding the diameter of the K-anonymous equivalence classes that are picked.

3.1 Diameter and minimum pairwise metric sum

Definition 2. *let $S \subseteq \Sigma^A$ and $u, v \in \Sigma^A$. Σ^A represents all possible members that can be made by the given attribute set. The distance between u, v is*

$$d(u, v) := |\{j : u[j] \neq v[j]\}|$$

. The diameter of an equivalence class is

$$d(S) := max_{u,v \in S} d(u, v)$$

What this definition means is, for each attribute of the database, find out if the two given members differ in that attribute. The sum of the differences gives the distance between the two members. The maximum pairwise distance in the given set of members is the diameter. There is a similarity between the concept of diameter and minimum pairwise metric sum (when distance is used as the metric), however, minimum pairwise metric sum is based on an average. An average does not give you a tight bound on the variance within the set of values unless more work is done then just computing the sum.

We initially used the concept of pairwise metric sum to find a set of points around each member such that the pairwise metric sum was minimized. The problem was defined as follows.

Definition 3. *Given a set of members M where each $M_i \in M$ is a set of $|A|$ values, where A is the set of attributes $A = \{A_1, A_2, \cdots, A_n\}$. Each A_i has n_i possible*

values. Thus, each $M_i = \left\{A_1^{k_1}, A_2^{k_2}, \cdots, A_n^{k_n}\right\}$ *where* k_i *is a value between* $(1 \ldots n_i)$. *We are also given a function* $f : A^n \times A^n \to [0, 1]$ *where* A^n *is all possible members. We want to find* $S_{M_i}^k$ *which is the set of points of cardinality* k *with a minimum pairwise metric sum using* f *as a metric The set* $S_{M_i}^k$ *must contain* M_i.

3.2 Bi-criteria compact location problem

Finding this minimum pairwise metric sum for an arbitrary metric is at the minimum a $NP - HARD$. However, we can find a solution to this problem using the bi-criteria compact location problem [20]. This can be done, as long as, our metric obeys the triangle inequality. However, the diameter and the distance used in [3] does not obey the triangle in equality. This can be fixed by giving an ordering to the attributes. When we are trying to find, the set of $k-1$ members around each M_i which minimize the pairwise metric sum, we are also looking for those points to be close to M_i. If the distance to M_i was not a constraint we could possibly end up with points that are very close together, but far from M_i. Thus, we can use the following problem definition from [20, 21] to solve this type of minimum pairwise metric sum:

Definition 4. [*Diameter constrained minimum average placement problem (DC-MAP)*] ***Input:*** *An undirected complete graph* $G = (E, V)$ *with two positive edge weights functions* $c, d : E \to \mathbb{Q}^+$, *an integer* $2 \leq k - 1 \leq |M|$ *and a number* $\Omega \in \mathbb{Q}^+$ ***Output:*** *A set* $P \subseteq V$ *with* $|P| = k - 1$, *minimizing the objective function*

$$S_c(P) = \sum_{v,w \in P,\ v \neq w} c(v, w)$$

Subject to the constraint

$$D_d(P) = \max_{v,w \in P,\ v \neq w} d(v, w) \leq \Omega$$

The algorithm used to solve this problem is a $(2 - \frac{2}{k}, 2) - approximation$ of the optimal minimum pairwise metric sum, which is shown in Figure 3.1.

Procedure HEUR-FOR-DIA

1: $G' := bottleneck(G, \delta_d, \Omega)$
2: $V_{cand} = \{v \in G' \mid deg(v) \geq p - 1\}$
3: **if** $V_{cand} = \emptyset$ **then**
4: RETURN FAIL
5: **end if**
6: Let $best := +\infty$
7: Let $P_{best} := \emptyset$
8: **for** $v \in V_{cand}$ **do**
9: Let $N(v)$ be the set of $P - 1$ nearest neighbors of v in G with respect to δ_c
10: Let $P(v) := N(v) \cup \{v\}$
11: **if** $M_{\delta_c}(P(v)) < best$ **then**
12: $P_{best} := P(v)$
13: $best := M_{\delta_c}(P(v))$
14: **end if**
15: **end for**

Figure 3.1: This is the algorithm for computing a $(2-\frac{2}{k}, 2)$-approximation of TI-DC-MAP from [21]. The syntax and the definition of bottleneck graph of a complete graph is in [21].

3.2.1 Using Bi-criteria problem on pairwise metric sum

If we wisely choose the functions c, d, we can end up solving the minimum pairwise metric sum for a given M_i and a $p = K - 1$. This can be done by removing m_i from M prior to running the procedure in Figure 3.1, and then, using c (which should be the given information loss metric) to solve for P_{best}. Also, the function d can be the maximum of the two distances from M_i to the points v, w (refer to Def. 4 for the meaning of v, w). The limiting value Ω can be found by taking the point M_i, and finding $\max_{w \in I_{M_i}} c(M_i, w)$, where I_{M_i} is the set of points closest to M_i by c (the information loss metric). This would give us an approximation of the minimum pairwise metric sum to $(2 - \frac{2}{k}, 2)$-approximation of the optimal solution, and in this case, both parts of the approximation are necessary. This approximation algorithm relies on the information loss metric being a metric (most importantly, obeying the triangle inequality). Also, if the solution to the algorithm is worse than I_{M_i}, we should pick I_{M_i} as the solution instead of the algorithm's output. We mentioned above that diameter and distance are not metrics because they do not obey the triangle inequality. However, if we make the assumption that there

exist an ordering of the attributes, the distance and diameter will obey the triangle inequality. It would have to be shown, however, that these changes do not change the work done in [3]; we will be using some parts of their algorithm.

Definition 5. *let $S \subseteq \Sigma^m$ and $u, v \in \Sigma^m$. Also, for each $A_i \in A$ assigned a number from $1 \ldots |A|$. The distance between u, v is*

$$d'(u,v) := \sum_{j}^{|A|} 2^j \cdot f(u[j], v[j])$$

$$f(u_j, v_j) = \begin{cases} 1 & u_j \neq v_j \\ 0 & u_j = v_j \end{cases}$$

The diameter of an equivalence class is

$$d(S) := max_{u,v \in S} d'(u,v)$$

3.2.2 Using Bi-criteria problem on diameter

This same method can be used to find the minimum diameter around the member M_i. However, a definition for distance and diameter is needed. To do so, we would need to carefully pick the functions c, d, which can be done in a similar fashion to the way we picked c, d for the pairwise metric sum approximation. Also, when finding the minimum diameter, the algorithm needs to run twice with two different values for Ω in order to find the best solution. The two values of Ω we would need to compute are, Ω_I, which is constrained to $\max_{w \in I_{M_i}} c(M_i, w)$, and Ω_{GD}, which is constrained to the largest diameter in I_{M_i}. In the first case Ω_I, we are constraining Ω to the distance from M_i, and in the second case Ω_{GD}, we are constraining Ω to the maximum diameter in I_{M_i}. The reason for these constraint will become more apparent when we discuss the functions c and d.

For the functions c and d, we need to make sure that they hold under the triangle inequality. Thus, it is necessary that we use the second definition of distance and diameter. We can then define $c(u,v) = \max\{d'(M_i, u), d'(M_i, v), d'(u, v)\}$. Thus, if we try to minimize the maximum distance between the points in P_{best}, we

are at the same time attempting to minimize their distance to M_i. Because we are taking the maximum of the three distances, we can make a guarantee on the largest distance between the points in P_{best}, as well as, the largest distance to M_i. The function is defined as $d(u, v) = \max\{d'(M_i, u), d'(M_i, v)\}$. This is done for all pairs of members v, w, since the original problem puts constraints on edges. Thus, the $(2 - \frac{2}{k})$-approximation algorithm of the minimum of the maximum diameters is going to bring the points closer to M_i, and the second 2-approximation algorithm constraining the distance to M_i will make the over all cost no worse than 2Ω.

this choice of functions c an d for solving the minimum pairwise metric sum in a bi-criteria compact location problem setting allows us to solve the minimum diameter to within (2)-approximation. This is the case because we have minimized the maximum diameter whilst solving for the minimum pairwise metric sum. Also, we have constrained the distance between the the set P_{best} and M_i, which is important for the diameter measurement. The set we are trying to find is supposed to be close to the initial point (M_i) in the first place.

Solving the problem stated above for the values of Ω_{GD} , Ω_I, and comparing solutions with the set I_{M_i}, we can take the set with the minimum of the maximum diameters as our solution. This derivation above gives a much better approximation than the solution in [3]. However, we made an important assumption– we assumed an ordering of the attributes exist. Without this assumption, diameter would not hold under the triangle inequality, and the problem would be much harder. As stated in [20], bi-criteria compact location problem is much harder for non-triangle inequality functions. There are no algorithms for the problem without the triangle inequality constraints other then brute force.

3.3 Partitions, Reductions, and Approximation from [3]

We have already discussed the concept of distance and diameter, as well as, the changes we have made to allow for a better approximation. To use the following theorem stated in [3], we must show how the bi-criteria compact location problem lead to a solution for K-minimum diameter sum.

First, we need to give the definition of a (k_1, k_2)-cover prior to discussing the

theorem.

Definition 6. *Let $k_1, k_2 \in \mathbb{N}$, a (k_1, k_2)-cover is a collection of subsets of the database $DB(M, A)$. The subsets are $\{S_1, S_2, \ldots, S_l\}$ s.t. $k_1 \leq |S_i| \leq k_2$ for each S_i and for each $m_i \in M$ $\exists S_i$ s.t. $m_i \in S_i$*

If we were to run the algorithm for bi-criteria compact location for each member we would get a set of K-sets. We can make $k_1 = K$ and $k_2 = 2k - 1$ in the definition of a (k_1, k_2)-cover. Also, being that we have an approximation of a minimum diameter set around each member, we have an approximation of the sum of the $|M|$ sets as well. We need to turn this (k_1, k_2)-cover into a partitioning of the database. The only difference between a partitioning and a cover is that a cover does not necessarily have disjoint sets, while a partition does. In [3], there is a procedure which makes a cover into a partitioning. Using the algorithm in Figure 3.2, we can make a (k_1, k_2)-cover into a (k_1, k_2)-partitioning. The diameter cannot increase by this K partitioning, because we either add a point to the set already containing it, remove a point from a set which can only make the diameter smaller or join two sets together which overlap. It can only decrease from this procedure.

Now to discuss this procedure in terms of bi-criteria compact location, we can use the $(2 - \frac{2}{k})$-approximation algorithm for the diameter to build a (k_1, k_2)-cover. The $(2 - \frac{2}{k})$-approximation algorithm can make a for each member in the database. All of the sets made by this procedure, when put together make a (k_1, k_2)-cover. Thus, to use the work done in [3], we need to make this (k_1, k_2)-cover into a (k_1, k_2)-partitioning. Also, we need to show that thus (k_1, k_2)-partitioning is an approximation of k-minimum diameter sum. This is quite simple because the k-minimum diameter sum is a sum of the sets in the (k_1, k_2)-partitioning. We could find the k-minimum diameter sum given the (k_1, k_2)-cover found by bi-criteria compact location algorithm. As we mentioned before, using the algorithm in Figure 3.2, we can get a (k_1, k_2)-partitioning without increasing the diameter of any of the sets in the (k_1, k_2)-cover. Thus, we end up with a k-minimum diameter sum with a $(2 - \frac{2}{k})$-approximation. These results can be used in the lemma and corollary from [3] to give an approximation of the K-anonymity problem. It is not a requirement that the function used should not obey the triangle inequality for their k-minimum

Reduce(Π):

1: Look for $S_i, S_j \in \Pi$ and $m_i \in M$ such that $m_i \in S_i \cap S_j$
2: If found do one of the following:
3: **if** $S_i > S_j$ && $m_i \in S_i \cap S_j$ **then**
4: remove m_i from S_i and put in S_j
5: **else if** $S_i < S_j$ && $m_i \in S_i \cap S_j$ **then**
6: remove m_i from S_j and put in S_i
7: **else**
8: Replace S_i and S_j with $S_i \cup S_j$ in Π
9: **end if**
10: Return the new Π

Figure 3.2: This is the procedure used to rid any (k_1, k_2)-cover of non-disjoint sets.

diameter sum lemma to be applicable. However, the solution they give is more general than the solution discussed here. It should be noted that it is very often the case that attributes in the database will have an ordering of importance. Thus, our assumption is not unlikely. The corollary which gives us an $O(k)$-approximation of K-anonymity is as follows from [3]:

Corollary 1. *Let $\alpha \geq 1$ and let Π be a $(k, 2k-1)$-partition with a diameter sum at most α times the optimal k-minimum diameter sum. Then the algorithm that anonymizes each $S \in \Pi$ by cell suppression is a $3\alpha k$-approximation to optimal k-anonymity.*

Using Corollary 1, we can set $\alpha = 2 - \frac{2}{k}$ which would give us a $(6k-6)$-approximation or an $O(k)$-approximation of the optimal k-anonymity. We will write the precise procedure in Chapter 5.

4. METHODS OF PROCEDURE

Here we will describe how we implemented the testing database, tested the algorithms, and what statistics we kept on the different algorithms as we tested them. We will first describe the details of the database implementation, i.e., the structure of the database and how members for the database were generated. Then, we will describe how each of the algorithms was tested, how many test where done and what each of the test cases was testing. Finally, we will describe what statistics where used. These statistics will be presented in the results chapter.

4.1 Implementation of Testing Database

The testing database software was designed to make databases of size n, where each member of the database has random values assigned to each attribute. The value were chosen from a pool of attribute values included in a database loading file (which we will discuss shortly). The attributes are broken into three categories, similarly to what was discussed in the introduction. The categories are the UID (uniquely identifying attributes), PID (pseudo-identifying attributes), DATA (data attributes). Only the PID and DATA attributes where randomly assigned. The UID attributes were assigned as an ordering of all the possible permutations. That is, they are assigned them as we made the members, such that, each member had a unique UID. There can only be as many members in the database as there are permutations in the UID, since the UID cannot repeat.

4.1.1 Database Loader File

The file in Figure 4.1 is an example of the database loading file. This file is used to make a .csv, which holds the members of the database and a header file. The header file holds important information used to build the database held in a .csv. We will discuss how each line in the file is used. First, the line beginning with **FILE** provides the prefix to be attached to the .csv and .header files. The next line **DISTRO** provides information about the distribution used to pick random

```
1  FILE,ALGORTESTTWODB(large)
2  DISTRO,poisson,LVAL:2.5
3  A,UID,20000
4  B,C,D,E,PID,6,10,12,16
5  F,DATA,2
6  ATTR,A,B,C,D,E,F
7  MAPS
8  B,b0,b1,b2,b3,b4,b5
9  C,c0,c1,c2,c3,c4,c5,c6,c7,c8,c9
10 D,d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11
11 E,e0,e1,e2,e3,e4,e5,e6,e7,e8,e9,e10,e11,e12,e13,e14,e15
12 F,f1,f2
```

Figure 4.1: This is an example database loading file named "*ALGORTESTTWODB-Large-Poisson-2-5.loader*" where the title of the file represents the size of the database and the distribution which is used. A detailed explanation of each line in the file is provided in 4.1.1.

attribute values. Next, there will be lines providing which category an attribute is in UID, PID, DATA, and the size of each attribute. The lines providing this information have the following format, { attributes } , { UID or PID or DATA } , { sizes } . Following these lines, there is a line starting with **ATTR**, which provides a list of all the attributes. Finally, following the word **MAPS**, each line provides the possible values for the attributes. The number of attribute values per attribute should match the size provided above, except for the UID attributes where the size provided is set to the maximum number of permutations.

4.1.2 Database header file

The header file provided in Figure 4.2 has the following layout. The first line of the header file starts with **NOA**, followed by a list of the different attributes in the database. This list of attributes needs to be in alphabetical order. Being that the header file is computer generated, this is always be the case. The next line in the file starts with **NOM**, which is the number of members in the .csv file corresponding to this header file. The number following **NOM** is the number of members. The next line in the file is the **RANGE**, which provides the size of each of the attributes. The format for this line is as follows {Attribute}:{size}. There should be one of the {Attribute}:{size} structures per attribute in the database, including the attributes

```
1  NOA,A,B,C,D,E,F
2  NOM,5000
3  RANGE,D:12,E:16,F:2,A:20000,B:6,C:10
4
5  MAPS,D:d0:0,D:d11:11,D:d1:1,D:d2:2,D:d3:3,D:d10:10,D:d9:9,D:d8
       :8,D:d5:5,D:d4:4,D:d7:7,D:d6:6
6  MAPS,E:e15:15,E:e14:14,E:e13:13,E:e12:12,E:e11:11,E:e9:9,E:e10
       :10,E:e8:8,E:e7:7,E:e6:6,E:e5:5,E:e3:3,E:e4:4,E:e1:1,E:e2:2,E
       :e0:0
7  MAPS,F:f1:0,F:f2:1
8  MAPS,B:b0:0,B:b1:1,B:b3:3,B:b2:2,B:b5:5,B:b4:4
9  MAPS,C:c1:1,C:c2:2,C:c0:0,C:c6:6,C:c5:5,C:c4:4,C:c3:3,C:c9:9,C:
       c8:8,C:c7:7
10 ALLOC,UID:A,PID:B,PID:C,PID:D,PID:E,DATA:F
```

Figure 4.2: This is an example of a database header file named "*ALGORTESTTWODB(large) (poisson:2.5).header*". The prefix appended to the .header can also be found on a .csv file. The .csv file is the database pairing with the header file. Included in the header file name is which loader file made it, the size of the database and the distribution that made the database. The header file layout is discussed in 4.1.2.

part of the UID. After the **RANGE** there is a **MAPS** line for each attribute, the formatting is as follows, {Attribute}:{Attribute value}:{Integer}. This is done for each of the attribute values. They are essentially being mapped to the natural numbers. For continuous attributes, it is assumed that the values have already been broken into different categories/ranges. The final line in the file is **ALLOC**, which states what category each of the attributes belong to, either UID ,PID , or DATA. For all the test done in this thesis, there is only one binary data attribute. We used this simplified $DATA$ attributes because no test were done on the database's $DATA$ attributes. However, if you look at some of the K-anonymous solutions, it is evident that security flaws can exist when the $DATA$ attributes are not evaluated. However, this was not the focus of this thesis.

We will not give a section specifically to discussing the .csv file, but we will make a note pertaining to the layout. The separating character in all the database csv files are commas. Also, strings need to have quotation marks around them. Other then this, there are no special definitions needed for the .csv files.

4.1.3 Probability Distributions

The six probability distributions used for testing the algorithms, are Gaussian, uniform, log-Gaussian, Poisson with $\lambda = 1$,Poisson with $\lambda = 2.5$, Poisson with $\lambda = 4$. Each random distribution was used on a single attribute, not on the set of all attributes. Essentially, the distribution was defined on the attribute values, not on the permutations. Thus, the random members are a finite sum of random variables of the given distribution. For each of the distributions tested, a sum of random variables defined on that distribution is, yet another random variable defined on that distribution , thus this method of randomizing the members did not change the expected outcome. The reason we chose these distributions was that they represent difficult cases for a K-anonymizing procedure to handle. For example, if we have a large permutation space for the pseudo-identifier, then an uniform distribution is less likely to pick members which are close together. Also, if we are using a Poisson with $\lambda = 1$, there will be a lot of members with very similar pseudo-identities and therefore, the algorithm has to watch for over-generalizing. We did not use a λ higher than four because as lambda increases the distribution converges to a discrete Gaussian.

4.1.4 Generalization trees

For some of the information loss metrics, mainly ancestor metric and GCP metric, a generalization tree is required. We have implemented the tree as a file of the form shown in Figure 4.3. The generalizations represented in Figure 4.3 are of the form { Attribute value 1 | . . . | Attribute value n }, and can be read as attribute value 1 or attribute value 2 etc.. This is the format all generalization will take, regardless of which information loss metric was used. In the case of generalization trees, the search space for generalization is limited. The tabbed values underneath the untabbed attribute values are the children of the parent value. The children need to be one tab away from the left margin. If an attribute in the tree does not have braces, it is not generalized. The database testing software takes count of the children of a parent, and the children know their parent, thus allowing an inverse traversal of the tree (starting at the leaves). One file of the type in Figure 4.3 must

```
1  {e0|e1|e2|e3|e4|e5|e6|e7|e8|e9|e10|e11|e12|e13|e14|e15}
2          {e0|e1|e2|e3|e4|e5|e6|e7}
3          {e8|e9|e10|e11|e12|e13|e14|e15}
4  {e0|e1|e2|e3|e4|e5|e6|e7}
5          {e0|e1|e2|e3}
6          {e4|e5|e6|e7}
7  {e8|e9|e10|e11|e12|e13|e14|e15}
8          {e8|e9|e10|e11}
9          {e12|e13|e14|e15}
10 {e0|e1|e2|e3}
11         {e0|e1}
12         {e2|e3}
13 {e4|e5|e6|e7}
14         {e4|e5}
15         {e6|e7}
16 {e8|e9|e10|e11}
17         {e8|e9}
18         {e10|e11}
19 {e12|e13|e14|e15}
20         {e12|e13}
21         {e14|e15}
22 {e0|e1}
23         e0
24         e1
25 {e2|e3}
26         e2
27         e3
28 {e4|e5}
29         e4
30         e5
31 {e6|e7}
32         e6
33         e7
34 {e8|e9}
35         e8
36         e9
37 {e10|e11}
38         e10
39         e11
40 {e12|e13}
41         e12
42         e13
43 {e14|e15}
44         e14
45         e15
```

Figure 4.3: This is an example generalization tree file. A description of the file is provided in 4.1.4

be included per attribute. The files should be in a designated directory. of that type

4.1.5 Running the database testing software

There are in total ten different tasks the database software can perform. Here we will provide an itemized list with each task, and the command format for that task (as it would be typed on the command line).

- Creating a database from a loader file.

  Command: 0 { number of members } (filename).loader

- Read a database and output statistics.

  Command: 7 (filename) (directory for gen tree)

- Read a directory of databases, and output a file with statistics for each database in the directory. The output file is a .csv.

  Command: 8 (directory of databases) (directory for gen tree) (output file)

- Build a database with a prefixed, hard coded size. The three possible sizes are small, medium and large (50,500,5000).

  Command: 9 (filename)

- Runs a comprehensive test of all the databases in a given directory. The test iterates through the following parameters: five different values of k, for all three information loss metrics, and for each of the four algorithms

  Command: 10 (directory of databases) (directory for gen tree) (directory for K-anonymized databases)

- Runs K-anonymity using the algorithm "*only smallest sets picked without breaking sets that are too large*" on the given database, using the given information loss metric and value for K. The accepted information loss metrics are "*Entro*", "*AnTree*" and "*Discern*".

  Command: 1 (Filename) { k } { metric } (directory for gen tree)

- Runs K-anonymity using the algorithm "*Any set can be picked without breaking sets that are too large*" on the given database, using the given information loss metric and value for K. The accepted information loss metrics are "*Entro*", "*AnTree*" and "*Discern*".

  Command: 2 (Filename) { k } { metric } (directory for gen tree)

- Runs K-anonymity using the algorithm "*only smallest sets picked and sets that are too large are broken*" on the given database, using the given information loss metric and value for K. The accepted information loss metrics are "*Entro*", "*AnTree*" and "*Discern*".

  Command: 3 (Filename) { k } { metric } (directory for gen tree)

- Runs K-anonymity using the algorithm "*Any set can be picked and sets that are too large are broken*" on the given database, using the given information loss metric and value for K. The accepted information loss metrics are "*Entro*", "*AnTree*" and "*Discern*".

  Command: 4 (Filename) { k } { metric } (directory for gen tree)

- Runs K-anonymity using the algorithm "*Optimal*" on the given database, using the given information loss metric and value for K. The accepted information loss metrics are "*Entro*", "*AnTree*" and "*Discern*". This was not used in the thesis since it is extremely expensive to run on small data sets. There are better optimal algorithms that can give some results.

  Command: 5 (Filename) { k } { metric } (directory for gen tree)

- This function tests correctness of the generalization trees' format. The functionality is implemented allowing users to debug problems with files, such as missing tabs.

  Command: 6 (Filename) (directory for gen tree)

4.2 Information loss metrics

The three information loss metrics implemented in the testing software, are Ancestor metric, GCP (global certainty penalty), and Entropy. The first two metrics require a generalization tree so the metrics can be implemented correctly, while the Entropy method we designed does not. However, this might not always be desired as some generalizations might be less advantageous than others and thus, we would not want these generalizations in the database. We will go over the implementation of each metric.

4.2.1 Ancestor metric

The ancestor metric which we have implemented is defined as the following:

Definition 7. *This is defined on a database DB(M,A) where M is the set of members and A is the attribute set. Given an attribute value A_i^j where $A_i \in A$ and j is the number of the value including generalizations, the ancestor metric of the attribute A_i is*

$$An_{A_i}(A_i^j) = \frac{\max_{l_i \in l} \ toleaf(A_i^j, l_i)}{\max_{l_i \in l} \ toleaf(A_i^{suppressed}, l_i)}$$

where $toleaf(\cdot,\cdot)$ finds the distance between the first value to the second value which is a leaf and $A_i^{suppressed}$ is the suppressed value for this attribute.

Using Def. 7, which only computes the ancestor metric for a single attribute, we can compute the ancestor metric for two specific members (or equivalence classes).[3]

Definition 8. *Given members M_i and M_j the ancestor difference between the two members is*

$$AnD(M_i, M_j) = \frac{\sum_{A_k \in A} \ An_{A_k}(gen(M_i[k], M_j[k]))}{|A|}$$

where the function $gen(\cdot,\cdot)$ takes two attribute values and generalizes them to a single attribute.

This metric is computed after picking the random pivot value. It is computed by comparing all the equivalence classes that are not the chosen pivot to the chosen pivot, using the result of these comparisons, equivalence class with the least information loss when generalized with the pivot is chosen.

4.2.2 Global certainty penalty

This metric was implemented exactly as it was described in the introduction. The generalizations for the member's attributes where chosen from the generalization trees, thus our implementation did not used the entire generalization space.[4]

[3]Note on syntax, we will use $M_i[j]$ to find the value of the j attribute for member M_i.

[4]In our implementation the value of the weights in the $\sum w_i$ is the same.

4.2.3 Entropy

This is the new metric for measuring information loss which we developed. The standard definition for information entropy is as follows

Definition 9. *Information entropy of a discrete random variable X is*

$$H(X) = \sum_{i=1}^{n} -P(X = x_i) \log_b P(X = x_i)$$

In the entropy equation, b is usually taken to be 2. This has to do with binary system being consider the basis of informational content. In our information loss metric, we used $|A_i|$ as the base for the logarithm. Doing so allows us to measure the information content of the attribute. Information content can also be described as, how close is an attribute to the uniform distribution if we where to consider the attribute a random variable. Thus, when too many equivalence classes are highly generalized the information content would go down. We should consider this case more thoroughly. When the classes are generalized to the point where most of the attributes values in the PID are near suppression, many of the equivalence classes will also be larger than 2K. This problem can be mitigated by keeping information about values no longer present the PIDs of equivalence classes. Entropy drops when the random variable has a skewed distribution; a drop in entropy means an increase in information loss. The reason entropy drops is new values where added to the attribute, thus increasing the size of the attribute, but old values have a lower probability of showing up.

Now the question to ask is, how do we find the distribution of the attributes? If we had the ability, it would be optimal to find the distribution in the overall population, in other words, the precise distribution. What we have done instead is, find the distribution of the attribute values at each iteration of the algorithm. Thus, as the database is generalized, the skewness goes up. The assumption is only necessary for the first iteration when the attributes are not generalized. This assumption is valid when dealing with large databases. The larger the database, the smaller the error between the database's distribution and the distribution of the attribute value in the overall population. This assumption allows us to compute

the entropy per attribute, and then, the entropy for a given generalized equivalence class. Another nice property of entropy is that entropy is sensitive to large groups of members. Thus, it will naturally avoid members from clumping together, which can cause unnecessary information loss.

One problem with entropy is that the function is non-monotonic when used in an iteration. However, one way we have perturbed this behavior was to leave in attribute values that have a count of zero. As mentioned earlier, this helps a lot with other aspects of iterative K-anonymity. We implemented the entropy metric without using a generalization tree. Implementing entropy with a generalization tree has some negatives. For instance, the user will not be able to imply which generalizations are less advantageous . There are ways to solve these problems, for instance, adding weights to certain attributes or using conditional entropy. However, we have not explored these concepts and will discuss them further in future work.

5. RESULTS

5.1 Algorithms

We will now discuss each of the algorithms in details. We will include the pseudo code for each algorithm for ease of reference.

5.1.1 O(K)-Approximation Algorithm

The algorithm outlined in Figure 5.1 is the algorithm we designed combining methods developed in [3, 21]. The algorithm first runs the bi-criteria compact location procedure [21] (the bi-criteria compact location algorithm is run on each member), and then, using the $(k, 2k-1)$-cover returned, we run the second and third phase of [3]. The final output of the procedure is a new K-anonymous database by

Procedure K-anon(D k)

1: Make a new database D' which is the same as D
2: $\Pi := \emptyset$
3: **for** $M_i \in M$ where M is the member set of D' **do**
4: Find functions c_{M_i}, d_{M_i} for computing distance
5: $I_{M_i} := \min^{(k-1)}_{m_j \in M \mid m_j \neq m_i} d'(m_i, m_j)$ this is using the metric version of distance.
6: Find $\Omega_m := \max_{I_j \in I_{M_i}} d'(I_j, m_i)$
7: Find $\Omega_I := \max_{I_i, I_j \in I_{M_i}} d'(I_j, I_i)$
8: $P_m := HEUR - FOR - DIA(\Omega_m, k, G(M - m_i), c_{M_i}, d_{M_i})$
9: $P_I := HEUR - FOR - DIA(\Omega_I, k, G(M), c_{M_i}, d_{M_i})$
10: $P_{fin} := \min_{P_I, P_m, I_{m_i}} \{ \max_{p_i, p_j \in (P_m \cup m_i)} d'(p_i, p_j), \max_{p_i, p_j \in (P_I \cup m_i)} d'(p_i, p_j), \max_{p_i, p_j \in (I_{M_i} \cup m_i)} d'(p_i, p_j) \}$
11: $\Pi = \Pi \cup P_{fin}$
12: **end for**
13: **repeat**
14: $\Pi := REDUCED(\Pi)$
15: **until** $\Pi := REDUCED(\Pi)$
16: For each $S \in \Pi$ input the minimum number of suppression in to the PID of the set S for it to be K-anonymous.
17: Put each of the sets in Π into D' and return D'

Figure 5.1: This is the our K-anonymization algorithm using some of the techniques from [3].

Algorithm: K-anonymization by minimum set size clustering and without breaking large sets

MinSetNoBreak(D,k)

1: Form equivalence classes EQ from $M \in D$
2: **while** $\exists\ eq_i \in EQ \mid eq_i < k$ **do**
3: Find $k_{min} := \min_{eq_i \in EQ} |eq_i|$
4: Randomly choose an $\{eq_i \in EQ| \ |eq_i| = k_{min}\}$
5: Find $d(eq_i, eq_j)$ for each $eq_j \in EQ$ where $i \neq j$
6: Find $eq_j \in EQ$ where $i \neq j$ s.t. $d(eq_i, eq_j) = \min_{k \neq i} d(eq_i, eq_k)\ \forall eq_k \in EQ$
7: Generalize eq_i and eq_j into eq'_i
8: **end while**

Figure 5.2: This algorithm is very similar to the algorithm from [1]. However we are only allowing the smallest equivalence classes to be used as the random equivalence class instead of equivalence classes smaller than k.

cell suppression. The running time of the bi-criteria compact location problem procedure is $O(M \log M + M + Mk^2)$. The reason for this worst case running time is that the procedure has to create and check a bottleneck graph (described in [21]), and has to check each of $(k-2)$-sets around a given member. The reduce procedure from [3] take $O(M^3)$ operations and thus, we have a total running time of $O(M \log M + M + Mk^2 + M^3)$. The phase three of the algorithm only changes values to suppressed, and has a running time of $O(M)$ and does not effect the order approximation.

5.1.2 "*Only smallest sets, Do not break large sets*" Algorithm

In the algorithm outlined in Figure 5.2, we decided to only use the smallest of all the equivalence classes when choosing the random pivot. Limiting the algorithm to selecting sets of size k allows larger sets to develop easily. For example, with $k = 10$, if there is a set of size 9 and already some k-anonymous sets of size 12, it is possible for a set of size 21 to develop. This is undesirable, at least in some cases. For example, when average set size is important, large sets would be undesirable. Of course, if all the sets where of size 9, then large sets would be inevitable. However, choosing only the smallest sets mitigates the effect.

The worst case running time of this algorithm is easy to compute because it

is easy to find the worst case. If all the equivalence classes are unique (in the first iteration) and at each iteration all the equivalence classes double in size, then we could expect the iteration to go as follows $M/2, M/4, M/8, \cdots, M/2^k$. Thus, we end up with the equation.

$$\sum_{i=1}^{\log_2 k} \frac{M}{2^i} = M \sum_{i=1}^{\log_2 k} \frac{1}{2^i} = M \left(\frac{1 - 2^{-(\log_2 k + 1)}}{\frac{1}{2}} \right) = M \left(2 - \frac{1}{k} \right) \tag{5.1}$$

The final result can be reduced to

$$M \left(2 - \frac{1}{k} \right) = M \left(\frac{2k - 1}{k} \right) \leq O(M) \tag{5.2}$$

During each iteration, the outer for loop makes a pass through $\frac{M-1}{2^i}$ of the equivalence classes and an inner loop passes through $\frac{M-1}{2^i} - 1$ equivalence classes. This gives us a worst case running time of $O(M) \cdot O(M) = O(M^2)$.

5.1.3 "*Only smallest sets, break large sets*" Algorithm

We added additional checks in our algorithm in Figure 5.3 which were not included in [2]. The algorithm from [2] easily allows sets larger then $2k$ to form. This is also the case with our method, however, we checked if the best possible solution is allowing a large set to enter the database. If you are looking for less information loss their method works better. However, if average K-set size is more important, our method will output K-sets closer to size K. Only after all the equivalence classes have been checked does our method allow an equivalence class with size greater than $2k$ into the solution. When this happens, it implies that the data was very sparse to begin with.

As for the running time of this algorithm, it is not as easy to compute as the previous case, since we have two expensive steps added. If the extra steps are not used, then the worst case running time is still the same. However, if we need to split a set, the algorithm to do so has a worst case running time of $O(k)$. The cost of looking through the equivalence classes is $O(M)$ where M is the number of Members or equivalence classes. However, this can be done while ordering the

Algorithm: K-anonymization by minimum set size clustering and breaking sets larger than 2k

MinSetBreak(D,k)

Form equivalence classes EQ from $M \in D$
while $\exists\ eq_i \in EQ \mid eq_i < k$ **do**
  Find $k_{min} := \min_{eq_i \in EQ} |eq_i|$
  Randomly choose an $\{eq_i \in EQ | \ |eq_i| = k_{min}\}$
  Find $d(eq_i, eq_j)$ for each $eq_j \in EQ$ where $i \neq j$
  $OrderEQ := \emptyset$
  for $eq_j \in EQ$ where $i \neq j$ **do**
    dis $:= d(eq_i, eq_j)$
    $OrderEQ[END] := eq_j$
    $InLineSort(OrderEQ, dis, eq_j)$ # We are keep track of eq classes, smallest first
  end for
  $it := 0$
  $eq_i' := eq_i \cup OrderEQ[it] \cup \{$ Any sets sharing generalization $\}$
  while $\lceil \frac{eq_i'}{2} \rceil >= 2k$ **do**
    $it := it + 1$
    $eq_i' := eq_i \cup OrderEQ[it] \cup \{$ Any sets sharing generalization $\}$
  end while
  if $it = END$ then choose the smallest eq_i'
  while $eq_i' >= 2k$ **do**
    if $eq_i > OrderEQ[it]$ **then**
      Move best fitting members to $OrderEQ[it]$ from eq_i
    else if $eq_i < OrderEQ[it]$ **then**
      Move best fitting members to eq_i from $OrderEQ[it]$
    else
      Don't generalize
    end if
  end while
end while

Figure 5.3: This algorithm uses methods from both [1, 2]. We changed the method by which we reduce the size of the equivalence classes such that, the equivalence classes will only have a size greater than $2k$ if there is no possible way to stop this from occurring.

Algorithm: K-anonymization by Any set in the equivalence classes and without breaking large sets

AnySetNoBreak(D,k)

1: Form equivalence classes EQ from $M \in D$
2: **while** $\exists\ eq_i \in EQ \mid eq_i < k$ **do**
3: Randomly choose an $\{eq_i \in EQ\}$
4: Find $d(eq_i, eq_j)$ for each $eq_j \in EQ$ where $i \neq j$
5: Find $eq_j \in EQ$ where $i \neq j$ s.t. $d(eq_i, eq_j) = \min_{k \neq i} d(eq_i, eq_k)\ \forall eq_k \in EQ$
6: Generalize eq_i and eq_j into eq'_i
7: **end while**

Figure 5.4: Like the algorithm from section Section 5.1.2, except for we do now allow any of the equivalence classes to be picked.

equivalence class. Thus, using the derivation from the previous section, we can have at most an extra $O(k)$ steps per iteration, giving a total running time of $O(M^2k)$.

5.1.4 "*Any set, Do not break large sets*" Algorithm

The algorithm in Figure 5.4 picks a random set as the pivot. We were attempting to minimize information loss by allowing sets that are close together to join. This is also the simplest algorithm we tested and designed since it makes equivalence classes without any constraints. The running time for this algorithm is the same as the algorithm in Section 5.1.2, $O(M^2)$.

5.1.5 "*Any set, break large sets*" Algorithm

We put the information pertaining to this algorithm into the caption of Figure 5.5 due to the relatively small amount of discussion necessary. Most of the context was discussed in Section 5.1.3.

5.2 Experimental results

We ran numerous test on the the four algorithms outlined in Section 5.1. The tests were done on all possible permutations of the following values: all three information loss metrics, three different sizes for the database (only part of the data was tested on 5000 member databases), six different distributions (three of which are Poisson 1,2.5,4), and 5 values of K. The values chosen for K were 2, 3, 5, 10, 18

Algorithm: K-anonymization by any set in the equivalence classes and breaking sets larger than 2k.

AnySetBreak(D,k)

Form equivalence classes EQ from $M \in D$
while $\exists\ eq_i \in EQ \mid eq_i < k$ **do**
  Randomly choose an $\{eq_i \in EQ\}$
  Find $d(eq_i, eq_j)$ for each $eq_j \in EQ$ where $i \neq j$
  $OrderEQ := \emptyset$
  for $eq_j \in EQ$ where $i \neq j$ **do**
    dis := $d(eq_i, eq_j)$
    $OrderEQ[END] := eq_j$
    $InLineSort(OrderEQ, dis, eq_j)$ # We are keep track of eq classes, smallest first
  end for
  $it := 0$
  $eq'_i := eq_i \cup OrderEQ[it] \cup \{$ Any sets sharing generalization $\}$
  while $\lceil \frac{eq'_i}{2} \rceil >= 2k$ **do**
    $it := it + 1$
    $eq'_i := eq_i \cup OrderEQ[it] \cup \{$ Any sets sharing generalization $\}$
  end while
  if $it = END$ then choose the smallest eq'_i
  while $eq'_i >= 2k$ **do**
    if $eq_i > OrderEQ[it]$ **then**
      Move best fitting members to $OrderEQ[it]$ from eq_i
    else if $eq_i < OrderEQ[it]$ **then**
      Move best fitting members to eq_i from $OrderEQ[it]$
    else
      Don't generalize
    end if
  end while
end while

Figure 5.5: This is the same as the algorithm in Section 5.1.3 except now any set is allowed to be picked as the pivot. We will not go into further details. The point of including this extra pseudo-code was to show the changes made.

and the database sizes were 50, 500, 5000. The following tables hold the results of the test, the discussion of the tables follows as well.

The common statistical tool used to measure how well a database is anonymized is the discernibility metric, which was discussed in the introduction. However, it is not applicable to what we wanted to test the databases for, i.e. the average anonymity and the average amount of generalization. We did not want to test the databases for equivalence class size only. We especially did not want a statistic based on member suppression. Thus, we have included different measures of quality for generalizations. For example, we used the following measures: largest equivalence class, smallest equivalence class, average equivalence class, average generalization, most generalized attribute, least generalized attribute, etc. Only some information measures will be included in this section. Extra information will be included in Appendix A.

The loader file used to create the databases for the test is essentially the loader file in File 4.1. The difference between loader files is the size of the database and distribution. The tree files will be included in Appendix A. Here, we will include four tables (we will not include table of Poisson $\lambda = 1$ and $\lambda = 4$), each table contains information for medium (500 members) sized databases . Each of the tables will include information pertaining to one of the distributions. The information which will be included in these charts is the following: metric, K, average set size, sets with size of k, members with at least one attribute not generalized, Members with no attributes generalized, least generalized member % , most generalized member % , and average member generalization %. We will include this list with associated numbers in each table. The tables are organized by distribution.

Table 5.1: Table of Results for the four algorithms when using a Gaussian distribution and medium sized database. The columns are labeled as follows: **0:** Algorithm, **1:** Metric, **2:** K value, **3:** Average set size, **4:** # of sets with size K, **5:** Members with at least one not generalized attribute, **6:** Members with no generalized attribute, **7:** Least generalized member by %, **8:** Most generalized member by %, **9:** Average Generalization by % .

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
SmallBreak	AnTree	2	2.59	100	309	37	0	100	49
SmallBreak	AnTree	3	4.46	33	153	3	0	100	67
SmallBreak	AnTree	5	8.06	8	93	0	42	100	76
SmallBreak	AnTree	10	18.52	3	24	0	58	100	82
SmallBreak	AnTree	18	35.71	2	19	0	75	100	88
SmallBreak	GCP	2	2.63	93	277	25	0	100	54
SmallBreak	GCP	3	4.76	21	131	1	0	100	70
SmallBreak	GCP	5	8.06	8	75	0	45	100	76
SmallBreak	GCP	10	19.23	1	28	0	74	100	83
SmallBreak	GCP	18	35.71	0	0	0	77	100	88
SmallBreak	Ent	2	2.45	118	200	9	0	45	22
SmallBreak	Ent	3	3.68	78	124	1	0	63	31
SmallBreak	Ent	5	7.04	18	10	0	21	62	45
SmallBreak	Ent	10	12.82	8	0	0	45	83	64
SmallBreak	Ent	18	20.83	0	0	0	58	89	74
SmallNoBreak	AnTree	2	4.42	66	196	29	0	100	65
SmallNoBreak	AnTree	3	14.29	16	46	7	0	100	85
SmallNoBreak	AnTree	5	62.50	0	6	0	49	100	95
SmallNoBreak	AnTree	10	166.67	0	0	0	74	100	98
SmallNoBreak	AnTree	18	500.00	0	0	0	100	100	100

Continued on next page

Table 5.1 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
SmallNoBreak	GCP	2	4.85	65	152	23	0	100	69
SmallNoBreak	GCP	3	19.23	10	44	7	0	100	88
SmallNoBreak	GCP	5	38.46	2	5	0	69	100	93
SmallNoBreak	GCP	10	500.00	0	0	0	100	100	100
SmallNoBreak	GCP	18	500.00	0	0	0	100	100	100
SmallNoBreak	Ent	2	2.73	91	256	9	0	46	23
SmallNoBreak	Ent	3	5.05	36	82	0	8	64	37
SmallNoBreak	Ent	5	10.20	3	0	0	24	72	52
SmallNoBreak	Ent	10	17.86	4	0	0	39	77	63
SmallNoBreak	Ent	18	45.45	0	0	0	60	87	78
AnyBreak	AnTree	2	2.65	75	300	52	0	100	45
AnyBreak	AnTree	3	4.55	12	203	15	0	100	58
AnyBreak	AnTree	5	9.80	1	88	1	0	100	74
AnyBreak	AnTree	10	16.13	1	54	0	42	100	79
AnyBreak	AnTree	18	33.33	0	62	1	0	100	74
AnyBreak	GCP	2	2.69	72	318	44	0	100	47
AnyBreak	GCP	3	4.63	10	197	12	0	100	61
AnyBreak	GCP	5	7.94	2	142	5	0	100	70
AnyBreak	GCP	10	18.52	0	64	1	0	100	77
AnyBreak	GCP	18	41.67	0	27	0	75	100	89
AnyBreak	Ent	2	2.65	70	161	13	0	45	25
AnyBreak	Ent	3	4.17	29	50	1	0	63	36
AnyBreak	Ent	5	7.46	1	6	0	23	79	52
AnyBreak	Ent	10	16.13	0	0	0	56	90	72
AnyBreak	Ent	18	26.32	1	0	0	71	94	79
AnyNoBreak	AnTree	2	500.00	0	0	0	100	100	100
AnyNoBreak	AnTree	3	500.00	0	0	0	100	100	100
AnyNoBreak	AnTree	5	500.00	0	0	0	100	100	100

Continued on next page

Table 5.1 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyNoBreak	AnTree	10	500.00	0	0	0	100	100	100
AnyNoBreak	AnTree	18	500.00	0	0	0	100	100	100
AnyNoBreak	GCP	2	500.00	0	0	0	100	100	100
AnyNoBreak	GCP	3	500.00	0	0	0	100	100	100
AnyNoBreak	GCP	5	500.00	0	0	0	100	100	100
AnyNoBreak	GCP	10	500.00	0	0	0	100	100	100
AnyNoBreak	GCP	18	500.00	0	0	0	100	100	100
AnyNoBreak	Ent	2	250.00	0	0	0	74	95	93
AnyNoBreak	Ent	5	100.00	0	0	0	59	95	90
AnyNoBreak	Ent	10	125.00	0	0	0	67	91	89
AnyNoBreak	Ent	18	500.00	0	0	0	95	95	95

Table 5.2: Table of Results for the four algorithms when using a log-Gaussian distribution and medium sized database. The columns are labeled as follows: **0:** Algorithm, **1:** Metric, **2:** K value, **3:** Average set size, **4:** # of sets with size K, **5:** Members with at least one not generalized attribute, **6:** Members with no generalized attribute, **7:** Least generalized member by %, **8:** Most generalized member by %, **9:** Average Generalization by % .

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
SmallBreak	AnTree	2	3.59	65	388	78	0	100	22
SmallBreak	AnTree	3	5.43	28	293	29	0	100	38
SmallBreak	AnTree	5	8.77	14	199	6	0	100	56
SmallBreak	AnTree	10	20	1	128	2	0	100	66
SmallBreak	AnTree	18	27.77	2	109	0	13	100	75

Continued on next page

Table 5.2 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
SmallBreak	GCP	2	3.70	54	386	79	0	100	21
SmallBreak	GCP	3	5.81	15	277	24	0	100	39
SmallBreak	GCP	5	8.92	11	169	9	0	100	56
SmallBreak	GCP	10	17.24	3	141	4	0	100	61
SmallBreak	GCP	18	35.71	0	0	0	65	100	84
SmallBreak	Ent	2	3.47	65	334	58	0	61	12
SmallBreak	Ent	3	4.76	45	249	32	0	67	19
SmallBreak	Ent	5	7.57	25	159	6	0	75	31
SmallBreak	Ent	10	16.12	3	104	3	0	92	49
SmallBreak	Ent	18	20	2	41	1	0	95	67
SmallNoBreak	AnTree	2	7.46	22	266	38	0	100	44
SmallNoBreak	AnTree	3	11.90	12	194	18	0	100	57
SmallNoBreak	AnTree	5	23.80	8	127	8	0	100	70
SmallNoBreak	AnTree	10	166.66	0	23	0	20	100	94
SmallNoBreak	AnTree	18	71.42	0	102	1	0	100	80
SmallNoBreak	GCP	2	7.042	19	276	35	0	100	43
SmallNoBreak	GCP	3	10.63	8	226	19	0	100	53
SmallNoBreak	GCP	5	21.73	6	164	8	0	100	65
SmallNoBreak	GCP	10	71.42	2	71	1	0	100	89
SmallNoBreak	GCP	18	500	0	0	0	100	100	100
SmallNoBreak	Ent	2	4.71	31	267	42	0	58	19
SmallNoBreak	Ent	3	6.57	13	195	16	0	63	26
SmallNoBreak	Ent	5	10	10	132	6	0	66	35
SmallNoBreak	Ent	10	21.73	2	74	2	0	88	53
SmallNoBreak	Ent	18	41.66	0	41	1	0	86	70
AnyBreak	AnTree	2	3.62	27	423	95	0	100	17
AnyBreak	AnTree	3	5.61	13	325	36	0	100	40
AnyBreak	AnTree	5	9.43	1	185	0	15	100	67

Continued on next page

Table 5.2 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyBreak	AnTree	10	16.66	1	30	0	28	100	77
AnyBreak	AnTree	18	31.25	0	61	2	0	100	73
AnyBreak	GCP	2	4.06	9	375	76	0	100	26
AnyBreak	GCP	3	6.49	1	291	30	0	100	40
AnyBreak	GCP	5	9.25	1	151	3	0	100	62
AnyBreak	GCP	10	21.73	0	56	0	28	100	76
AnyBreak	GCP	18	29.41	2	36	0	50	100	79
AnyBreak	Ent	2	3.62	47	326	52	0	49	13
AnyBreak	Ent	3	5.37	12	212	22	0	80	24
AnyBreak	Ent	5	8.77	0	101	4	0	75	37
AnyBreak	Ent	10	18.51	0	93	2	0	96	56
AnyBreak	Ent	18	18	0	41	1	0	98	75
AnyNoBreak	AnTree	2	250	0	0	0	83	100	99
AnyNoBreak	AnTree	3	500	0	0	0	100	100	100
AnyNoBreak	AnTree	5	500	0	0	0	100	100	100
AnyNoBreak	AnTree	10	500	0	0	0	100	100	100
AnyNoBreak	AnTree	18	500	0	0	0	100	100	100
AnyNoBreak	GCP	2	500	0	0	0	100	100	100
AnyNoBreak	GCP	3	166.66	1	0	0	50	100	99
AnyNoBreak	GCP	5	500	0	0	0	100	100	100
AnyNoBreak	GCP	10	500	0	0	0	100	100	100
AnyNoBreak	GCP	18	500	0	0	0	100	100	100
AnyNoBreak	Ent	2	27.77	0	39	0	22	90	70
AnyNoBreak	Ent	3	250	0	0	0	90	100	98
AnyNoBreak	Ent	5	100	0	0	0	56	100	94
AnyNoBreak	Ent	10	500	0	0	0	100	100	100
AnyNoBreak	Ent	18	500	0	0	0	100	100	100

Table 5.3: Table of Results for the four algorithms when using an uniform distribution and medium sized database. The columns are labeled as follows: **0:** Algorithm, **1:** Metric, **2:** K value, **3:** Average set size, **4:** # of sets with size K, **5:** Members with at least one not generalized attribute, **6:** Members with no generalized attribute, **7:** Least generalized member by %, **8:** Most generalized member by %, **9:** Average Generalization by % .

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
SmallBreak	AnTree	3	4.81	18	58	0	41	100	75
SmallBreak	AnTree	5	8.33	8	42	0	65	100	80
SmallBreak	AnTree	10	18.52	2	12	0	73	100	85
SmallBreak	AnTree	18	33.33	4	0	0	66	100	87
SmallBreak	AnTree	2	2.65	97	210	31	0	100	61
SmallBreak	GCP	2	2.75	80	185	26	0	100	63
SmallBreak	GCP	3	4.81	15	79	0	43	100	75
SmallBreak	GCP	5	8.33	7	38	0	65	100	79
SmallBreak	GCP	10	17.24	4	0	0	74	100	85
SmallBreak	GCP	18	31.25	2	0	0	77	100	86
SmallBreak	Ent	5	6.58	25	0	0	36	89	56
SmallBreak	Ent	10	13.16	8	0	0	54	100	79
SmallBreak	Ent	2	2.36	136	137	9	0	58	27
SmallBreak	Ent	3	3.88	58	16	0	21	69	41
SmallBreak	Ent	18	20.00	0	0	0	78	100	90
SmallNoBreak	AnTree	2	5.81	59	67	10	0	100	81
SmallNoBreak	AnTree	3	31.25	7	11	0	66	100	94
SmallNoBreak	AnTree	5	125.00	0	0	0	81	100	98
SmallNoBreak	AnTree	10	500.00	0	0	0	100	100	100
SmallNoBreak	AnTree	18	500.00	0	0	0	100	100	100

Continued on next page

Table 5.3 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
SmallNoBreak	GCP	2	5.62	67	92	10	0	100	81
SmallNoBreak	GCP	3	41.67	3	6	0	48	100	96
SmallNoBreak	GCP	5	100.00	0	0	0	80	100	98
SmallNoBreak	GCP	10	500.00	0	0	0	100	100	100
SmallNoBreak	GCP	18	500.00	0	0	0	100	100	100
SmallNoBreak	Ent	2	2.94	90	136	8	0	62	33
SmallNoBreak	Ent	3	5.32	25	6	0	27	80	49
SmallNoBreak	Ent	5	10.87	6	0	0	38	98	69
SmallNoBreak	Ent	10	20.83	0	0	0	69	98	87
SmallNoBreak	Ent	18	55.56	0	0	0	79	100	96
AnyBreak	AnTree	5	9.26	2	81	3	0	100	76
AnyBreak	AnTree	10	17.86	0	51	0	41	100	80
AnyBreak	AnTree	18	41.67	0	0	0	72	100	87
AnyBreak	AnTree	2	2.76	59	197	38	0	100	60
AnyBreak	AnTree	3	5.05	5	121	5	0	100	71
AnyBreak	GCP	10	20.00	1	10	0	29	100	82
AnyBreak	GCP	5	8.62	1	100	2	0	100	76
AnyBreak	GCP	18	31.25	0	0	0	51	100	85
AnyBreak	GCP	3	5.15	6	112	6	0	100	69
AnyBreak	GCP	2	2.78	54	184	32	0	100	61
AnyBreak	Ent	2	2.49	103	112	5	0	62	29
AnyBreak	Ent	10	20.83	0	0	0	81	100	93
AnyBreak	Ent	3	4.17	24	9	0	19	70	44
AnyBreak	Ent	18	62.50	0	0	0	89	100	98
AnyBreak	Ent	5	7.81	0	0	0	45	96	64
AnyNoBreak	AnTree	2	125.00	0	0	0	63	100	99
AnyNoBreak	AnTree	3	500.00	0	0	0	100	100	100
AnyNoBreak	AnTree	5	500.00	0	0	0	100	100	100

Continued on next page

Table 5.3 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyNoBreak	AnTree	10	500.00	0	0	0	100	100	100
AnyNoBreak	AnTree	18	500.00	0	0	0	100	100	100
AnyNoBreak	GCP	2	500.00	0	0	0	100	100	100
AnyNoBreak	GCP	3	500.00	0	0	0	100	100	100
AnyNoBreak	GCP	5	500.00	0	0	0	100	100	100
AnyNoBreak	GCP	10	500.00	0	0	0	100	100	100
AnyNoBreak	GCP	18	500.00	0	0	0	100	100	100
AnyNoBreak	Ent	2	23.81	0	0	0	32	100	88
AnyNoBreak	Ent	3	250.00	0	0	0	93	100	99
AnyNoBreak	Ent	5	500.00	0	0	0	100	100	100
AnyNoBreak	Ent	10	166.67	0	0	0	90	100	99
AnyNoBreak	Ent	18	500.00	0	0	0	100	100	100

Table 5.4: Table of results for the four algorithms when using a Poisson distribution with $\lambda = 2.5$ and medium size database. The columns are labeled as follows: **0:** Algorithm, **1:** Metric, **2:** K value, **3:** Average set size, **4:** # of sets with size K, **5:** Members with at least one not generalized attribute, **6:** Members with no generalized attribute, **7:** Least generalized member by %, **8:** Most generalized member by %, **9:** Average Generalization by % .

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
SmallBreak	AnTree	2	2.63	96	336	109	0	91	28
SmallBreak	AnTree	18	33.33	4	0	0	56	91	70
SmallBreak	AnTree	5	8.06	10	113	0	22	91	58
SmallBreak	AnTree	10	17.86	4	10	0	37	91	64

Continued on next page

Table 5.4 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
SmallBreak	AnTree	3	4.35	36	213	25	0	91	45
SmallBreak	GCP	2	2.56	111	338	116	0	91	25
SmallBreak	GCP	10	16.67	1	0	0	42	91	64
SmallBreak	GCP	3	4.50	27	211	25	0	91	46
SmallBreak	GCP	5	7.58	10	116	0	28	91	58
SmallBreak	GCP	18	41.67	0	0	0	52	91	70
SmallBreak	Ent	10	12.50	9	0	0	35	70	55
SmallBreak	Ent	18	26.32	1	0	0	37	69	62
SmallBreak	Ent	3	3.70	74	160	21	0	62	28
SmallBreak	Ent	5	6.58	21	11	0	21	67	42
SmallBreak	Ent	2	2.50	113	277	83	0	58	16
SmallNoBreak	AnTree	18	166.67	0	0	0	70	100	93
SmallNoBreak	AnTree	10	166.67	0	0	0	77	91	87
SmallNoBreak	AnTree	3	13.51	13	94	18	0	91	68
SmallNoBreak	AnTree	2	4.55	83	229	82	0	91	47
SmallNoBreak	AnTree	5	55.56	2	10	2	0	91	84
SmallNoBreak	GCP	10	125.00	1	0	0	58	91	89
SmallNoBreak	GCP	5	50.00	4	21	3	0	91	79
SmallNoBreak	GCP	3	13.51	11	90	17	0	88	68
SmallNoBreak	GCP	2	4.85	78	223	82	0	91	48
SmallNoBreak	GCP	18	166.67	0	0	0	72	100	87
SmallNoBreak	Ent	10	21.74	0	0	0	44	68	59
SmallNoBreak	Ent	3	6.33	19	90	10	0	69	37
SmallNoBreak	Ent	18	38.46	2	0	0	47	73	66
SmallNoBreak	Ent	5	11.11	11	34	0	14	69	48
SmallNoBreak	Ent	2	3.62	65	222	68	0	67	25
AnyBreak	AnTree	2	2.66	87	361	121	0	91	23
AnyBreak	AnTree	3	4.81	10	203	17	0	91	47

Continued on next page

Table 5.4 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyBreak	AnTree	5	8.33	1	117	0	14	91	59
AnyBreak	AnTree	10	16.67	2	63	1	0	91	62
AnyBreak	AnTree	18	29.41	2	24	0	45	91	68
AnyBreak	GCP	2	2.73	67	366	105	0	91	26
AnyBreak	GCP	3	4.59	5	208	9	0	91	49
AnyBreak	GCP	5	8.33	1	80	0	37	91	60
AnyBreak	GCP	18	38.46	0	0	0	56	91	71
AnyBreak	GCP	10	15.63	0	68	0	44	100	63
AnyBreak	Ent	3	4.24	23	95	10	0	65	33
AnyBreak	Ent	18	31.25	0	0	0	62	77	69
AnyBreak	Ent	10	16.67	0	0	0	54	75	62
AnyBreak	Ent	2	2.49	116	278	73	0	50	17
AnyBreak	Ent	5	7.58	1	0	0	30	65	50
AnyNoBreak	AnTree	5	500.00	0	0	0	100	100	100
AnyNoBreak	AnTree	18	500.00	0	0	0	100	100	100
AnyNoBreak	AnTree	3	500.00	0	0	0	100	100	100
AnyNoBreak	AnTree	2	500.00	0	0	0	100	100	100
AnyNoBreak	AnTree	10	500.00	0	0	0	100	100	100
AnyNoBreak	GCP	5	500.00	0	0	0	100	100	100
AnyNoBreak	GCP	2	500.00	0	0	0	100	100	100
AnyNoBreak	GCP	3	500.00	0	0	0	100	100	100
AnyNoBreak	GCP	10	500.00	0	0	0	100	100	100
AnyNoBreak	GCP	18	500.00	0	0	0	100	100	100
AnyNoBreak	Ent	2	25.00	0	36	0	32	71	60
AnyNoBreak	Ent	3	50.00	0	0	0	43	78	71
AnyNoBreak	Ent	18	250.00	0	0	0	74	80	78
AnyNoBreak	Ent	10	250.00	0	0	0	57	80	78
AnyNoBreak	Ent	5	250.00	0	0	0	44	80	78

6. DISCUSSION AND CONCLUSION

The data collected highlights the importance of member suppression. If you look through the tables in the experimental test section you will see that there are many completely suppressed databases. This occurs especially when the distribution has a long tail, for instance log-Gaussian. The effect of having a couple of extreme outliers will not perturb the resulting database's information loss when the value of k is small.Only an insignificant number of equivalence classes will be affected by the outliers. However, when you have a large value for k, the equivalence classes with the outlier will join with other classes, thereby increasing, the amount of generalization necessary to join equivalence classes. This increase can potentially cause massive levels of suppression. This is why the algorithms which do not break sets lager than 2K suffer the most, especially, when any set can be chosen as the random pivot.

What we haven't tested, but seems to affect the outcome of the experiments is the number of times we run the algorithm. Being that all the algorithms tested are randomized algorithms, the order in which equivalence classes are selected affects the outcome (same is true for greedy algorithms). Thus, if we were to run the algorithms N times in a row and select the best results the outcome might be much better. Nonetheless, some results are apparent even without sequential runs, for example, the importance of breaking sets lager than 2K. There is a strict divide in the results for algorithms that break large sets and those that do not. Most of the complete generalizations occur when the algorithm does not break sets larger than 2K. This is cause by the lack of sequential runs. If the algorithm makes a poor choice when it comes to the pivot, in algorithms that do not break sets larger than 2K the choice cannot be undone, while in algorithms that break sets larger than 2K there is a possibility for the algorithm to fix the mistake in the future. It should still be possible for algorithms which do not break sets larger than 2K to reach the same resulting database as algorithms which do break sets larger than 2K but, the number of sequential trials that need to occur might be astronomical. The three important statistics that where affected when using algorithms which do not break

sets larger than 2K were average generalization, number of suppressed members, and number of equivalence classes with size near k. Entropy did the best when used on algorithms which do not break sets larger than 2K, however, in some cases the improvement was marginal. The result for Entropy might be due to the lack of constraints on the allowed generalizations.

The difference between algorithms choosing any set as the pivot and only choosing the smallest sets was marginal, but it is obvious that choosing any set did worst in many cases. However, interestingly enough, the best result for "any set and do not break large sets" was using Entropy on a Poisson distribution with $\lambda = 2.5$. As for the GCP and AnTree metrics, they do better when the distribution is skewed, however, if it has a long tail, they end up doing poorly on large values of k. These metrics do much better when the PID values in the member set use only a small compact subspace of the attribute space rather then the whole space (the uniform distribution selects values uniformly across the attribute space). However, this is not strictly a problem for GCP and AnTree metrics. Entropy also does worst on the uniform distribution.

The best performing metric was Entropy, and the best performing algorithm was "only smallest sets, and break large sets" (this was the case for every distribution and every metric), and the best distribution was Gaussian/Poisson. The statistics were the most favorable for when using Entropy,"only smallest sets, and break large sets", and Gaussian/Poisson distribution. Some of the results for Entropy might be skewed because it was not implemented using generalization trees. However, this is one of the upsides of Entropy compared to the other two methods, which rely more on generalization trees. The GCP can be implemented without Generalization trees, but AnTree cannot be. One benefit that Entropy does not share with the other metrics is that the attributes can be weighted in AnTree and GCP. However, if Entropy metric is designed with a conditional probability distributions it might be possible to include attribute and attribute value weightings. The improved performance of the best performing algorithm is due to the fact that it limits the search space the most. While some of the other algorithms are not far from having similar results, there is a possibility the best algorithm is working on

a local minimum in the search space. The two best distributions are skewed, but do not have as long of tails as log-Gaussian. Thus, it is understandable why they would give the best results. However, we do not know how well these distributions reflect real life.

6.1 Future work

Three algorithms so far have attained an $O(k)$-approximation of the optimal solution for cell suppression. Moreover, some algorithms give an $O(k)$-approximation and do more general K-anonymization than cell suppression. It seems that K-anonymity might be inapproximiable passed $O(k)$ the optimal solution. This seems highly likely when taking into account the already shown inapproximability result. Thus, one possible direction would be finding a inapproximability bound on K-anonymity with local re-encoding. Another possible direction for future work is, running the statistical analysis on a more general array of algorithms. The set that we have looked at here is myopic in both algorithms, information loss metric, and constraints. The reason for this is that we wanted the volume of data to be feasible to handle. Given more time, more comparisons could be done, that lead to better ways of analyzing databases. Also, a more thorough test of Entropy. This test should include an implementation of conditional Entropy, as well as, testing Entropy with a generalization tree structure. This would be helpful for comparison to generalization tree dependent metrics. Finally, testing the algorithms, metrics and constraints on real dataset will definitely improve the applicability of the results to real world applications. So to conclude, there is still much to be done in this area.

LITERATURE CITED

[1] Jiuyong Li, Raymond Chi wing Wong, Ada Wai chee Fu, and Jian Pei. Achieving k-anonymity by clustering in attribute hierarchical structures. In *in Proceesing of 8th International Conference on Data Warehousing and Knowledge Discovery*, pages 405–416, 2006.

[2] Jian Xu, Wei Wang, Jian Pei, Xiaoyuan Wang, Baile Shi, and Ada Wai chee Fu. Utility-based anonymization using local recoding. In *In SIGKDD*, pages 785–790, 2006.

[3] Adam Meyerson and Ryan Williams. On the complexity of optimal k-anonymity. In *PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 223–228, New York, NY, USA, 2004. ACM.

[4] George T. Duncan and Sumitra Mukherjee. Optimal disclosure limitation strategy in statistical databases: Deterring tracker attacks through additive noise. *Journal of the American Statistical Association*, 95(451):720–729, 2000.

[5] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data*, 1(1):3, 2007.

[6] Ninghui Li and Tiancheng Li. t-closeness: Privacy beyond k-anonymity and -diversity. In *In Proc. of IEEE 23rd Intl Conf. on Data Engineering (ICDE07*, 2007.

[7] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, and P. Samarati. k-Anonymity. In T. Yu and S. Jajodia, editors, *Secure Data Management in Decentralized Systems*. Springer-Verlag, 2007.

[8] V. Ciriani, S. De, Capitani Vimercati, S. Foresti, and P. Samarati. *Privacy preserving data mining: models and algorithms*, chapter Chapter 5 K-ANONYMOUS DATA MINING: A SURVEY. Springer, 2008.

[9] Roberto J. Bayardo. Data privacy through optimal k-anonymization. In *In ICDE*, pages 217–228, 2005.

[10] Gabriel Ghinita, Panagiotis Karras, Panos Kalnis, and Nikos Mamoulis. Fast data anonymization with low information loss. In *in VLDB, 2007*, pages 758–769, 2007.

[11] John Miller, Alina Campan, and Traian Marius Truta. Constrained k-anonymity: Privacy with generalization boundaries.

[12] Kristen Lefevre, David J. Dewitt, and Raghu Ramakrishnan. Mondrian multidimensional k-anonymity. In *In ICDE*, 2006.

[13] Gagan Aggarwal, Tomas Feder, Krishnaram Kenthapadi, Rajeev Motwani, Rina Panigrahy, Dilys Thomas, and An Zhu. Anonymizing tables for privacy protection, 2005.

[14] Rhonda Chaytor. A better problem representation for k-anonymity. In *In Proc. 1st ACM SIGKDD Intl Work. on Privacy, Security, and Trust in KDD*, pages 52–61, 2007.

[15] Pierangela Samara Ti. Protecting respondents' identities in microdata release. *IEEE Transactions on Knowledge and Data Engineering*, 13:1010–1027, 2001.

[16] Kristen LeFevre, David J. DeWitt, and Raghu Ramakrishnan. Incognito: efficient full-domain k-anonymity. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 49–60, New York, NY, USA, 2005. ACM.

[17] Kristen LeFevre, David J. DeWitt, and Raghu Ramakrishnan. Mondrian multidimensional k-anonymity. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, page 25, Washington, DC, USA, 2006. IEEE Computer Society.

[18] Vijay S. Iyengar. Transforming data to satisfy privacy constraints, 2002.

[19] Gagan Aggarwal, Tomas Feder, Krishnaram Kenthapadi, Rajeev Motwani, Rina Panigrahy, Dilys Thomas, and An Zhu. Approximation algorithms for k-anonymity, 2005.

[20] Sven Krumke, H. Noltemeier, S. S. Ravi, and Madhav V. Marathe. Bicriteria compact location problems, 1996.

[21] S. O. Krumke, H. Noltemeier, S. S. Ravi, and M. V. Marathe. Compact location problems with budget and communication constraints, 1995.

APPENDIX A
Generalization tree files, extra charts on algorithms performance

A.1 Generalization tree files

These are the generalization tree files that where used for the experimental result. **This is for the first attribute of the pseudo-identifier: B.**

```
{b0|b1|b2|b3|b4|b5}
        {b0|b1|b2}
        {b3|b4|b5}
{b0|b1|b2}
        {b0|b1}
        b2
{b3|b4|b5}
        b3
        {b4|b5}
{b0|b1}
        b0
        b1
{b4|b5}
        b4
        b5
```

This is for the second attribute of the pseudo-identifier: C.

```
{c0|c1|c2|c3|c4|c5|c6|c7|c8|c9}
        {c0|c1|c2|c3}
        {c4|c5|c6|c7|c8|c9}
{c0|c1|c2|c3}
        {c0|c1|c2}
        c3
{c4|c5|c6|c7|c8|c9}
        {c4|c5|c6}
        {c7|c8|c9}
{c0|c1|c2}
        c2
        {c0|c1}
{c4|c5|c6}
        {c4|c5}
        c6
{c7|c8|c9}
        {c7|c8}
```

```
18          c9
19 {c0 | c1}
20          c0
21          c1
22
23 {c4 | c5}
24          c4
25          c5
26 {c7 | c8}
27          c7
28          c8
```

This is for the third attribute of the pseudo-identifier: D.

```
1  {d0 | d1 | d2 | d3 | d4 | d5 | d6 | d7 | d8 | d9 | d10 | d11}
2           {d0 | d1 | d2 | d3 | d4 | d5 | d6 | d7}
3           {d8 | d9 | d10 | d11}
4  {d0 | d1 | d2 | d3 | d4 | d5 | d6 | d7}
5           {d0 | d1 | d2 | d3}
6           {d4 | d5 | d6 | d7}
7  {d0 | d1 | d2 | d3}
8           {d0 | d1}
9           {d2 | d3}
10 {d4 | d5 | d6 | d7}
11          {d4 | d5}
12          {d6 | d7}
13 {d8 | d9 | d10 | d11}
14          {d8 | d9}
15          {d10 | d11}
16 {d0 | d1}
17          d0
18          d1
19 {d2 | d3}
20          d2
21          d3
22 {d4 | d5}
23          d4
24          d5
25 {d6 | d7}
26          d6
27          d7
28 {d8 | d9}
29          d8
30          d9
31 {d10 | d11}
32          d10
33          d11
```

This is for the fourth attribute of the pseudo-identifier: E.

```
{e0|e1|e2|e3|e4|e5|e6|e7|e8|e9|e10|e11|e12|e13|e14|e15}
        {e0|e1|e2|e3|e4|e5|e6|e7}
        {e8|e9|e10|e11|e12|e13|e14|e15}
{e0|e1|e2|e3|e4|e5|e6|e7}
        {e0|e1|e2|e3}
        {e4|e5|e6|e7}
{e8|e9|e10|e11|e12|e13|e14|e15}
        {e8|e9|e10|e11}
        {e12|e13|e14|e15}
{e0|e1|e2|e3}
        {e0|e1}
        {e2|e3}
{e4|e5|e6|e7}
        {e4|e5}
        {e6|e7}
{e8|e9|e10|e11}
        {e8|e9}
        {e10|e11}
{e12|e13|e14|e15}
        {e12|e13}
        {e14|e15}
{e0|e1}
        e0
        e1
{e2|e3}
        e2
        e3
{e4|e5}
        e4
        e5
{e6|e7}
        e6
        e7
{e8|e9}
        e8
        e9
{e10|e11}
        e10
        e11
{e12|e13}
        e12
        e13
{e14|e15}
        e14
        e15
```

A.2 Extra Tables for Medium Sized Databases

Table A.1: Table of Results for the four algorithms when using a Poisson distribution with $\lambda = 1.0$ and medium database size. The columns are labeled as follows: **0:** Algorithm, **1:** Metric, **2:** K value, **3:** Average set size, **4:** # of sets with size K, **5:** Members with at least one not generalized attribute, **6:** Members with no generalized attribute, **7:** Least generalized member by %, **8:** Most generalized member by %, **9:** Average Generalization by % .

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyBreak	AnTree	2	4.85	0	482	90	0	56	3
AnyBreak	AnTree	18	27.78	0	92	1	0	77	50
AnyBreak	AnTree	10	15.15	0	168	0	5	77	45
AnyBreak	AnTree	3	7.04	1	409	45	0	70	13
AnyBreak	AnTree	5	9.43	0	312	16	0	77	27
AnyBreak	GCP	5	8.93	4	299	14	0	72	30
AnyBreak	GCP	10	16.13	2	150	1	0	77	46
AnyBreak	GCP	2	5.43	2	474	82	0	70	3
AnyBreak	GCP	18	29.41	0	0	0	39	77	57
AnyBreak	GCP	3	7.35	1	427	44	0	77	11
AnyBreak	Ent	5	8.93	1	41	0	9	52	33
AnyBreak	Ent	2	3.82	35	413	81	0	44	7
AnyBreak	Ent	18	31.25	0	0	0	39	54	48
AnyBreak	Ent	10	18.52	0	19	0	18	54	41
AnyBreak	Ent	3	5.10	20	338	46	0	52	13
AnyNoBreak	AnTree	3	250.00	0	4	1	0	77	76
AnyNoBreak	AnTree	5	500.00	0	0	0	77	77	77
AnyNoBreak	AnTree	18	500.00	0	0	0	77	77	77
AnyNoBreak	AnTree	10	500.00	0	0	0	77	77	77

Continued on next page

Table A.1 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyNoBreak	AnTree	2	500.00	0	0	0	77	77	77
AnyNoBreak	GCP	18	500.00	0	0	0	77	77	77
AnyNoBreak	GCP	10	500.00	0	0	0	77	77	77
AnyNoBreak	GCP	2	500.00	0	0	0	77	77	77
AnyNoBreak	GCP	3	166.67	0	15	0	3	77	75
AnyNoBreak	GCP	5	500.00	0	0	0	77	77	77
AnyNoBreak	Ent	3	50.00	0	8	0	27	56	48
AnyNoBreak	Ent	18	250.00	0	0	0	48	56	55
AnyNoBreak	Ent	2	35.71	0	62	0	20	54	43
AnyNoBreak	Ent	5	35.71	0	156	0	33	53	44
AnyNoBreak	Ent	10	100.00	0	0	0	36	54	51
SmallBreak	AnTree	2	4.03	46	462	103	0	77	5
SmallBreak	AnTree	3	5.26	29	424	63	0	77	11
SmallBreak	AnTree	5	8.20	13	326	17	0	77	27
SmallBreak	AnTree	10	14.71	0	145	0	8	77	43
SmallBreak	AnTree	18	27.78	1	82	1	0	77	45
SmallBreak	GCP	2	4.20	39	466	100	0	70	4
SmallBreak	GCP	3	5.15	34	420	69	0	77	11
SmallBreak	GCP	5	7.58	21	280	24	0	77	25
SmallBreak	GCP	10	13.89	5	160	2	0	77	41
SmallBreak	GCP	18	29.41	2	64	1	0	77	47
SmallBreak	Ent	2	3.85	44	445	91	0	48	5
SmallBreak	Ent	3	4.90	35	364	54	0	44	11
SmallBreak	Ent	5	7.69	15	182	14	0	51	23
SmallBreak	Ent	10	12.82	7	55	0	14	52	33
SmallBreak	Ent	18	22.73	5	18	0	18	56	43
SmallNoBreak	AnTree	2	5.15	28	415	93	0	77	13
SmallNoBreak	AnTree	3	6.85	15	362	49	0	77	21

Continued on next page

Table A.1 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
SmallNoBreak	AnTree	5	12.82	11	227	20	0	77	39
SmallNoBreak	AnTree	10	55.56	0	64	3	0	77	64
SmallNoBreak	AnTree	18	166.67	0	0	0	46	77	71
SmallNoBreak	GCP	2	5.26	26	415	93	0	77	13
SmallNoBreak	GCP	3	7.35	15	362	48	0	77	22
SmallNoBreak	GCP	5	11.90	12	257	21	0	77	36
SmallNoBreak	GCP	10	55.56	0	51	4	0	77	63
SmallNoBreak	GCP	18	166.67	0	0	0	35	77	72
SmallNoBreak	Ent	2	4.55	29	406	83	0	45	7
SmallNoBreak	Ent	3	6.02	13	357	35	0	49	14
SmallNoBreak	Ent	5	9.43	13	189	7	0	49	24
SmallNoBreak	Ent	10	27.78	0	84	3	0	56	39
SmallNoBreak	Ent	18	45.45	1	0	0	35	54	47

Table A.2: Table of Results for the four algorithms when using a Poisson distribution with $\lambda = 4.0$ and medium sized database. The columns are labeled as follows: **0:** Algorithm, **1:** Metric, **2:** K value, **3:** Average set size, **4:** # of sets with size K, **5:** Members with at least one not generalized attribute, **6:** Members with no generalized attribute, **7:** Least generalized member by %, **8:** Most generalized member by %, **9:** Average Generalization by % .

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyBreak	AnTree	2	4.85	0	482	90	0	56	3
AnyBreak	AnTree	18	27.78	0	92	1	0	77	50
AnyBreak	AnTree	10	15.15	0	168	0	5	77	45

Continued on next page

Table A.2 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyBreak	AnTree	3	7.04	1	409	45	0	70	13
AnyBreak	AnTree	5	9.43	0	312	16	0	77	27
AnyBreak	GCP	5	8.93	4	299	14	0	72	30
AnyBreak	GCP	10	16.13	2	150	1	0	77	46
AnyBreak	GCP	2	5.43	2	474	82	0	70	3
AnyBreak	GCP	18	29.41	0	0	0	39	77	57
AnyBreak	GCP	3	7.35	1	427	44	0	77	11
AnyBreak	Ent	5	8.93	1	41	0	9	52	33
AnyBreak	Ent	2	3.82	35	413	81	0	44	7
AnyBreak	Ent	18	31.25	0	0	0	39	54	48
AnyBreak	Ent	10	18.52	0	19	0	18	54	41
AnyBreak	Ent	3	5.10	20	338	46	0	52	13
AnyNoBreak	AnTree	3	250.00	0	4	1	0	77	76
AnyNoBreak	AnTree	5	500.00	0	0	0	77	77	77
AnyNoBreak	AnTree	18	500.00	0	0	0	77	77	77
AnyNoBreak	AnTree	10	500.00	0	0	0	77	77	77
AnyNoBreak	AnTree	2	500.00	0	0	0	77	77	77
AnyNoBreak	GCP	18	500.00	0	0	0	77	77	77
AnyNoBreak	GCP	10	500.00	0	0	0	77	77	77
AnyNoBreak	GCP	2	500.00	0	0	0	77	77	77
AnyNoBreak	GCP	3	166.67	0	15	0	3	77	75
AnyNoBreak	GCP	5	500.00	0	0	0	77	77	77
AnyNoBreak	Ent	3	50.00	0	8	0	27	56	48
AnyNoBreak	Ent	18	250.00	0	0	0	48	56	55
AnyNoBreak	Ent	2	35.71	0	62	0	20	54	43
AnyNoBreak	Ent	5	35.71	0	156	0	33	53	44
AnyNoBreak	Ent	10	100.00	0	0	0	36	54	51
SmallBreak	AnTree	2	4.03	46	462	103	0	77	5

Continued on next page

Table A.2 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
SmallBreak	AnTree	3	5.26	29	424	63	0	77	11
SmallBreak	AnTree	5	8.20	13	326	17	0	77	27
SmallBreak	AnTree	10	14.71	0	145	0	8	77	43
SmallBreak	AnTree	18	27.78	1	82	1	0	77	45
SmallBreak	GCP	2	4.20	39	466	100	0	70	4
SmallBreak	GCP	3	5.15	34	420	69	0	77	11
SmallBreak	GCP	5	7.58	21	280	24	0	77	25
SmallBreak	GCP	10	13.89	5	160	2	0	77	41
SmallBreak	GCP	18	29.41	2	64	1	0	77	47
SmallBreak	Ent	2	3.85	44	445	91	0	48	5
SmallBreak	Ent	3	4.90	35	364	54	0	44	11
SmallBreak	Ent	5	7.69	15	182	14	0	51	23
SmallBreak	Ent	10	12.82	7	55	0	14	52	33
SmallBreak	Ent	18	22.73	5	18	0	18	56	43
SmallNoBreak	AnTree	2	5.15	28	415	93	0	77	13
SmallNoBreak	AnTree	3	6.85	15	362	49	0	77	21
SmallNoBreak	AnTree	5	12.82	11	227	20	0	77	39
SmallNoBreak	AnTree	10	55.56	0	64	3	0	77	64
SmallNoBreak	AnTree	18	166.67	0	0	0	46	77	71
SmallNoBreak	GCP	2	5.26	26	415	93	0	77	13
SmallNoBreak	GCP	3	7.35	15	362	48	0	77	22
SmallNoBreak	GCP	5	11.90	12	257	21	0	77	36
SmallNoBreak	GCP	10	55.56	0	51	4	0	77	63
SmallNoBreak	GCP	18	166.67	0	0	0	35	77	72
SmallNoBreak	Ent	2	4.55	29	406	83	0	45	7
SmallNoBreak	Ent	3	6.02	13	357	35	0	49	14
SmallNoBreak	Ent	5	9.43	13	189	7	0	49	24
SmallNoBreak	Ent	10	27.78	0	84	3	0	56	39

Continued on next page

Table A.2 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
SmallNoBreak	Ent	18	45.45	1	0	0	35	54	47

A.3 Extra Tables for Small Sized Databases

Table A.3: Table of Results for the four algorithms when using a Gaussian distribution and small sized database. The columns are labeled as follows: **0:** Algorithm, **1:** Metric, **2:** K value, **3:** Average set size, **4:** # of sets with size K, **5:** Members with at least one not generalized attribute, **6:** Members with no generalized attribute, **7:** Least generalized member by %, **8:** Most generalized member by %, **9:** Average Generalization by % .

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyBreak	AnTree	2	2.63	8	17	2	0	100	70
AnyBreak	AnTree	3	4.17	1	5	0	55	100	81
AnyBreak	AnTree	5	7.14	1	9	1	0	100	71
AnyBreak	AnTree	10	16.67	0	12	0	50	100	81
AnyBreak	AnTree	18	25.00	0	0	0	80	100	90
AnyBreak	GCP	2	2.63	10	24	7	0	100	53
AnyBreak	GCP	3	5.00	0	9	0	33	100	81
AnyBreak	GCP	5	8.33	0	0	0	80	100	86
AnyBreak	GCP	10	12.50	0	0	0	83	100	91
AnyBreak	GCP	18	25.00	0	25	0	75	100	87
AnyBreak	Ent	2	2.50	10	8	0	16	47	28
AnyBreak	Ent	3	4.17	3	0	0	28	49	37
AnyBreak	Ent	5	6.25	1	0	0	41	59	51

Continued on next page

Table A.3 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyBreak	Ent	10	16.67	0	0	0	59	77	70
AnyBreak	Ent	18	25.00	0	0	0	73	83	79
AnyNoBreak	AnTree	2	50.00	0	0	0	100	100	100
AnyNoBreak	AnTree	3	50.00	0	0	0	100	100	100
AnyNoBreak	AnTree	5	50.00	0	0	0	100	100	100
AnyNoBreak	AnTree	10	50.00	0	0	0	100	100	100
AnyNoBreak	AnTree	18	50.00	0	0	0	100	100	100
AnyNoBreak	GCP	2	50.00	0	0	0	100	100	100
AnyNoBreak	GCP	3	50.00	0	0	0	100	100	100
AnyNoBreak	GCP	5	50.00	0	0	0	100	100	100
AnyNoBreak	GCP	10	50.00	0	0	0	100	100	100
AnyNoBreak	GCP	18	50.00	0	0	0	100	100	100
AnyNoBreak	Ent	2	3.13	6	21	0	8	68	35
AnyNoBreak	Ent	3	50.00	0	0	0	83	83	83
AnyNoBreak	Ent	5	10.00	1	0	0	56	83	69
AnyNoBreak	Ent	10	16.67	0	0	0	68	81	75
AnyNoBreak	Ent	18	50.00	0	0	0	83	83	83
SmallBreak	AnTree	2	2.78	6	11	3	0	100	64
SmallBreak	AnTree	3	5.00	3	3	0	75	100	87
SmallBreak	AnTree	5	10.00	0	0	0	83	100	92
SmallBreak	AnTree	10	16.67	2	0	0	80	100	92
SmallBreak	AnTree	18	25.00	0	0	0	91	100	95
SmallBreak	GCP	2	2.94	7	13	1	0	100	76
SmallBreak	GCP	3	4.55	3	3	1	0	100	81
SmallBreak	GCP	5	12.50	0	0	0	88	100	93
SmallBreak	GCP	10	12.50	1	10	0	75	100	90
SmallBreak	GCP	18	25.00	0	0	0	91	100	95
SmallBreak	Ent	2	2.27	16	13	0	11	44	26

Continued on next page

Table A.3 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
SmallBreak	Ent	3	3.85	6	0	0	24	60	41
SmallBreak	Ent	5	6.25	2	0	0	35	57	45
SmallBreak	Ent	10	8.33	0	0	0	60	70	63
SmallBreak	Ent	18	25.00	0	0	0	74	81	77
SmallNoBreak	AnTree	2	3.33	9	13	0	47	100	79
SmallNoBreak	AnTree	3	50.00	0	0	0	100	100	100
SmallNoBreak	AnTree	3	12.50	2	0	0	68	100	93
SmallNoBreak	AnTree	5	50.00	0	0	0	100	100	100
SmallNoBreak	AnTree	10	50.00	0	0	0	100	100	100
SmallNoBreak	AnTree	18	50.00	0	0	0	100	100	100
SmallNoBreak	GCP	2	3.85	7	8	0	38	100	78
SmallNoBreak	GCP	3	12.50	2	3	0	45	100	93
SmallNoBreak	GCP	5	50.00	0	0	0	100	100	100
SmallNoBreak	GCP	10	50.00	0	0	0	100	100	100
SmallNoBreak	GCP	18	50.00	0	0	0	100	100	100
SmallNoBreak	Ent	2	3.57	2	2	0	8	45	34
SmallNoBreak	Ent	3	4.55	3	3	0	26	59	40
SmallNoBreak	Ent	5	16.67	0	0	0	51	76	65
SmallNoBreak	Ent	10	50.00	0	0	0	83	83	83
SmallNoBreak	Ent	18	50.00	0	0	0	83	83	83

Table A.4: Table of Results for the four algorithms when using a log-Gaussian distribution and small sized database. The columns are labeled as follows: **0:** Algorithm, **1:** Metric, **2:** K value, **3:** Average set size, **4:** # of sets with size K, **5:** Members with at least one not generalized attribute, **6:** Members with no generalized attribute, **7:** Least generalized member by %, **8:** Most generalized member by %, **9:** Average Generalization by % .

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyBreak	AnTree	2	2.78	5	29	4	0	100	55
AnyBreak	AnTree	3	4.17	3	10	1	0	100	68
AnyBreak	AnTree	5	7.14	0	9	0	50	100	77
AnyBreak	AnTree	10	25.00	0	0	0	88	100	94
AnyBreak	AnTree	18	25.00	0	0	0	89	100	94
AnyNoBreak	AnTree	2	12.50	3	4	0	15	100	93
AnyNoBreak	AnTree	3	50.00	0	0	0	100	100	100
AnyNoBreak	AnTree	5	50.00	0	0	0	100	100	100
AnyNoBreak	AnTree	10	50.00	0	0	0	100	100	100
AnyNoBreak	AnTree	18	50.00	0	0	0	100	100	100
SmallBreak	AnTree	2	2.94	3	23	2	0	100	55
SmallBreak	AnTree	3	4.55	3	11	1	0	100	71
SmallBreak	AnTree	5	12.50	0	15	0	50	100	87
SmallBreak	AnTree	10	50.00	0	0	0	100	100	100
SmallBreak	AnTree	18	50.00	0	0	0	100	100	100
SmallNoBreak	AnTree	2	7.14	2	9	1	0	100	81
SmallNoBreak	AnTree	3	25.00	1	0	0	72	100	98
SmallNoBreak	AnTree	5	50.00	0	0	0	100	100	100
SmallNoBreak	AnTree	10	50.00	0	0	0	100	100	100
SmallNoBreak	AnTree	18	50.00	0	0	0	100	100	100

Continued on next page

Table A.4 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyBreak	Ent	2	2.78	9	31	1	0	43	20
AnyBreak	Ent	3	4.55	1	13	1	0	46	32
AnyBreak	Ent	5	8.33	0	0	0	34	56	48
AnyBreak	Ent	10	12.50	0	0	0	66	79	73
AnyBreak	Ent	18	25.00	0	0	0	68	76	72
AnyNoBreak	Ent	2	16.67	0	0	0	35	76	67
AnyNoBreak	Ent	3	50.00	0	0	0	82	82	82
AnyNoBreak	Ent	5	16.67	0	0	0	39	64	59
AnyNoBreak	Ent	10	50.00	0	0	0	82	82	82
AnyNoBreak	Ent	18	50.00	0	0	0	82	82	82
SmallBreak	Ent	2	2.63	11	32	1	0	37	19
SmallBreak	Ent	3	3.57	9	8	1	0	44	28
SmallBreak	Ent	5	6.25	4	10	1	0	53	37
SmallBreak	Ent	10	12.50	0	0	0	53	66	63
SmallBreak	Ent	18	16.67	0	0	0	69	82	76
SmallNoBreak	Ent	2	3.33	3	20	1	0	44	24
SmallNoBreak	Ent	3	6.25	1	7	0	21	48	38
SmallNoBreak	Ent	5	10.00	1	0	0	39	62	49
SmallNoBreak	Ent	10	50.00	0	0	0	82	82	82
SmallNoBreak	Ent	18	50.00	0	0	0	82	82	82
AnyBreak	GCP	2	2.63	10	32	6	0	100	46
AnyBreak	GCP	3	5.00	0	14	2	0	100	63
AnyBreak	GCP	5	7.14	0	0	0	70	100	86
AnyBreak	GCP	10	16.67	0	0	0	88	100	93
AnyBreak	GCP	18	25.00	0	25	1	0	100	50
AnyNoBreak	GCP	2	25.00	0	0	0	70	100	94
AnyNoBreak	GCP	3	50.00	0	0	0	100	100	100
AnyNoBreak	GCP	5	50.00	0	0	0	100	100	100

Continued on next page

Table A.4 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyNoBreak	GCP	10	50.00	0	0	0	100	100	100
AnyNoBreak	GCP	18	50.00	0	0	0	100	100	100
SmallBreak	GCP	2	2.94	5	21	1	0	100	60
SmallBreak	GCP	3	4.17	4	19	1	0	100	61
SmallBreak	GCP	5	8.33	1	5	0	68	100	87
SmallBreak	GCP	10	25.00	0	25	1	0	100	50
SmallBreak	GCP	18	25.00	0	0	0	90	100	95
SmallNoBreak	GCP	2	8.33	2	5	1	0	100	84
SmallNoBreak	GCP	3	12.50	1	9	0	50	100	87
SmallNoBreak	GCP	5	50.00	0	0	0	100	100	100
SmallNoBreak	GCP	10	50.00	0	0	0	100	100	100
SmallNoBreak	GCP	18	50.00	0	0	0	100	100	100

Table A.5: Table of Results for the four algorithms when using a uniform distribution and small sized database. The columns are labeled as follows: **0:** Algorithm, **1:** Metric, **2:** K value, **3:** Average set size, **4:** # of sets with size K, **5:** Members with at least one not generalized attribute, **6:** Members with no generalized attribute, **7:** Least generalized member by %, **8:** Most generalized member by %, **9:** Average Generalization by % .

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyBreak	AnTree	2	2.94	5	9	0	55	100	81
AnyBreak	AnTree	3	4.17	2	4	0	64	100	84
AnyBreak	AnTree	5	8.33	0	0	0	85	100	90
AnyBreak	AnTree	10	50.00	0	0	0	100	100	100

Continued on next page

Table A.5 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyBreak	AnTree	18	50.00	0	0	0	100	100	100
AnyNoBreak	AnTree	2	16.67	1	0	0	72	100	97
AnyNoBreak	AnTree	3	50.00	0	0	0	100	100	100
AnyNoBreak	AnTree	5	50.00	0	0	0	100	100	100
AnyNoBreak	AnTree	10	50.00	0	0	0	100	100	100
AnyNoBreak	AnTree	18	50.00	0	0	0	100	100	100
SmallBreak	AnTree	2	2.94	5	4	1	0	100	81
SmallBreak	AnTree	3	5.56	2	0	0	85	100	90
SmallBreak	AnTree	5	7.14	3	0	0	78	100	87
SmallBreak	AnTree	10	50.00	0	0	0	100	100	100
SmallBreak	AnTree	18	25.00	0	0	0	88	100	94
SmallNoBreak	AnTree	2	4.17	9	8	0	21	100	83
SmallNoBreak	AnTree	3	50.00	0	0	0	100	100	100
SmallNoBreak	AnTree	5	50.00	0	0	0	100	100	100
SmallNoBreak	AnTree	10	50.00	0	0	0	100	100	100
SmallNoBreak	AnTree	18	50.00	0	0	0	100	100	100
AnyBreak	Ent	2	2.63	7	10	0	15	40	31
AnyBreak	Ent	3	4.17	3	0	0	35	63	49
AnyBreak	Ent	5	7.14	1	0	0	46	78	59
AnyBreak	Ent	10	12.50	1	0	0	74	93	82
AnyBreak	Ent	18	25.00	0	0	0	92	93	92
AnyNoBreak	Ent	2	25.00	0	0	0	38	100	95
AnyNoBreak	Ent	3	16.67	0	0	0	43	92	86
AnyNoBreak	Ent	5	25.00	1	0	0	42	98	92
AnyNoBreak	Ent	10	50.00	0	0	0	100	100	100
AnyNoBreak	Ent	18	50.00	0	0	0	100	100	100
SmallBreak	Ent	2	2.50	10	8	0	16	42	30
SmallBreak	Ent	3	4.17	2	4	0	28	51	41

Continued on next page

Table A.5 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
SmallBreak	Ent	5	7.14	0	0	0	45	72	59
SmallBreak	Ent	10	12.50	0	0	0	72	82	76
SmallBreak	Ent	18	25.00	0	0	0	94	96	95
SmallNoBreak	Ent	2	3.85	1	0	0	26	60	40
SmallNoBreak	Ent	3	4.55	5	0	0	28	71	47
SmallNoBreak	Ent	5	12.50	0	0	0	59	85	76
SmallNoBreak	Ent	10	50.00	0	0	0	100	100	100
SmallNoBreak	Ent	18	50.00	0	0	0	100	100	100
AnyBreak	GCP	2	2.78	5	17	0	55	100	79
AnyBreak	GCP	3	4.55	0	5	0	62	100	84
AnyBreak	GCP	5	8.33	0	0	0	88	100	92
AnyBreak	GCP	10	25.00	0	25	0	25	100	62
AnyBreak	GCP	18	25.00	0	25	1	0	88	44
AnyNoBreak	GCP	2	50.00	0	0	0	100	100	100
AnyNoBreak	GCP	3	50.00	0	0	0	100	100	100
AnyNoBreak	GCP	5	50.00	0	0	0	100	100	100
AnyNoBreak	GCP	10	50.00	0	0	0	100	100	100
AnyNoBreak	GCP	18	50.00	0	0	0	100	100	100
SmallBreak	GCP	2	2.78	7	11	0	52	100	78
SmallBreak	GCP	3	5.00	0	0	0	83	100	88
SmallBreak	GCP	5	10.00	2	0	0	88	100	94
SmallBreak	GCP	10	25.00	0	0	0	88	100	94
SmallBreak	GCP	18	25.00	0	0	0	91	100	95
SmallNoBreak	GCP	2	5.00	7	6	0	40	100	85
SmallNoBreak	GCP	3	12.50	1	0	0	74	100	96
SmallNoBreak	GCP	5	50.00	0	0	0	100	100	100
SmallNoBreak	GCP	10	50.00	0	0	0	100	100	100
SmallNoBreak	GCP	18	50.00	0	0	0	100	100	100

Table A.6: Table of Results for the four algorithms when using a Poisson distribution with $\lambda = 4.0$ and small sized database. The columns are labeled as follows: **0:** Algorithm, **1:** Metric, **2:** K value, **3:** Average set size, **4:** # of sets with size K, **5:** Members with at least one not generalized attribute, **6:** Members with no generalized attribute, **7:** Least generalized member by %, **8:** Most generalized member by %, **9:** Average Generalization by % .

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyBreak	AnTree	2	2.78	6	25	4	0	86	51
AnyBreak	AnTree	3	4.55	2	7	0	27	86	64
AnyBreak	AnTree	5	7.14	1	12	0	36	86	64
AnyBreak	AnTree	10	16.67	0	0	0	66	86	75
AnyBreak	AnTree	18	25.00	0	0	0	77	86	81
AnyNoBreak	AnTree	2	50.00	0	0	0	86	86	86
AnyNoBreak	AnTree	3	50.00	0	0	0	86	86	86
AnyNoBreak	AnTree	5	50.00	0	0	0	86	86	86
AnyNoBreak	AnTree	10	50.00	0	0	0	86	86	86
AnyNoBreak	AnTree	18	50.00	0	0	0	86	86	86
SmallBreak	AnTree	2	2.78	6	23	1	0	86	55
SmallBreak	AnTree	3	5.00	0	14	0	36	77	62
SmallBreak	AnTree	5	7.14	1	0	0	61	86	71
SmallBreak	AnTree	10	12.50	1	11	0	20	86	62
SmallBreak	AnTree	18	25.00	0	0	0	50	86	68
SmallNoBreak	AnTree	2	10.00	1	3	1	0	77	68
SmallNoBreak	AnTree	3	16.67	1	3	1	0	86	78
SmallNoBreak	AnTree	5	50.00	0	0	0	86	86	86
SmallNoBreak	AnTree	10	25.00	0	0	0	77	86	80

Continued on next page

Table A.6 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
SmallNoBreak	AnTree	18	50.00	0	0	0	86	86	86
AnyBreak	Ent	2	2.63	8	13	0	6	38	25
AnyBreak	Ent	3	5.00	0	5	0	26	50	39
AnyBreak	Ent	5	6.25	2	0	0	30	55	42
AnyBreak	Ent	10	16.67	0	0	0	54	65	60
AnyBreak	Ent	18	25.00	0	0	0	64	77	71
AnyNoBreak	Ent	2	7.14	1	5	0	14	53	45
AnyNoBreak	Ent	3	25.00	0	0	0	42	74	70
AnyNoBreak	Ent	5	25.00	0	0	0	41	75	70
AnyNoBreak	Ent	10	50.00	0	0	0	77	77	77
AnyNoBreak	Ent	18	50.00	0	0	0	77	77	77
SmallBreak	Ent	2	2.50	11	23	0	10	35	22
SmallBreak	Ent	3	3.85	6	12	1	0	39	29
SmallBreak	Ent	5	6.25	3	0	0	34	66	46
SmallBreak	Ent	10	12.50	2	0	0	52	73	58
SmallBreak	Ent	18	16.67	0	0	0	58	77	68
SmallNoBreak	Ent	2	3.57	6	12	0	8	44	30
SmallNoBreak	Ent	3	5.56	4	0	0	27	56	40
SmallNoBreak	Ent	5	12.50	0	0	0	39	59	53
SmallNoBreak	Ent	10	25.00	0	0	0	66	66	66
SmallNoBreak	Ent	18	50.00	0	0	0	77	77	77
AnyBreak	GCP	2	2.63	8	27	4	0	86	49
AnyBreak	GCP	3	4.17	2	3	0	45	86	66
AnyBreak	GCP	5	10.00	0	15	1	0	86	63
AnyBreak	GCP	10	12.50	2	20	0	51	77	62
AnyBreak	GCP	18	25.00	0	0	0	66	86	76
AnyNoBreak	GCP	2	16.67	1	0	0	53	86	77
AnyNoBreak	GCP	3	50.00	0	0	0	86	86	86

Continued on next page

Table A.6 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyNoBreak	GCP	5	50.00	0	0	0	86	86	86
AnyNoBreak	GCP	10	50.00	0	0	0	86	86	86
AnyNoBreak	GCP	18	50.00	0	0	0	86	86	86
SmallBreak	GCP	2	2.50	10	22	5	0	77	47
SmallBreak	GCP	3	5.00	0	4	0	52	86	68
SmallBreak	GCP	5	8.33	0	0	0	61	86	72
SmallBreak	GCP	10	16.67	0	0	0	70	86	77
SmallBreak	GCP	18	25.00	0	0	0	77	86	81
SmallNoBreak	GCP	2	8.33	2	7	1	0	77	68
SmallNoBreak	GCP	3	25.00	0	0	0	77	86	84
SmallNoBreak	GCP	5	50.00	0	0	0	86	86	86
SmallNoBreak	GCP	10	50.00	0	0	0	86	86	86
SmallNoBreak	GCP	18	50.00	0	0	0	86	86	86

Table A.7: Table of Results for the four algorithms when using a Poisson distribution with $\lambda = 2.5$ and small sized database. The columns are labeled as follows: **0:** Algorithm, **1:** Metric, **2:** K value, **3:** Average set size, **4:** # of sets with size K, **5:** Members with at least one not generalized attribute, **6:** Members with no generalized attribute, **7:** Least generalized member by %, **8:** Most generalized member by %, **9:** Average Generalization by % .

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyNoBreak	AnTree	2	50.00	0	0	0	77	77	77
AnyNoBreak	AnTree	3	50.00	0	0	0	77	77	77
AnyNoBreak	AnTree	5	50.00	0	0	0	77	77	77

Continued on next page

Table A.7 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyNoBreak	AnTree	10	50.00	0	0	0	77	77	77
AnyNoBreak	AnTree	18	50.00	0	0	0	77	77	77
SmallBreak	AnTree	2	2.63	8	15	2	0	77	52
SmallBreak	AnTree	3	4.55	2	3	0	45	77	62
SmallBreak	AnTree	5	10.00	0	0	0	53	77	70
SmallBreak	AnTree	10	12.50	1	0	0	66	77	71
SmallBreak	AnTree	18	25.00	0	0	0	69	77	73
SmallNoBreak	AnTree	2	12.50	3	0	0	42	77	73
SmallNoBreak	AnTree	3	12.50	3	0	0	42	77	73
SmallNoBreak	AnTree	5	50.00	0	0	0	77	77	77
SmallNoBreak	AnTree	10	50.00	0	0	0	77	77	77
SmallNoBreak	AnTree	18	50.00	0	0	0	77	77	77
AnyNoBreak	Ent	2	50.00	0	0	0	68	68	68
AnyNoBreak	Ent	3	50.00	0	0	0	68	68	68
AnyNoBreak	Ent	5	50.00	0	0	0	68	68	68
AnyNoBreak	Ent	10	50.00	0	0	0	68	68	68
AnyNoBreak	Ent	18	50.00	0	0	0	68	68	68
SmallBreak	Ent	2	2.17	19	22	0	8	33	21
SmallBreak	Ent	3	3.85	6	4	0	24	50	34
SmallBreak	Ent	5	7.14	0	0	0	41	59	49
SmallBreak	Ent	10	16.67	0	0	0	57	59	58
SmallBreak	Ent	18	25.00	0	0	0	60	64	62
SmallNoBreak	Ent	2	2.94	8	17	0	3	38	26
SmallNoBreak	Ent	3	5.56	2	3	0	18	53	39
SmallNoBreak	Ent	10	50.00	0	0	0	68	68	68
SmallNoBreak	Ent	18	50.00	0	0	0	68	68	68

Table A.8: Table of Results for the four algorithms when using a Poisson distribution with $\lambda = 1.0$ and small sized database. The columns are labeled as follows: **0:** Algorithm, **1:** Metric, **2:** K value, **3:** Average set size, **4:** # of sets with size K, **5:** Members with at least one not generalized attribute, **6:** Members with no generalized attribute, **7:** Least generalized member by %, **8:** Most generalized member by %, **9:** Average Generalization by % .

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyBreak	AnTree	2	2.50	10	29	8	0	65	24
AnyBreak	AnTree	3	3.85	5	20	1	0	70	39
AnyBreak	AnTree	5	8.33	0	0	0	45	65	55
AnyBreak	AnTree	10	12.50	0	0	0	46	65	54
AnyBreak	AnTree	18	25.00	0	0	0	53	65	59
AnyNoBreak	AnTree	2	25.00	0	0	0	51	65	61
AnyNoBreak	AnTree	3	50.00	0	0	0	70	70	70
AnyNoBreak	AnTree	5	50.00	0	0	0	70	70	70
AnyNoBreak	AnTree	10	50.00	0	0	0	70	70	70
AnyNoBreak	AnTree	18	50.00	0	0	0	70	70	70
SmallBreak	AnTree	2	2.94	7	29	7	0	65	26
SmallBreak	AnTree	3	4.17	4	13	1	0	65	38
SmallBreak	AnTree	5	8.33	0	0	0	33	65	48
SmallBreak	AnTree	10	16.67	0	0	0	48	70	59
SmallBreak	AnTree	18	25.00	0	0	0	48	70	59
SmallNoBreak	AnTree	2	3.57	7	18	2	0	62	35
SmallNoBreak	AnTree	3	8.33	0	4	0	23	70	51
SmallNoBreak	AnTree	5	10.00	0	0	0	33	70	54
SmallNoBreak	AnTree	10	50.00	0	0	0	70	70	70
SmallNoBreak	AnTree	18	50.00	0	0	0	70	70	70

Continued on next page

Table A.8 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyBreak	Ent	2	2.78	6	23	0	3	29	20
AnyBreak	Ent	3	4.17	3	13	0	3	45	28
AnyBreak	Ent	5	7.14	2	0	0	33	39	36
AnyBreak	Ent	10	12.50	1	0	0	39	47	42
AnyBreak	Ent	18	25.00	0	0	0	47	49	47
AnyNoBreak	Ent	2	50.00	0	0	0	50	50	50
AnyNoBreak	Ent	3	12.50	0	0	0	35	39	37
AnyNoBreak	Ent	5	50.00	0	0	0	50	50	50
AnyNoBreak	Ent	10	50.00	0	0	0	50	50	50
AnyNoBreak	Ent	18	50.00	0	0	0	50	50	50
SmallBreak	Ent	2	2.50	11	30	2	0	28	15
SmallBreak	Ent	3	4.17	3	20	0	9	42	26
SmallBreak	Ent	5	6.25	2	8	0	20	45	34
SmallBreak	Ent	10	16.67	0	0	0	39	47	43
SmallBreak	Ent	18	25.00	0	0	0	47	50	48
SmallNoBreak	Ent	2	3.13	7	22	3	0	36	20
SmallNoBreak	Ent	3	7.14	1	7	0	16	41	32
SmallNoBreak	Ent	5	25.00	0	0	0	41	50	48
SmallNoBreak	Ent	10	25.00	0	0	0	45	48	45
SmallNoBreak	Ent	18	50.00	0	0	0	50	50	50

A.4 Extra Tables for Large Sized Databases

Table A.9: Table of Results for the four algorithms when using a log-Gaussian and small sized database. The columns are labeled as follows: **0:** Algorithm, **1:** Metric, **2:** K value, **3:** Average set size, **4:** # of sets with size K, **5:** Members with at least one not generalized attribute, **6:** Members with no generalized attribute, **7:** Least generalized member by %, **8:** Most generalized member by %, **9:** Average Generalization by % .

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyBreak	AnTree	2	8.87	2	4667	460	0	100	9
AnyBreak	AnTree	3	11.66	7	4063	259	0	100	22
AnyBreak	AnTree	5	15.15	1	3384	161	0	100	32
AnyBreak	AnTree	10	29.41	0	2348	58	0	100	46
AnyBreak	AnTree	18	37.88	2	2103	45	0	100	48
AnyNoBreak	AnTree	2	625.00	1	49	2	0	100	98
AnyNoBreak	AnTree	3	5000.00	0	0	0	100	100	100
AnyNoBreak	AnTree	5	5000.00	0	0	0	100	100	100
AnyNoBreak	AnTree	10	5000.00	0	0	0	100	100	100
AnyNoBreak	AnTree	18	5000.00	0	0	0	100	100	100
SmallBreak	AnTree	2	6.93	237	4635	527	0	100	7
SmallBreak	AnTree	3	8.91	149	4296	352	0	100	14
SmallBreak	AnTree	5	13.77	60	3822	186	0	100	21
SmallBreak	AnTree	10	23.47	16	3088	87	0	100	34
SmallBreak	AnTree	18	37.04	12	2726	42	0	100	38
SmallNoBreak	AnTree	2	10.33	168	3115	409	0	100	39
SmallNoBreak	AnTree	3	16.56	77	2858	239	0	100	46
SmallNoBreak	AnTree	5	30.49	30	2106	129	0	100	57
SmallNoBreak	AnTree	10	69.44	2	1876	42	0	100	64
SmallNoBreak	AnTree	18	119.05	2	1235	17	0	100	72
AnyBreak	Ent	2	6.32	226	2953	317	0	62	14

Continued on next page

Table A.9 – continued from previous page

# 0	# 1	# 2	# 3	# 4	# 5	# 6	# 7	# 8	# 9
AnyBreak	Ent	3	8.68	80	2368	163	0	71	18
AnyBreak	Ent	5	11.52	14	1914	74	0	87	27
AnyBreak	Ent	10	20.83	0	1178	30	0	100	47
AnyBreak	Ent	18	31.85	0	819	13	0	100	62
AnyNoBreak	Ent	2	128.21	5	50	5	0	100	91
AnyNoBreak	Ent	3	131.58	2	104	1	0	100	88
AnyNoBreak	Ent	5	714.29	0	0	0	69	100	97
AnyNoBreak	Ent	10	5000.00	0	0	0	100	100	100
AnyNoBreak	Ent	18	1666.67	0	0	0	85	100	98
SmallBreak	Ent	2	6.39	278	3054	390	0	60	12
SmallBreak	Ent	3	7.70	236	2754	245	0	70	16
SmallBreak	Ent	5	11.29	111	2210	129	0	79	22
SmallBreak	Ent	10	17.99	40	1727	58	0	97	34
SmallBreak	Ent	18	28.25	21	1474	31	0	100	47
SmallNoBreak	Ent	2	7.46	191	3011	357	0	53	14
SmallNoBreak	Ent	3	10.40	113	2502	226	0	73	19
SmallNoBreak	Ent	5	16.45	56	2108	130	0	90	30
SmallNoBreak	Ent	10	31.45	8	1573	58	0	96	42
SmallNoBreak	Ent	18	49.02	5	1210	32	0	96	56
AnyNoBreak	GCP	2	5000.00	0	0	0	100	100	100
SmallBreak	GCP	2	6.84	247	4669	535	0	100	7
SmallNoBreak	GCP	2	10.37	166	3233	409	0	100	38

APPENDIX B
CODE FOR ANALYSIS OF ALGORITHMS

B.1 Main.java

The main class for the project.

```
1  import genTree.GenTree;
2
3  import java.io.File;
4  import java.util.ArrayList;
5  import java.util.Arrays;
6  import java.util.HashMap;
7
8  import dataBase.DB;
9  import dataBase.ImportantDBFunction;
10 import kAnonymity.KAnonymitySupport;
11 import kAnonymity.MondrianKAnonymity;
12 import kAnonymity.MyKAnonymity.K_Split_Algorithm;
13 import kAnonymity.MyKAnonymity.K_Split_Algorithm2;
14 import kAnonymity.MyKAnonymity.MyKAnonymity;
15 import kAnonymity.MyKAnonymity.MyKAnonymityTwo;
16 import kAnonymity.OptimalKAnonymity.OptimalKAnonymity;
17
18 public class Main {
19  public static void main(String[] args){
20    DB DataBase = null;
21    int TreeMetric = 0;
22    if(args[0].compareTo("0") == 0) {
23     int temp =  Integer.parseInt(args[1]);
24     String temp2 = args[2];
25     ImportantDBFunction.buildDB(temp,temp2,null,false);
26    }
27    else if(args[0].compareTo("7") == 0){
28     ArrayList<String> head = ImportantDBFunction.LoadHeaderFile(args[1]);
29     DataBase = ImportantDBFunction.LoadDB(head,args[1]);
30     DataBase.GenTreesForAttr = GenTree.MakeAllGenTrees(new ArrayList<String>(
           DataBase.Data.PidAttrAndRanges.keySet()), args[2]);
31     KAnonymitySupport.MakeStatistics(DataBase,args[1],null,false);
32    }
33    else if(args[0].compareTo("8") == 0){
34     boolean flaggy = true;
35     HashMap<String, ArrayList<File>> thefilepairs = new HashMap<String, ArrayList<
           File>>();
```

```
36      File Directory = new File(args[1]);
37      if(Directory.isDirectory()){
38       for(File a : Directory.listFiles()){
39        String[] temp = a.toString().split("/");
40        String temp2 = "";
41         if(temp[1].contains(".header")) temp2 =  (String) temp[1].subSequence(0,
              temp[1].length()-7);
42         else if (temp[1].contains(".csv")) temp2 = (String) temp[1].subSequence(0,
              temp[1].length()-4);
43        if(thefilepairs.containsKey(temp2)) thefilepairs.get(temp2).add(a);
44        else {
45         thefilepairs.put(temp2,new ArrayList<File>());
46         thefilepairs.get(temp2).add(a);
47        }
48       }
49      }
50      else return;
51      for(String a : thefilepairs.keySet()){
52       ArrayList<String> head = null;
53       DataBase = null ;
54       String FileHandle = "";
55       String FileHandle1 = "";
56       for(File b : thefilepairs.get(a)){
57        if(b.toString().contains(".header")){
58         String temp = b.toString().split("/")[1];
59         FileHandle1 = b.toString().substring(0, b.toString().length()-7);
60         FileHandle = temp.substring(0,temp.length()-7);
61        }
62        else if(b.toString().contains(".csv")){
63         String temp = b.toString().split("/")[1];
64         FileHandle1 = b.toString().substring(0, b.toString().length()-4);
65         FileHandle = temp.substring(0,temp.length()-4);
66        }
67           break;
68       }
69       head =  ImportantDBFunction.LoadHeaderFile(FileHandle1);
70       DataBase = ImportantDBFunction.LoadDB(head,FileHandle1);
71       DataBase.GenTreesForAttr = GenTree.MakeAllGenTrees(new ArrayList<String>(
             DataBase.Data.PidAttrAndRanges.keySet()), args[2]);
72       KAnonymitySupport.MakeStatistics(DataBase,FileHandle,args[3],flaggy);
73       flaggy = false;
74      }
75     }
76     else if(args[0].compareTo("9") == 0){
77      boolean flaggy = true;
78      HashMap<String, ArrayList<File>> thefilepairs = new HashMap<String, ArrayList<
             File>>();
```

```
79      File Directory = new File(args[1]);
80      if(Directory.isDirectory()){
81       for(File a : Directory.listFiles()){
82        if(a.toString().contains("Small"))
83         ImportantDBFunction.buildDB(50,a.toString(),args[2],true);
84        else if(a.toString().contains("Medium"))
85         ImportantDBFunction.buildDB(500,a.toString(),args[2],true);
86        else if(a.toString().contains("Large"))
87         ImportantDBFunction.buildDB(5000,a.toString(),args[2],true);
88       }
89      }
90     }
91     else if(args[0].compareTo("10") == 0){
92      boolean flaggy = true;
93      HashMap<String, ArrayList<File> > thefilepairs = new HashMap<String, ArrayList<
            File> >();
94      File Directory = new File(args[1]);
95      if(Directory.isDirectory()){
96       for(File a : Directory.listFiles()){
97        String[] temp = a.toString().split("/");
98         temp = temp[1].split("[.]{1}");
99        if(thefilepairs.containsKey(temp[0])) thefilepairs.get(temp[0]).add(a);
100       else {
101        thefilepairs.put(temp[0],new ArrayList<File>());
102        thefilepairs.get(temp[0]).add(a);
103       }
104      }
105     }
106     for(String a : thefilepairs.keySet()){
107      ArrayList<String> head = null;
108      DataBase = null ;
109      String FileHandle = "";
110      String FileHandle1 = "";
111      File mainDirec = new File(args[2]);
112      for(File b : thefilepairs.get(a)){
113       if(b.toString().contains(".header")){
114        String temp = b.toString().split("/")[1];
115        FileHandle1 = b.toString().substring(0, b.toString().length()-7);
116        FileHandle = temp.substring(0,temp.length()-7);
117       }
118       else if(b.toString().contains(".csv")){
119        String temp = b.toString().split("/")[1];
120        FileHandle1 = b.toString().substring(0, b.toString().length()-4);
121        FileHandle = temp.substring(0,temp.length()-4);
122       }
123          break;
124      }
```

```
125       int[] kvals = {2,3,5,10,18};
126       String[] met= {"Entro","AnTree","Discern"};
127       head =  ImportantDBFunction.LoadHeaderFile(FileHandle1);
128       DataBase = ImportantDBFunction.LoadDB(head,FileHandle1);
129       DataBase.GenTreesForAttr = GenTree.MakeAllGenTrees(new ArrayList<String>(
              DataBase.Data.PidAttrAndRanges.keySet()), args[3]);
130       for(int i =0 ; i < 2; i++){
131        for(int j : kvals){
132         String [] files = mainDirec.list();
133         Arrays.sort(files);
134         if((Arrays.binarySearch(files, 0, files.length, FileHandle+"_"+j+"_"+"
              AnonymityLowKOnlyAndNoBreak_"+met[i]+".csv")<0)){
135          DB dataKed = MyKAnonymityTwo.MyKAnonymityAlgorithm(DataBase,j,i,false);
136          dataKed.WriteDataBaseToFile(args[2]+"/"+FileHandle+"_"+j+"_"+"
               AnonymityLowKOnlyAndNoBreak_"+met[i]);
137         }
138         if((Arrays.binarySearch(files, 0, files.length, FileHandle+"_"+j+"_"+"
              AnonymityAnySetAndNoBreak_"+met[i]+".csv")<0)){
139          DB dataKed = MyKAnonymityTwo.MyKAnonymityAlgorithm(DataBase,j,i,true);
140          dataKed.WriteDataBaseToFile(args[2]+"/"+FileHandle+"_"+j+"_"+"
               AnonymityAnySetAndNoBreak_"+met[i]);
141         }
142         if((Arrays.binarySearch(files, 0, files.length, FileHandle+"_"+j+"_"+"
              AnonymityLowKOnlyAndBreak_"+met[i]+".csv")<0)){
143          DB dataKed = K_Split_Algorithm.KAnonymityAlgorithm(DataBase,j,i);
144          dataKed.WriteDataBaseToFile(args[2]+"/"+FileHandle+"_"+j+"_"+"
               AnonymityLowKOnlyAndBreak_"+met[i]);
145         }
146         if((Arrays.binarySearch(files, 0, files.length, FileHandle+"_"+j+"_"+"
              AnonymityAnySetAndBreak_"+met[i]+".csv")<0)){
147          DB dataKed = K_Split_Algorithm2.KAnonymityAlgorithm(DataBase,j,i);
148          dataKed.WriteDataBaseToFile(args[2]+"/"+FileHandle+"_"+j+"_"+"
               AnonymityAnySetAndBreak_"+met[i]);
149         }
150        }
151       }
152       }
153      }
154      else{
155       ArrayList<String> head = ImportantDBFunction.LoadHeaderFile(args[1]);
156       DataBase = ImportantDBFunction.LoadDB(head,args[1]);
157       if(args[3].compareTo("AnTree") == 0 || args[3].compareTo("Discern") == 0){
158       if (args[3].compareTo("AnTree") == 0 )TreeMetric = 1;
159       if (args[3].compareTo("Discern") == 0)TreeMetric = 2;
160       if(args[0].compareTo("6") != 0){
161         DataBase.GenTreesForAttr = GenTree.MakeAllGenTrees(new ArrayList<String>(
              DataBase.Data.PidAttrAndRanges.keySet()), args[4]);
```

```
162         }
163
164         }
165         if(args[0].compareTo("1") == 0){
166          DB dataKed = MyKAnonymity.MyKAnonymityAlgorithm(DataBase,new Integer(args[2]),
                 TreeMetric);
167         dataKed.WriteDataBaseToFile(args[1]+"_"+args[2]+"_"+"
                AnonymityLowKOnlyAndNoBreak_"+args[3]);
168         }
169         else if(args[0].compareTo("2") == 0){
170          DB dataKed = MyKAnonymityTwo.MyKAnonymityAlgorithm(DataBase,new Integer(args
                 [2]),TreeMetric,true);
171          dataKed.WriteDataBaseToFile(args[1]+"_"+args[2]+"_"+"
                 AnonymityAnySetAndNoBreak_"+args[3]);
172         }
173         else   if(args[0].compareTo("3") == 0){
174          DB dataKed = K_Split_Algorithm.KAnonymityAlgorithm(DataBase,new Integer(args
                 [2]),TreeMetric);
175          dataKed.WriteDataBaseToFile(args[1]+"_"+args[2]+"_"+"
                 AnonymityLowKOnlyAndBreak_"+args[3]);
176         }
177         else   if(args[0].compareTo("4") == 0){
178          DB dataKed = K_Split_Algorithm2.KAnonymityAlgorithm(DataBase,new Integer(args
                 [2]),TreeMetric);
179          dataKed.WriteDataBaseToFile(args[1]+"_"+args[2]+"_"+"AnonymityAnySetAndBreak_"
                +args[3]);
180         }
181         else   if(args[0].compareTo("5") == 0){
182          DB dataKed = OptimalKAnonymity.OptimalKAnonymityAlgorithm(DataBase,new Integer
                 (args[2]),TreeMetric);
183          dataKed.WriteDataBaseToFile(args[1]+"_"+args[2]+"_"+"Anonymity_Optimal");
184         }
185         else   if(args[0].compareTo("6") == 0){
186          DataBase.GenTreesForAttr = GenTree.MakeAllGenTrees(new ArrayList<String>(
                 DataBase.Data.PidAttrAndRanges.keySet()), args[4]);
187          for(String attr : DataBase.GenTreesForAttr.keySet()){
188           GenTree g = DataBase.GenTreesForAttr.get(attr);
189           System.out.print(g.toString());
190          }
191         }
192        }
193         }
194 }
```

B.2 DB.java

This is the class for database construction.

```
1  package dataBase;
2   import genTree.GenTree;
3
4  import java.io.DataOutputStream;
5  import java.io.File;
6  import java.io.FileOutputStream;
7  import java.io.IOException;
8  import java.io.InputStreamReader;
9  import java.io.OutputStreamWriter;
10 import java.util.ArrayList;
11 import java.util.Date;
12 import java.util.HashMap;
13 import java.util.Random;
14
15 import kAnonymity.KAnonymitySupport;
16
17 public class DB {
18
19  public DB(DBData data, Mapper maps) {
20   Data = data;
21   Maps = maps;
22   NumberofMembers = 0;
23   MembersList = new HashMap<HashMap<String,String>,Members>();
24   Occurances = new  HashMap<String, HashMap<String,Integer> >();
25   for(String Attr : Maps.RangeMaps.keySet()){
26    if(!Data.UidAttrAndRanges.keySet().contains(Attr)){
27     Occurances.put(Attr, new HashMap<String,Integer>());
28     for(Integer inte: Maps.InverseMappings.get(Attr).keySet())
29       Occurances.get(Attr).put(Maps.InverseMappings.get(Attr).get(inte), 0);
30    }
31   }
32   time = new Date();
33   rand = new Random(time.getTime());
34   AttrEntropy = new HashMap<String,Double>();
35   GenTreesForAttr = new HashMap<String, GenTree>();
36  }
37  public DB(DB dataBase) {
38   MembersList = new  HashMap<HashMap<String,String>,Members>();
39   Occurances = new HashMap<String, HashMap<String,Integer> >();
40   GenTreesForAttr = new HashMap<String,GenTree>();
41   Data = new DBData(dataBase.Data);
42   Maps = new Mapper(dataBase.Maps);
43   AttrEntropy = new  HashMap<String,Double>();
44   for( String a : dataBase.AttrEntropy.keySet())
```

```
45      AttrEntropy.put(new String(a), new Double(dataBase.AttrEntropy.get(a)));
46     for(HashMap<String,String> a : dataBase.MembersList.keySet()){
47      HashMap<String, String> temp = new HashMap<String, String>();
48      for(String b : a.keySet()) temp.put(new String(b), new String(a.get(b)));
49      MembersList.put(temp, new Members(dataBase.MembersList.get(a)));
50     }
51     for(String Attr: dataBase.Occurances.keySet()){
52      HashMap<String,Integer> temp = new HashMap<String,Integer>();
53      for(String val : dataBase.Occurances.get(Attr).keySet()){
54       temp.put(new String(val), new Integer( dataBase.Occurances.get(Attr).get(val).
             intValue()));
55      }
56      Occurances.put(new String(Attr), temp);
57     }
58     for(String a : dataBase.GenTreesForAttr.keySet()){
59      GenTreesForAttr.put(new String(a), new GenTree(dataBase.GenTreesForAttr.get(a)))
            ;
60     }
61     rand = dataBase.rand;
62     time = dataBase.time;
63    }
64    public void fillGenTreesAttr(String location){
65     GenTreesForAttr = GenTree.MakeAllGenTrees(new ArrayList<String>(Data.
           PidAttrAndRanges.keySet()), location);
66    }
67    public void SetEntropy(){
68      for(String Attr : this.Occurances.keySet()){
69       double temp= KAnonymitySupport.enthropy(this.Occurances.get(Attr), this.
             NumberofMembers);
70       AttrEntropy.put(Attr, temp);
71       }
72    }
73    Members GetAMember(int MemberId){ return MembersList.get(MemberId);}
74    ArrayList<Members> GetNMembers(ArrayList<Integer> MemberId){
75     ArrayList<Members> ReturnMembers = new ArrayList<Members>();
76     for(Integer Member : MemberId){
77      ReturnMembers.add(MembersList.get(Member));
78     }
79     return ReturnMembers;
80    }
81    public boolean  AddAMember(Members M){
82     if(M.IsComplete()){
83      MembersList.put(M.GetUID(Data), M);
84      for(String Attr : Data.AttrNames){
85       if(Data.UidAttrAndRanges.keySet().contains(Attr) == false){
86        int temp = Occurances.get(Attr).get(M.GetAttributeValue(Attr)).intValue();
87        temp++;
```

```
88          Occurances.get(Attr).put(M.GetAttributeValue(Attr), temp);
89         }
90        }
91        return true;
92       }
93       else{
94
95        System.out.print(M.toString()+"   A member without complete attributes cannot
              enter the database\n");
96        System.out.println(M.GetAllAttributesSetValues().toString());
97        return false;
98       }
99      }
100     public boolean RemoveAMember(Members M){
101      if(M.IsComplete()){
102       if(MembersList.containsKey(M.GetUID(Data))){
103        for(String Attr : Data.AttrNames){
104         if(Data.UidAttrAndRanges.keySet().contains(Attr) == false){
105          int temp = Occurances.get(Attr).get(M.GetAttributeValue(Attr)).intValue();
106          temp--;
107          Occurances.get(Attr).put(M.GetAttributeValue(Attr), temp);
108         }
109        }
110        NumberofMembers--;
111        return true;
112       }
113       else return false;
114      }
115      else{
116       System.out.print(M.toString()+"   A member without complete attributes cannot
              enter the database\n");
117       System.out.println(M.GetAllAttributesSetValues().toString());
118       return false;
119      }
120     }
121     void AddAMember(HashMap<String,String> MAttrVal,HashMap<String,String>UID){
122      if(MAttrVal.size() == (Data.AttrNames.size()- Data.UidAttrAndRanges.keySet().size
           ())){
123       Members M = new Members(Data.AttrNames.size(),UID,Data,Maps);
124       M.AssignGroupAttribute(MAttrVal, Maps);
125       if(AddAMember(M)){
126        NumberofMembers++;
127       }
128      }
129     }
130     void AddRandomMember(ArrayList<String> attr,HashMap<String,String>UID){
131       Members M = new Members(Data.AttrNames.size(),UID,Data, Maps);
```

```
132     M.AssignGroupRandomAttribute(attr, rand, Data,Maps);
133     if(AddAMember(M)) NumberofMembers++;
134   }
135   void AddRandomGaussianMember(ArrayList<String> attr,HashMap<String,String>UID){
136    Members M = new Members(Data.AttrNames.size(),UID,Data, Maps);
137    M.AssignGroupRandomGaussianAttribute(attr, rand, Data,Maps);
138    if(AddAMember(M)) NumberofMembers++;
139  }
140   public void AddRandomPoissonMember(ArrayList<String> attr,
141     HashMap<String, String> UID,double lambda) {
142    Members M = new Members(Data.AttrNames.size(),UID,Data, Maps);
143    M.AssignGroupRandomPoissonAttribute(attr, rand, Data,Maps, lambda);
144    if(AddAMember(M)) NumberofMembers++;
145
146   }
147   public void AddRandomLogGaussianMember(ArrayList<String> attr,
148     HashMap<String, String> UID) {
149    Members M = new Members(Data.AttrNames.size(),UID,Data, Maps);
150    M.AssignGroupRandomLogGaussianAttribute(attr, rand, Data,Maps);
151    if(AddAMember(M)) NumberofMembers++;
152   }
153   ArrayList<Members> GetAllMembersWith(HashMap<String,String> AttrAndValues){
154    ArrayList<Members> ReturnValue = new ArrayList<Members>();
155    int i = 0, k=0;
156    for(HashMap<String,String> key : MembersList.keySet()){
157     k =0;
158     for(String attr: AttrAndValues.keySet()){
159      if(MembersList.get(key).GetAttributeValue(attr).compareTo(AttrAndValues.get(
            attr)) == 0 )i++;
160      k++;
161     }
162
163     if(i == AttrAndValues.keySet().size()){
164      ReturnValue.add(MembersList.get(key));
165     }
166     i =k= 0;
167    }
168    return ReturnValue;
169   }
170   public void WriteDataBaseToFile(String FileName){
171    File file;
172    FileOutputStream fos = null;
173    OutputStreamWriter oos = null;
174       String temp = "";
175       boolean flag = false;
176       try{
177        file= new File(FileName.concat(".csv"));
```

```
178         fos = new FileOutputStream(file);
179         oos=new  OutputStreamWriter(fos,"UTF-8");
180        }catch (IOException e) { e.printStackTrace();}
181        for(HashMap<String,String> key :MembersList.keySet()){
182         temp = "";
183         flag = false;
184         HashMap<String,String> tempattrpairs =  MembersList.get(key).GetAllAttributes
                ();
185         for( String Attr: tempattrpairs.keySet()){
186          if(flag == false){
187           temp += Attr+":"+tempattrpairs.get(Attr);
188           flag = true;
189          }
190          else{
191           temp += ',';
192           temp += Attr+":"+tempattrpairs.get(Attr);
193          }
194         }
195         temp += '\n';
196         try{
197          oos.write(temp.toCharArray());
198         }catch (IOException e) {
199                 e.printStackTrace();
200            }
201        }
202        try{oos.close();}catch (IOException e) {e.printStackTrace();}
203           try{
204         file= new File(FileName.concat(".header"));
205         fos = new FileOutputStream(file);
206         oos=new OutputStreamWriter(fos,"UTF-8");
207        }catch (IOException e) {e.printStackTrace();}
208    temp = new String();
209    flag = false;
210    for( String Attr: Data.AttrNames){
211     if(flag == false){
212      temp ="NOA"+','+Attr;
213      flag = true;
214     }
215     else{
216      temp += ',';
217      temp += Attr;
218     }
219    }
220    temp += '\n';
221    try{
222     oos.write(temp.toCharArray());
223    }catch (IOException e) {
```

```
224             e.printStackTrace();
225         }
226         try{
227          oos.write(("NOM"+','+NumberofMembers+"\n").toCharArray());
228
229        }catch (IOException e) {
230             e.printStackTrace();
231         }
232        temp = new String();
233        flag = false;
234        for( String Attr: Maps.RangeMaps.keySet()){
235         if(flag == false){
236          temp = "RANGE"+","+Attr+":"+Maps.RangeMaps.get(Attr).toString();
237          flag = true;
238         }
239         else{
240          temp += ',';
241          temp += Attr+":"+Maps.RangeMaps.get(Attr).toString();
242         }
243        }
244        temp += '\n';
245        try{
246         oos.write(temp.toCharArray());
247        }catch (IOException e) {
248             e.printStackTrace();
249         }
250        for(String ATTR: Maps.Mappings.keySet()){
251         if(Data.UidAttrAndRanges.containsKey(ATTR) == true){
252          temp = new String();
253          temp = "MAPS"+","+ATTR+":BYVALUE";
254          temp += '\n';
255          try{
256           oos.write(temp.toCharArray());
257          }catch (IOException e) {
258                 e.printStackTrace();
259             }
260         }
261         else{
262          temp = new String();
263          flag = false;
264          for( String Val :Maps.Mappings.get(ATTR).keySet()){
265           if(flag == false){
266            temp = "MAPS"+","+ATTR+":"+Val+":"+Maps.Mappings.get(ATTR).get(Val).toString
                   ();
267            flag = true;
268           }
269           else{
```

```
270         temp += ',';
271         temp += ATTR+":"+Val+":"+Maps.Mappings.get(ATTR).get(Val).toString();
272        }
273       }
274       temp += '\n';
275       try{
276        oos.write(temp.toCharArray());
277       }catch (IOException e) {
278              e.printStackTrace();
279          }
280      }
281     }
282     temp = new String();
283     flag = false;
284     String temp2 = "";
285     for( String attr :Data.AttrNames){
286      if(Data.DataAttrAndRanges.containsKey(attr) == true) temp2 = "DATA";
287      else if (Data.PidAttrAndRanges.containsKey(attr) == true) temp2 = "PID";
288      else if (Data.UidAttrAndRanges.containsKey(attr) == true) temp2 = "UID";
289      else System.out.print("Should not be here, this means the attr was not put into
           one of the attribute type");
290      if(flag == false){
291       temp = "ALLOC"+","+temp2+":"+attr;
292       flag = true;
293      }
294      else{
295       temp += ',';
296       temp += temp2+":"+attr;
297      }
298     }
299     temp += '\n';
300     try{
301      oos.write(temp.toCharArray());
302     }catch (IOException e) {
303            e.printStackTrace();
304     }
305     try{oos.close();}catch (IOException e) {e.printStackTrace();}
306    }
307   //Members are not to be recognized by a single value but instead by a Uid array
308   //which holds all the information about member in the database such that the
       members are uniquely
309   //associated with that value
310    public HashMap<HashMap<String,String>,Members> MembersList;
311     //Holds the number of occurances of each attribute and what the attributes are
312    public HashMap<String, HashMap<String,Integer>> Occurances;
313    //Holds alot of important information about the database
314    public DBData Data;
```

```
315  //Holds the conversion functions for much of the information it the database
316  public Mapper Maps;
317  //holds the entropy of each attribute in the datbase one dimensional only
318  public  HashMap<String,Double>AttrEntropy;
319  public int NumberofMembers;// The number of member in the database
320  private Random rand;
321  private Date time;
322  public HashMap<String, GenTree> GenTreesForAttr; //need to add to constructors
323 }
```

B.3 DBData.java

This is the class for important database information.

```
1  package dataBase;
2
3  import java.util.ArrayList;
4  import java.util.HashMap;
5
6  public class DBData {
7   DBData(){
8    UidAttrAndRanges = new HashMap<String,Integer>();
9    PidAttrAndRanges = new HashMap<String,Integer>();
10   DataAttrAndRanges = new HashMap<String,Integer>();
11   AttrNames = new ArrayList<String>();
12  }
13 public DBData(DBData data) {
14  UidAttrAndRanges = new HashMap<String,Integer>();
15  PidAttrAndRanges = new HashMap<String,Integer>();
16  DataAttrAndRanges = new HashMap<String,Integer>();
17  AttrNames = new ArrayList<String>();
18  for(String a : data.UidAttrAndRanges.keySet())
19   UidAttrAndRanges.put(new String(a), new Integer(data.UidAttrAndRanges.get(a).
        intValue()));
20  for(String a : data.PidAttrAndRanges.keySet())
21   PidAttrAndRanges.put(new String(a), new Integer(data.PidAttrAndRanges.get(a).
        intValue()));
22  for(String a : data.DataAttrAndRanges.keySet())
23   DataAttrAndRanges.put(new String(a), new Integer(data.DataAttrAndRanges.get(a).
        intValue()));
24  for(String a : data.AttrNames)
25   AttrNames.add(new String(a));
26 }
27  // This reads in the attributes which are part of the unique identifier
28  public HashMap<String,Integer> UidAttrAndRanges;
29  //This reads in the range for attributes in the unique identifier
```

```
30   //This reads in the attributes which are part of the PseudoIndentifier
31   public HashMap<String,Integer> PidAttrAndRanges;
32   //This read in the attributes which are part of the data
33   public HashMap<String,Integer> DataAttrAndRanges;
34   //This holds the attribute names
35   public ArrayList<String> AttrNames;
36  }
```

B.4 ImportantDBFunction.java

This is the class for important database functionality.

```
1   package dataBase;
2   import java.io.DataInputStream;
3   import java.io.EOFException;
4   import java.io.File;
5   import java.io.FileInputStream;
6   import java.io.FileNotFoundException;
7   import java.io.IOException;
8   import java.io.InputStreamReader;
9   import java.util.ArrayList;
10  import java.util.HashMap;
11
12  public final class ImportantDBFunction {
13   public static ArrayList<String> LoadHeaderFile(String TheFile){
14    File file;
15     FileInputStream FIS = null;
16     DataInputStream DIS;
17     ArrayList<String> header = new  ArrayList<String>();
18     header.add(new String());
19     int i = 0;
20     char single;
21     try{
22          file= new File(TheFile);
23          try {
24        FIS = new FileInputStream(file+".header");
25       } catch (FileNotFoundException e) {e.printStackTrace();}
26       DIS=new DataInputStream(FIS);
27         while(true){
28        single = (char)DIS.readByte();
29          if(single == '\n'){
30           header.add(new String());
31           i++;
32          }
33          else  header.set(i, header.get(i) + single);
34         }
```

```
35
36          }catch (EOFException e) {}
37          catch (IOException e) {e.printStackTrace();}
38          return header;
39      }
40      //This loads the DataBaseloader file and then puts the data into a database value.
41      //I think I need to add in the string values at this point. Most like in the
            loader file.
42      public static DB  buildDB(int numberofmembers, String FileToRead,String
            outputlocation, boolean todo){
43       DB DataBase = null;
44       boolean flag  = false;
45       String FileToSendTo = null;
46       String DistroToUse = null;
47       double lambda = 0.0;
48       DBData Data = new DBData();
49       Mapper maps = new Mapper();
50       ArrayList<String> returnval = ReadFromFile(FileToRead);
51       for(String sss : returnval){
52        System.out.println(sss);
53       }
54       for(int i = 0; i < returnval.size(); i++){
55        if(returnval.get(i).contains("UID")&& !flag){
56         String[] Str =  returnval.get(i).split(",");
57         int j =0;
58         while(Str[j].compareTo("UID") != 0){
59          Data.UidAttrAndRanges.put(Str[j],0);
60          j++;
61         }
62         j++;
63         for(int k = j, r = 0; k < Str.length;k++,r++){
64          maps.RangeMaps.put(Str[r], new Integer(Str[k]));
65          Data.UidAttrAndRanges.put(Str[r],new Integer(Str[k]));
66         }
67        }
68        else if(returnval.get(i).contains("PID")&& !flag){
69         String[] Str =  returnval.get(i).split(",");
70         int j =0;
71         while(Str[j].compareTo("PID") != 0){
72          Data.PidAttrAndRanges.put(Str[j],0);
73          j++;
74         }
75         j++;
76         for(int k = j,r = 0; k < Str.length;k++,r++){
77          maps.RangeMaps.put(Str[r], new Integer(Str[k]));
78          Data.PidAttrAndRanges.put(Str[r],new Integer(Str[k]));
79         }
```

```
80      }
81      else if(returnval.get(i).contains("DATA")&& !flag){
82       String[] Str =  returnval.get(i).split(",");
83       int j =0;
84       while(Str[j].compareTo("DATA") != 0){
85        Data.DataAttrAndRanges.put(Str[j],0);
86        j++;
87       }
88       j++;
89       for(int k = j,r = 0; k < Str.length;k++,r++){
90        maps.RangeMaps.put(Str[r], new Integer(Str[k]));
91        Data.DataAttrAndRanges.put(Str[r],new Integer(Str[k]));
92       }
93      }
94      else if (returnval.get(i).contains("FILE")&& !flag){
95       String[] Str =  returnval.get(i).split(",");
96          FileToSendTo = Str[1];
97      }
98      else if (returnval.get(i).contains("ATTR")&& !flag){
99       String[] Str =  returnval.get(i).split(",");
100           for(String s: Str)if(s.compareTo("ATTR") != 0) Data.AttrNames.add(s);
101     }
102     else if(returnval.get(i).contains("DISTRO")&& !flag){
103      String[] Str =  returnval.get(i).split(",");
104      for(String s: Str)
105       if(s.compareTo("ATTR") != 0)
106        if(s.contains("LVAL")){
107         lambda = (double)(new Double(s.split(":")[ s.split(":").length-1]));
108        }
109        else DistroToUse = s;
110     }
111     else if(returnval.get(i).contains("MAPS")&& !flag){
112      flag = true;
113      continue;
114     }
115     else if(flag){
116      String[] Str =  returnval.get(i).split(",");
117      maps.Mappings.put(Str[0], new HashMap<String,Integer>());
118      maps.InverseMappings.put(Str[0], new HashMap<Integer,String>());
119      for(int w = 0; w < Str.length-1; w++){
120       maps.Mappings.get(Str[0]).put(Str[w+1], w);
121       maps.InverseMappings.get(Str[0]).put(w,Str[w+1]);
122      }
123     }
124     else{}
125    }
126    DataBase = new DB(Data,maps);
```

```
127     for(int i = 0; i < numberofmembers; i++){
128      HashMap<String,String> ThisUID = new HashMap<String,String>();
129      for( String Attr : Data.UidAttrAndRanges.keySet()){
130       ThisUID.put(Attr,((Integer)DataBase.NumberofMembers).toString());
131      }
132       if(DistroToUse.equals("uniform"))DataBase.AddRandomMember(Data.AttrNames,
              ThisUID);
133       else if(DistroToUse.equals("gaussian"))DataBase.AddRandomGaussianMember(Data.
              AttrNames,ThisUID);
134       else if(DistroToUse.equals("poisson"))DataBase.AddRandomPoissonMember(Data.
              AttrNames,ThisUID,lambda);
135       else if(DistroToUse.equals("log-gaussian"))DataBase.AddRandomLogGaussianMember(
              Data.AttrNames,ThisUID);
136       else DataBase.AddRandomMember(Data.AttrNames,ThisUID);
137     }
138     for(HashMap<String,String> M: DataBase.MembersList.keySet()){
139      System.out.println(DataBase.MembersList.get(M).toString());
140     }
141     if(!DistroToUse.equals("poisson")){
142      if(todo) DataBase.WriteDataBaseToFile(outputlocation+"/"+FileToSendTo+"("+
             DistroToUse+")");
143      else DataBase.WriteDataBaseToFile(FileToSendTo+"("+DistroToUse+")");
144     }
145     else {
146      if(todo)DataBase.WriteDataBaseToFile(outputlocation+"/"+FileToSendTo+"("+
             DistroToUse+":"+lambda+")");
147      else DataBase.WriteDataBaseToFile(FileToSendTo+"("+DistroToUse+":"+lambda+")");
148     }
149     return DataBase;
150    }
151    //Head is just the line from the file at the moment and still needs to be parsed
152    //I think I might make a parsing function to do this for me however I have the
           problem
153    //That I need to return two values from the single function.
154    //I think I might just make a static subclass in this class to help with this
155    public static DB LoadDB(ArrayList<String> head, String FileToLoad) {
156         DB DataBase  = null;
157         MapAndData retval = ParseHead(head);
158      DataBase = new DB(retval.Data,retval.maps);
159      File file;
160      FileInputStream FIS = null;
161      DataInputStream DIS;
162      String header = new String();
163      char single;
164      try{
165           file= new File(FileToLoad+".csv");
166           try {
```

```
167       FIS = new FileInputStream(file);
168      } catch (FileNotFoundException e) {e.printStackTrace();}
169      DIS=new DataInputStream(FIS);
170        while(true){
171       single = (char)DIS.readByte();
172         if(single == '\n'){
173          String[] str = header.split(",");
174          HashMap<String,String> MAttrVal = new HashMap<String,String>();
175          HashMap<String,String> UID = new HashMap<String,String>();
176          for(String s: str){
177           String[] str2  = s.split(":");
178           if(DataBase.Data.UidAttrAndRanges.keySet().contains(str2[0])){
179            UID.put(str2[0], str2[1]);
180           }
181           else MAttrVal.put(str2[0], str2[1]);
182          }
183
184          DataBase.AddAMember(MAttrVal, UID);
185          header = new String();
186         }
187         else  header+=single;
188        }
189
190       }catch (EOFException e) {}
191       catch (IOException e) {e.printStackTrace();}
192   return DataBase;
193 }
194  private static MapAndData ParseHead(ArrayList<String> head) {
195   MapAndData retval = new MapAndData();
196   String[] Str  = null;
197      for(String S: head){
198       if(S.contains("NOA")== true){
199         Str =  S.split(",");
200        for(int i =1; i < Str.length;i++){
201         retval.Data.AttrNames.add(Str[i]);
202        }
203       }
204       else if(S.contains("NOM")== true){
205        Str =  S.split(",");
206        retval.NOM = new Integer(Str[1]);
207       }
208       else if(S.contains("RANGE")== true){
209        Str =  S.split(",");
210        for(int i =1; i < Str.length; i++){
211         String[] Str2 = Str[i].split(":");
212         retval.maps.RangeMaps.put(Str2[0], new Integer(Str2[1]));
213        }
```

```
214         }
215         else if(S.contains("MAPS")== true){
216          Str =  S.split(",");
217          boolean flagger = false;
218          String Attr = null;
219          HashMap<String,Integer> temp = new HashMap<String,Integer>();
220          HashMap<Integer,String> tempinverse = new HashMap<Integer,String>();
221          HashMap<String,Integer> Btemp = new HashMap<String,Integer>();
222          HashMap<Integer,String> Btempinverse = new HashMap<Integer,String>();
223          for(int i =1; i < Str.length; i++){
224           String[] Str2 = Str[i].split(":");
225           if(!flagger){
226              Attr = Str2[0];
227              flagger = true;
228           }
229           Btemp.put(new String(Str2[1]), new Integer(Str2[2]));
230           temp.put(Str2[1], new Integer(Str2[2]));
231           Btempinverse.put(new Integer(Str2[2]),new String(Str2[1]));
232           tempinverse.put(new Integer(Str2[2]),Str2[1]);
233          }
234          retval.maps.Mappings.put(Attr, temp);
235          retval.maps.InverseMappings.put(Attr, tempinverse);
236          retval.maps.BasicMappings.put(new String(Attr), Btemp);
237          retval.maps.BasicInverseMappings.put(new String(Attr), Btempinverse);
238         }
239         else if(S.contains("ALLOC")== true){
240          Str =  S.split(",");
241          for(int i =1; i < Str.length; i++){
242           String[] Str2 = Str[i].split(":");
243           if(Str2[0].compareTo("UID") == 0)
244            retval.Data.UidAttrAndRanges.put(Str2[1], retval.maps.RangeMaps.get(Str2
                  [1]));
245           else if(Str2[0].compareTo("PID") == 0)
246            retval.Data.PidAttrAndRanges.put(Str2[1], retval.maps.RangeMaps.get(Str2
                  [1]));
247           else if(Str2[0].compareTo("DATA") == 0)
248            retval.Data.DataAttrAndRanges.put(Str2[1], retval.maps.RangeMaps.get(Str2
                  [1]));
249           else System.out.println("Somthing went wrong, this means a attribute was
                 not allocated to a data type");
250          }
251         }
252        }
253   return retval;
254  }
255  public static ArrayList<String> ReadFromFile(String FileTORead){
256    File file;
```

```
257     FileInputStream FIS = null;
258     InputStreamReader IIS;
259     ArrayList<String> header = new  ArrayList<String>();
260     header.add(new String());
261     int i = 0;
262     char single;
263     int charcheck;
264     try{
265         file= new File(FileTORead);
266         try {
267       FIS = new FileInputStream(file);
268      } catch (FileNotFoundException e) {e.printStackTrace();}
269      IIS=new InputStreamReader(FIS,"UTF-8");
270        while(true){
271         charcheck = IIS.read();
272         if(charcheck == -1) throw new EOFException();
273         if(Character.toChars(charcheck).length != 1)
274          throw new IllegalArgumentException();
275         else single = Character.toChars(charcheck)[0];
276         if(single == '\n'){
277          header.add(new String());
278          i++;
279         }
280         else
281          header.set(i, header.get(i) + single);
282        }
283        }catch (EOFException e) {}
284        catch (IOException e) {e.printStackTrace();}
285        catch (IllegalArgumentException e){e.printStackTrace();}
286        return header;
287  }
288
289 }
```

B.5 Members.java

This is the class holding the members functionality.

```
1  package dataBase;
2  import java.util.ArrayList;
3  import java.util.HashMap;
4  import java.util.Random;
5
6  //These will be tuples in the database
7  public class Members {
8  //This version of making a member only works when we have one unique identifier
```

```
9  //which is the member identification number if we have more then one we would nedd
10 // a more complex structure
11     Members(int NumberOfAttributes,HashMap<String,String> MemberIdentification,
           DBData Data, Mapper maps){
12   AttributesAreSet = new HashMap<String,Boolean>();
13   NumberofAttr = Data.AttrNames.size();
14   Attributes = new HashMap<String,String>();
15   for( String uid : Data.UidAttrAndRanges.keySet()){
16    Attributes.put(uid,MemberIdentification.get(uid));
17    AttributesAreSet.put(uid, true);
18   }
19   for(String attr: Data.AttrNames){
20    if(Data.UidAttrAndRanges.keySet().contains(attr) == false){
21     AttributesAreSet.put(attr, false);
22    }
23   }
24  }
25     public Members(Members M){
26      Attributes = new  HashMap<String,String>();
27      AttributesAreSet = new HashMap<String,Boolean>();
28      NumberofAttr = M.GetNumberOfAttributes();
29      for(String keys : M.GetAllAttributes().keySet())
30       Attributes.put(new String(keys), new String(M.GetAllAttributes().get(keys)));
31      for(String keys : M.GetAllAttributesSetValues().keySet())
32       AttributesAreSet.put(new String(keys),true);
33     }
34 //The range is to be provided for a specific attribute
35  void AssignRandomAttribute(String attr,Random rand,DBData Data,Mapper maps){
36   if(AttributesAreSet.get(attr)== false){
37    if(Data.PidAttrAndRanges.keySet().contains(attr) == true){
38     double temp = ((int)Math.abs(rand.nextInt()))%(Data.PidAttrAndRanges.get(attr).
           intValue());
39     Attributes.put(attr, maps.InverseMappings.get(attr).get((int)(temp)));
40     AttributesAreSet.put(attr, true);
41    }
42    else if(Data.DataAttrAndRanges.keySet().contains(attr) == true) {
43     double temp = ((int)Math.abs(rand.nextInt()))%(Data.DataAttrAndRanges.get(attr)
           .intValue());
44     Attributes.put(attr, maps.InverseMappings.get(attr).get((int)(temp)));
45     AttributesAreSet.put(attr, true);
46    }
47   }
48  }
49  //The range is to be provided for a specific attribute
50  void AssignRandomGaussianAttribute(String attr,Random rand,DBData Data,Mapper maps
        ){
51   //we can assume that the size of the gaussian range is going to be 6
```

```
52    double nextval = rand.nextGaussian()+3;
53    double Increment_size = 0.0;
54    if(AttributesAreSet.get(attr)== false){
55     if(Data.PidAttrAndRanges.keySet().contains(attr) == true){
56      Increment_size = (6.0/((double)Data.PidAttrAndRanges.get(attr).intValue()));
57      if(nextval < 0.0) nextval = 0.0;
58      else if(nextval > 6.0) nextval = (double)Data.PidAttrAndRanges.get(attr).
            intValue()-1;
59      else nextval = Math.floor(nextval/Increment_size);
60     }
61     else if(Data.DataAttrAndRanges.keySet().contains(attr) == true){
62      Increment_size = (6.0/((double)Data.DataAttrAndRanges.get(attr).intValue()));
63      if(nextval < 0.0) nextval = 0.0;
64      else if(nextval > 6.0) nextval = (double)Data.DataAttrAndRanges.get(attr).
            intValue()-1;
65      else nextval = Math.floor(nextval/Increment_size);
66     }
67     Attributes.put(attr, maps.InverseMappings.get(attr).get((int)nextval));
68     AttributesAreSet.put(attr, true);
69    }
70   }
71   private void AssignRandomPoissonAttribute(String attr, Random rand,
72     DBData Data, Mapper maps, double lambda) {
73    int nextval = 0;
74    if(AttributesAreSet.get(attr)== false)
75     if(Data.PidAttrAndRanges.keySet().contains(attr) == true)
76      nextval = nextPoisson(rand,lambda,Data.PidAttrAndRanges.get(attr).intValue());
77     else if(Data.DataAttrAndRanges.keySet().contains(attr) == true)
78      nextval = nextPoisson(rand,lambda,Data.DataAttrAndRanges.get(attr).intValue());
79    Attributes.put(attr, maps.InverseMappings.get(attr).get((int)nextval));
80    AttributesAreSet.put(attr, true);
81   }
82   private void AssignRandomLogGaussianAttribute(String attr, Random rand,
83     DBData Data, Mapper maps) {
84    //we can assume that the size of the gaussian range is going to be 6
85    double nextval = nextLogGaussian(rand);
86    System.out.println("dsjklfjdsl "+ nextval);
87    double Increment_size = 0.0;
88    final double std = Math.sqrt(Math.E)*Math.sqrt(Math.E-1);
89    final double stdten = 10*std;
90    if(AttributesAreSet.get(attr)== false){
91     System.out.println("\t"+attr);
92     if(Data.PidAttrAndRanges.keySet().contains(attr) == true){
93      Increment_size = stdten/(((double)Data.PidAttrAndRanges.get(attr).intValue()))
            ;
94      if(nextval > stdten) nextval = (double)Data.PidAttrAndRanges.get(attr).
            intValue()-1;
```

```
95        else nextval = Math.floor(nextval/Increment_size);
96       }
97       else if(Data.DataAttrAndRanges.keySet().contains(attr) == true){
98        Increment_size = stdten/(((double)Data.DataAttrAndRanges.get(attr).intValue())
              );
99        if(nextval > stdten) nextval = (double)Data.DataAttrAndRanges.get(attr).
              intValue()-1;
100       else nextval = Math.floor(nextval/Increment_size);
101      }
102      System.out.println("dsjklfjdsl "+ nextval);
103     Attributes.put(attr, maps.InverseMappings.get(attr).get((int)nextval));
104     AttributesAreSet.put(attr, true);
105    }
106
107
108  }
109  double nextNextGaussian = 0.0;
110  boolean haveNextNextGaussian = false;
111  private double nextLogGaussian(Random rand) {
112    if (haveNextNextGaussian) {
113              haveNextNextGaussian = false;
114              return nextNextGaussian;
115      } else {
116              double v1, v2, s;
117              do {
118                      v1 = 2 * rand.nextDouble() - 1;   // between -1.0 and 1.0
119                      v2 = 2 * rand.nextDouble() - 1;   // between -1.0 and 1.0
120                      s = v1 * v1 + v2 * v2;
121              } while (s >= 1 || s == 0);
122              double multiplier = Math.exp(Math.sqrt(-2 * Math.log(s)/s));
123              nextNextGaussian = (v2+1) * multiplier;
124              haveNextNextGaussian = true;
125              return (v1+1) * multiplier;
126      }
127  }
128  private int nextPoisson(Random rand, double lambda,int setsize) {
129              double v1,sum = Poisson(0,lambda);
130              v1 = rand.nextDouble();
131              for(int i = 0;i < setsize ;i++)
132               if(v1 <= sum)return i;
133               else sum+= Poisson(i+1,lambda);
134              return setsize-1;
135      }
136  private double Poisson(int k, double lambda){
137   return (Math.pow(lambda, k)*Math.exp(-lambda))/factorial(k);
138  }
139   public static long factorial( int n )
```

```
140     {
141         if( n <= 1 )      // base case
142             return 1;
143         else
144             return n * factorial( n - 1 );
145     }
146 /**
147  *
148  * @param NewVal
149  * @param Assignment
150  * @param database
151  * @return
152  */
153 public boolean changeAnAttribute(String NewVal,String Assignment,DB database){
154  if(database.Maps.Mappings.get(Assignment).containsKey(NewVal)){
155   Attributes.put(Assignment, NewVal);
156   return true;
157  }
158  else return false;
159 }
160 /**
161  * This function is unsafe being that it changes the member without looking to see
        if the
162  * attribute is allowed for the members of the given database. This could lead to
       a loss of
163  * information when scanning the database.
164  * @param NewVal
165  * @param Assignment
166  * @param database
167  */
168 public void changeAnAttributeUnsafe(String NewVal,String Assignment){
169   Attributes.put(Assignment, NewVal);
170 }
171 void AssignGroupRandomAttribute(ArrayList<String> attr,Random rand,DBData Data,
      Mapper maps){
172   for( int i =0; i <attr.size(); i++)
173    if(!Data.UidAttrAndRanges.keySet().contains(attr.get(i)))
174     AssignRandomAttribute(attr.get(i),rand,Data,maps);
175
176 }
177 void AssignGroupRandomGaussianAttribute(ArrayList<String> attr,Random rand,DBData
      Data,Mapper maps){
178   for( int i =0; i <attr.size(); i++)
179    if(!Data.UidAttrAndRanges.keySet().contains(attr.get(i)))
180     AssignRandomGaussianAttribute(attr.get(i),rand,Data,maps);
181
182 }
```

```
183  public void AssignGroupRandomPoissonAttribute(ArrayList<String> attr,
184    Random rand, DBData Data, Mapper maps, double lambda) {
185   for( int i =0; i <attr.size(); i++)
186    if(!Data.UidAttrAndRanges.keySet().contains(attr.get(i)))
187     AssignRandomPoissonAttribute(attr.get(i),rand,Data,maps, lambda);
188
189  }
190
191  public void AssignGroupRandomLogGaussianAttribute(ArrayList<String> attr,
192    Random rand, DBData Data, Mapper maps) {
193   for( int i =0; i <attr.size(); i++)
194    if(!Data.UidAttrAndRanges.keySet().contains(attr.get(i))){
195     System.out.println(attr.get(i));
196     AssignRandomLogGaussianAttribute(attr.get(i),rand,Data,maps);
197
198    }
199  }
200  void AssignAttribute(String attr,Mapper maps, String value){
201   if(AttributesAreSet.get(attr) == false && maps.Mappings.get(attr).containsKey(
         value)){
202    Attributes.put(attr, value);
203    AttributesAreSet.put(attr, true);
204   }
205  }
206  void AssignGroupAttribute(HashMap<String,String> MAttrVal,Mapper maps){
207   if(NumberofAttr >= MAttrVal.size() ){
208    for(String Attr :MAttrVal.keySet())
209     AssignAttribute(Attr, maps,MAttrVal.get(Attr));
210   }
211  }
212
213  public String GetAttributeValue(String attr){
214   return (Attributes.get(attr));
215  }
216  public HashMap<String,String> GetAllAttributes(){
217   return Attributes;
218  }
219  public HashMap<String,Boolean> GetAllAttributesSetValues(){
220   return AttributesAreSet;
221  }
222  public int GetNumberOfAttributes(){
223   return NumberofAttr;
224  }
225  boolean IsComplete(){
226   boolean Returnvalue = true;
227   for(String sb: AttributesAreSet.keySet()){
228    Returnvalue &= AttributesAreSet.get(sb) ;
```

```
229     }
230     return Returnvalue;
231   }
232   @Override public String toString(){
233     String temp = new String();
234     boolean flag = false;
235     int i = 0;
236     for(String  Attr :Attributes.keySet()){
237       i++;
238       if(!flag){
239         temp+= '{';
240         flag = true;
241       }
242       if(i == Attributes.size()) temp+= Attr+":"+Attributes.get(Attr)+'}';
243       else temp+= Attr+":"+Attributes.get(Attr)+ ',';
244     }
245     return temp;
246   }
247   public HashMap<String,String> GetUID(DBData data) {
248     HashMap<String,String>  tempUID = new HashMap<String,String>();
249     for(String Attr : data.AttrNames)
250       if(data.UidAttrAndRanges.keySet().contains(Attr) == true)
251         tempUID.put(Attr, Attributes.get(Attr));
252     return tempUID;
253   }
254   public PseudoIdentity getPID(DBData Data) {
255     PseudoIdentity CurPid = null;
256     HashMap<String,String> theattr =  this.GetAllAttributes();
257     HashMap<String,String> thisPID = new HashMap<String,String>();
258     for(String pids :Data.PidAttrAndRanges.keySet()) thisPID.put(pids,theattr.get(
          pids));
259       CurPid = new PseudoIdentity(thisPID);
260     return CurPid;
261   }
262   private HashMap<String,String> Attributes;
263   private HashMap<String,Boolean> AttributesAreSet;
264   private int NumberofAttr;
265 }
```

B.6 PseudoIdentity.java

This is the class holding pseudo-identity functionality.

```
1  package dataBase;
2  import java.util.HashMap;
3
```

```
4
5  public class PseudoIdentity  {
6
7   public PseudoIdentity(){
8    PID = new HashMap<String,String>();
9    PIDSize = 0;
10  }
11  public PseudoIdentity(HashMap<String,String> PIDw){
12   PID = PIDw;
13   PIDSize = PID.keySet().size();
14  }
15  public  PseudoIdentity(PseudoIdentity PIDw){
16   PID = new HashMap<String,String>();
17   for(String a : PIDw.PID.keySet()){
18    PID.put(new String(a), new String (PIDw.PID.get(a)));
19   }
20   PIDSize = PID.keySet().size();
21  }
22  @Override public boolean equals(Object obj) {
23        if ( !(obj instanceof PseudoIdentity) ) return false;
24        PseudoIdentity that = (PseudoIdentity)obj;
25        for(String keys: PID.keySet() ){
26         if(this.PID.get(keys).compareTo(that.PID.get(keys)) != 0){
27          return false;
28         }
29        }
30    return true;
31  }
32  @Override public int hashCode(){
33   return this.PID.hashCode();
34  }
35  @Override public String toString(){
36   String temp = new String();
37   boolean flag = false;
38   for(String inte: PID.keySet()){
39    if(!flag){
40     temp+= PID.get(inte);
41     flag = true;
42    }
43    else temp+= ','+PID.get(inte);
44   }
45   return temp;
46  }
47  public String GetDim(int dim){
48   if(dim <= PIDSize) return PID.get(dim);
49   else return null;
50  }
```

```
51  //I believe this should be a Attribute and a value
52  public HashMap<String,String> PID;
53  public int PIDSize;
54 }
```

B.7 GenTree.java

This is the class holding Generalization tree functionality.

```
1  package dataBase;
2  import java.util.HashMap;
3
4
5  public class PseudoIdentity  {
6
7   public PseudoIdentity(){
8    PID = new HashMap<String,String>();
9    PIDSize = 0;
10  }
11  public PseudoIdentity(HashMap<String,String> PIDw){
12   PID = PIDw;
13   PIDSize = PID.keySet().size();
14  }
15  public  PseudoIdentity(PseudoIdentity PIDw){
16   PID = new HashMap<String,String>();
17   for(String a : PIDw.PID.keySet()){
18    PID.put(new String(a), new String (PIDw.PID.get(a)));
19   }
20   PIDSize = PID.keySet().size();
21  }
22  @Override public boolean equals(Object obj) {
23        if ( !(obj instanceof PseudoIdentity) ) return false;
24        PseudoIdentity that = (PseudoIdentity)obj;
25        for(String keys: PID.keySet() ){
26         if(this.PID.get(keys).compareTo(that.PID.get(keys)) != 0){
27          return false;
28         }
29        }
30    return true;
31  }
32  @Override public int hashCode(){
33   return this.PID.hashCode();
34  }
35  @Override public String toString(){
36   String temp = new String();
37   boolean flag = false;
```

```
38    for(String inte: PID.keySet()){
39     if(!flag){
40      temp+= PID.get(inte);
41      flag = true;
42     }
43     else temp+= ','+PID.get(inte);
44    }
45    return temp;
46   }
47   public String GetDim(int dim){
48    if(dim <= PIDSize) return PID.get(dim);
49    else return null;
50   }
51   //I believe this should be a Attribute and a value
52   public HashMap<String,String> PID;
53   public int PIDSize;
54  }
```

B.8 KAnonymitySupport.java

This is the class holding important function for K-anonymity.

```
1   package kAnonymity;
2
3   import genTree.GenTree;
4
5   import java.io.File;
6   import java.io.FileOutputStream;
7   import java.io.IOException;
8   import java.io.OutputStreamWriter;
9   import java.util.ArrayList;
10  import java.util.Arrays;
11  import java.util.HashMap;
12
13
14  import dataBase.DB;
15  import dataBase.Members;
16  import dataBase.PseudoIdentity;
17
18
19  public class KAnonymitySupport {
20   final static String csvheader = "\"Algorithm Type\",\"Distribution\",\"Metric\",\"
        DB Size\",\"Smallest Set\",\"Largest Set\",\"Average Set\",\"K\",\"Sets with K
        ";
21   final static String cvsheader2 = "\",\"Most Gen. value Attr:";
22   final  static String cvsheader3 = "\",\"Most Gen. value total Attr:";
```

```
23  final  static String cvsheader4 = "\",\"Most␣Gen.␣Attr\",";
24  final  static String cvsheader5 = "\"Most␣Gen.␣Attr␣%␣";
25  final  static String cvsheader6 = "\",\"Avg.␣Gen.␣Attr:␣";
26  final static String Forgotten = "\",\"Mem.␣with␣no␣Gen␣Attr:␣";
27  final  static String cvsheader7 = "\",\"Mem.␣with␣1␣Attr␣no␣gen.\",␣";
28  final  static String cvsheader8 = "\"Mem.␣with␣no␣gen.\",␣";
29  final  static String cvsheader9 = "\"Least␣gen.␣Mem.\",␣";
30  final  static String cvsheader10 = "\"Least␣gen.␣Mem.␣%\",␣";
31  final  static String cvsheader11 = "\"Most␣gen.␣Mem.\",␣";
32  final  static String cvsheader12 = "\"Most␣gen.␣Mem.␣%\",␣";
33  final  static String cvsheader13 = "\"Avg.␣gen.␣Mem.␣%␣";
34  final  static String cvsheader14 = "\",\"Sets␣of␣Size␣␣";
35  final  static String cvsheader15 = "\"";
36  public static SupportClassOne GetEQClasses(DB dataBase){
37    HashMap<PseudoIdentity, ArrayList<Members>> EQClasses = new  HashMap<
          PseudoIdentity, ArrayList<Members>>();
38   for(HashMap<String,String> keys : dataBase.MembersList.keySet()){
39    Members M = dataBase.MembersList.get(keys);
40    HashMap<String,String> theattr =  M.GetAllAttributes();
41    HashMap<String,String> thisPID = new HashMap<String,String>();
42    for(String pids :dataBase.Data.PidAttrAndRanges.keySet()){
43     thisPID.put(pids,theattr.get(pids));
44    }
45    PseudoIdentity PIDForHash = (new PseudoIdentity(thisPID));
46     if(EQClasses.get(PIDForHash) != null){
47     ArrayList<Members> temp = EQClasses.get(PIDForHash);
48     temp.add(M);
49          EQClasses.put(PIDForHash, temp);
50     }
51     else {
52      ArrayList<Members> temp = new ArrayList<Members>();
53      temp.add(M);
54      EQClasses.put(PIDForHash, temp);
55    }
56   }
57   int i =1;
58     int anonymity = 20000;
59     int strength = 0;
60     for(PseudoIdentity keys : EQClasses.keySet()){
61      if (anonymity> EQClasses.get(keys).size()){
62       anonymity = EQClasses.get(keys).size();
63       strength = 1;
64      }
65      else if(anonymity == EQClasses.get(keys).size()){
66       strength++;
67      }
68     i++;
```

```
69      }
70      SupportClassOne returnval = new SupportClassOne();
71      returnval.EQclasses = EQClasses;
72      returnval.CurK = anonymity;
73      returnval.Str = strength;
74      return returnval;
75    }
76  /**
77    * The discernibility metric of a given pseudo identity is essentially the number
          of times
78    * it shows up in the database. This happens to be saved in the hashmap. This
          metric was provided by
79    * Data privacy through optimal k-anonymization.
80    * @param EQClasses The size of each eqclass
81    * @param pid The pseudoidentity we are trying to find the metric for
82    * @return The metric value
83    */
84    public static int DiscernibilityMetric(HashMap<PseudoIdentity, Integer> EQClasses,
          PseudoIdentity pid ){
85     return (int)Math.pow(EQClasses.get(pid),2);
86    }
87    /**
88       *This needs to have the eq classes all the generalization trees.
89     * @param EQClasses The size of each eqclass
90     * @param pid The pseudoidentity we are trying to find the metric for
91     * @return The metric value
92     */
93    public static int DiscernibilityOverTuplesMetric(DB database,int k, HashMap<
          PseudoIdentity, ArrayList<Members>> EQClasses){
94     //we need to find for each member in the member set if the member is completely
          suppressed
95     //If so then we at add the size of the data base. if not then we add the size of
          the eq class it is
96     //part of.
97     int total = 0;
98     for(HashMap<String,String> a : database.MembersList.keySet()){
99      Members M = database.MembersList.get(a);
100     PseudoIdentity MPID = new PseudoIdentity(M.getPID(database.Data));
101     int count = 0;
102     for(String Attr : database.Data.PidAttrAndRanges.keySet()){
103      String attrval = M.GetAttributeValue(Attr);
104      if(database.GenTreesForAttr.get(Attr).isSuppressed(attrval))count++;
105     }
106     if(count == database.Data.PidAttrAndRanges.size())total+= database.MembersList.
          size();
107     else total += EQClasses.get(MPID).size();
108    }
```

```
109     return total;
110   }
111   /**
112     *This needs to have the eq classes all the generalization trees.
113    * @param EQClasses The size of each eqclass
114    * @param pid The pseudoidentity we are trying to find the metric for
115    * @return The metric value
116    */
117   public static int DiscernibilityOverAttributesMetric(DB database,int k, HashMap<
          PseudoIdentity, ArrayList<Members>> EQClasses,String Attr){
118    //Check for the given attribute if the members are completely surpressed.
119    // if so then return the size of the database else the size of the eqclass
120    int total = 0;
121    for(HashMap<String,String> a : database.MembersList.keySet()){
122     Members M = database.MembersList.get(a);
123     PseudoIdentity MPID = new PseudoIdentity(M.getPID(database.Data));
124     String attrval = M.GetAttributeValue(Attr);
125     if(database.GenTreesForAttr.get(Attr).isSuppressed(attrval))total+= database.
            MembersList.size();
126     else total += EQClasses.get(MPID).size();
127    }
128    return total;
129   }
130   public static double GCP(DB database,HashMap<PseudoIdentity, ArrayList<Members>>
          EQClasses){
131    double total = 0.0;
132    for(PseudoIdentity LM : EQClasses.keySet()){
133     total += EQClasses.get(LM).size()*NCP(database,LM);
134    }
135    total/= database.Data.PidAttrAndRanges.size()*database.NumberofMembers;
136    return total;
137   }
138   private static int NCP(DB database, PseudoIdentity LM) {
139    double total = 0.0;
140    for(String attr : database.Data.PidAttrAndRanges.keySet()){
141     if(LM.PID.get(attr).contains("|")){
142      double top = (double)(LM.PID.get(attr).split("|").length);
143      double bottom = (double)(database.GenTreesForAttr.get(attr).supression.split("|
             ").length);
144      total += top/bottom;
145     }
146     else total+= 1;
147    }
148    return 0;
149   }
150   public static double enthropy(HashMap<String, Integer> occur,int DBsize){
151    double InfoVal = 0.0;
```

```
152     for(String AttrVal : occur.keySet()){
153      double temp = ((double)occur.get(AttrVal))/((double)DBsize);
154       if(temp != 0.0)InfoVal += temp*(Math.log10(temp)/Math.log10(occur.keySet().size
              ()));
155       else InfoVal += temp;
156     }
157     InfoVal*=-1;
158     return InfoVal;
159   }
160   public static String MakeAttrStrings(String AtValOne, String AtValTwo , HashMap<
          String,Integer> maps, HashMap<Integer,String> inversemaps){
161     ArrayList<String> valsinAttr = new ArrayList<String>();
162     Integer[] retval = null;
163     //This gets the values contained in the two strings from the pid values
164     //This is necessary and would used the basic maps instead of the updated maps
165     for(String val : maps.keySet() ){
166       if(AtValOne.contains(val) && !valsinAttr.contains(val)) valsinAttr.add(val);
167       else if(AtValTwo.contains(val)&& !valsinAttr.contains(val))valsinAttr.add(val);
168     }
169     retval = new Integer[valsinAttr.size()];
170     //This changes the values retrieved from the first part to integers
171     //So that we can build the string used the inverse mapping
172     for(int i= 0 ; i < valsinAttr.size(); i++ ) retval[i] = maps.get(valsinAttr.get(i
              ));
173     //We sort the array because we want the values to be in order from smallest to
              greatest
174     Arrays.sort(retval);
175     //Now we build the string
176     String fin = "";
177     fin+='{';
178     boolean flag = false;
179     for(int i = 0; i <retval.length;i++ ){
180      if(!flag){
181       fin+=new String(inversemaps.get(retval[i]));
182       flag = true;
183      }
184      else {
185       fin+= '|';
186       fin+=new String(inversemaps.get(retval[i]));
187      }
188     }
189     fin+= '}';
190     return fin;
191   }
192   public static PseudoIdentity AncestorMetric(PseudoIdentity theChosenOne,
193      PseudoIdentity a, DB dataBase,int ChangeRequired) {
194     PseudoIdentity retval = new PseudoIdentity();
```

```
195   for(String Attr : dataBase.Data.PidAttrAndRanges.keySet()){
196    GenTree TreeTemp = dataBase.GenTreesForAttr.get(Attr);
197    String temp = TreeTemp.findAncestor(theChosenOne.PID.get(Attr), a.PID.get(Attr))
           ;
198    if(temp != null)ChangeRequired+=TreeTemp.maxSubTreesize(a.PID.get(Attr))+
           TreeTemp.maxSubTreesize(theChosenOne.PID.get(Attr))+TreeTemp.findheightTo(
           theChosenOne.PID.get(Attr), temp)+TreeTemp.findheightTo(a.PID.get(Attr),
           temp);
199    retval.PID.put(Attr, temp);
200    retval.PIDSize++;
201   }
202   return retval;
203  }
204  public static void MakeStatistics(DB dataBase,String filename,String outfilename,
         boolean dogenout) {
205   SupportClassOne info = KAnonymitySupport.GetEQClasses(dataBase);
206   int Smallest=Integer.MAX_VALUE, Largest=0,NumberSetsAtK =0;
207   int k = new Integer(filename.split("_")[1]).intValue();
208   String distribution = filename.split("_")[0];
209   String algor = filename.split("_")[2];
210   String Metric = filename.split("_")[3];
211   distribution =  distribution.split("[(]{1}")[2];
212   distribution= distribution.substring(0, distribution.length()-1);
213   //opening the file
214   File file;
215   FileOutputStream fos = null;
216   OutputStreamWriter oos = null;
217   String TheFilehead ="";
218      try{
219       file= new File(outfilename.concat(".csv"));
220       fos = new FileOutputStream(file,true);
221       oos=new OutputStreamWriter(fos,"UTF-8");
222
223      }catch (IOException e) { e.printStackTrace();}
224   ArrayList<String> theattr = new ArrayList<String>(dataBase.Data.PidAttrAndRanges.
         keySet());
225   String[] TA =theattr.toArray(new String[dataBase.Data.PidAttrAndRanges.size()]);
226   Arrays.sort(TA);
227       if(dogenout){
228        TheFilehead+=csvheader;
229        for(int i = 0; i < TA.length; i++){
230         String Attr = TA[i];
231         TheFilehead+=(cvsheader2+"_"+Attr);
232        }
233        for(int i = 0; i < TA.length; i++){
234         String Attr = TA[i];
235         TheFilehead+=(cvsheader3+"_"+Attr);
```

```
236         }
237         TheFilehead+=cvsheader4;
238         TheFilehead+=cvsheader5;
239         for(int i = 0; i < TA.length; i++){
240          String Attr = TA[i];
241          TheFilehead+=(cvsheader6+" "+Attr);
242         }
243         for(int i = 0; i < TA.length; i++){
244          String Attr = TA[i];
245          TheFilehead+=(Forgotten+" "+Attr);
246         }
247
248         TheFilehead+=cvsheader7;
249         TheFilehead+=cvsheader8;
250         TheFilehead+=cvsheader9;
251         TheFilehead+=cvsheader10;
252         TheFilehead+=cvsheader11;
253         TheFilehead+=new String(cvsheader12);
254         TheFilehead+= new String (cvsheader13);
255
256        }
257       String Size = "";
258    if(Metric.equals("Entro")) Metric = "Entropy";
259    if(Metric.equals("AnTree")) Metric = "Ancestor Tree";
260    if(Metric.equals("Dicern")) Metric = "Global Certainty Penalty";
261    if(filename.contains("Large")||filename.contains("large")) Size = "Large";
262    if(filename.contains("Small") || filename.contains("small")) Size = "Small";
263    if(filename.contains("Medium")|| filename.contains("medium")) Size = "Medium";
264    if(algor.equals("AnonymityLowKOnlyAndNoBreak")) algor = "Only choose smallest set
         as random pivot and does not break sets which are too big";
265    if(algor.equals("AnonymityAnySetAndNoBreak")) algor = "Any set can be the random
         pivot and does not break sets which are too big";
266    if(algor.equals("AnonymityLowKOnlyAndBreak")) algor = "Only choose smallest set
         as random pivot and breaks sets which are too big";
267    if(algor.equals("AnonymityAnySetAndBreak")) algor = "Any set can be the random
         pivot and breaks sets which are too big";
268    System.out.println("Algorithm is , "+algor);
269    if(!dogenout)TheFilehead+="\""+algor+"\""+",";
270    System.out.println("The database is was made with a "+distribution+" distribution
         ");
271    if(!dogenout)TheFilehead+="\""+distribution+"\""+",";
272    System.out.println("The metric used for K-anonymizing the database was "+Metric);
273    if(!dogenout)TheFilehead+="\""+Metric+"\""+",";
274    if(!dogenout)TheFilehead+="\""+Size+"\""+",";
275    HashMap<String,String> MostGeneralized = new HashMap<String,String>();
276    HashMap<String,Integer> MostGeneralizedAmount = new HashMap<String,Integer>();
277    HashMap<String,Integer>AveragePerAttr =  new HashMap<String,Integer>();
```

```
278  HashMap<String,Integer>NoGensPerAttr =  new HashMap<String,Integer>();
279  int numMemberwithatleastnogen = 0;
280  int Memberwithnogen = 0;
281  double average =0;
282
283  for(PseudoIdentity ML : info.EQclasses.keySet()){
284   if( info.EQclasses.get(ML).size()< Smallest) Smallest = info.EQclasses.get(ML).
         size();
285   if( info.EQclasses.get(ML).size() > Largest) Largest = info.EQclasses.get(ML).
         size();
286   if(info.EQclasses.get(ML).size() == k )NumberSetsAtK++;
287   average+=(double)info.EQclasses.get(ML).size();
288   boolean flag = true;
289   int Memberwithno = 0;
290   for(int i = 0; i < TA.length; i++){
291    String Attr = TA[i];
292    if(!ML.PID.get(Attr).contains("|")){
293     Memberwithno++;
294     if (flag == true){
295      numMemberwithatleastnogen+= info.EQclasses.get(ML).size();
296      flag = false;
297     }
298     if(!NoGensPerAttr.containsKey(Attr)){
299      NoGensPerAttr.put(Attr, info.EQclasses.get(ML).size());
300     }
301     else NoGensPerAttr.put(Attr, new Integer(NoGensPerAttr.get(Attr).intValue()+
           info.EQclasses.get(ML).size()));
302    }
303
304    if(!AveragePerAttr.containsKey(Attr))
305     AveragePerAttr.put(Attr,info.EQclasses.get(ML).size()*ML.PID.get(Attr).split("
           |").length);
306    else{
307     int com = info.EQclasses.get(ML).size()*ML.PID.get(Attr).split("|").length;
308     AveragePerAttr.put(Attr,new Integer(AveragePerAttr.get(Attr).intValue()+com));
309    }
310    if(!MostGeneralized.containsKey(Attr)){
311     MostGeneralized.put(Attr, ML.PID.get(Attr));
312    }else {
313     if(ML.PID.get(Attr).contains("|") && MostGeneralized.get(Attr).contains("|"))
314      if(MostGeneralized.get(Attr).split("|").length < ML.PID.get(Attr).split("|").
           length)
315       MostGeneralized.put(Attr, ML.PID.get(Attr));
316     else if (!ML.PID.get(Attr).contains("|") && MostGeneralized.get(Attr).contains
           ("|"))
317      if(MostGeneralized.get(Attr).split("|").length < 1)
318       MostGeneralized.put(Attr, ML.PID.get(Attr));
```

```
319         else if (ML.PID.get(Attr).contains("|") && !MostGeneralized.get(Attr).
                contains("|"))
320           MostGeneralized.put(Attr, ML.PID.get(Attr));
321       }
322      }
323      if (Memberwithno == ML.PID.size())Memberwithnogen++;
324     }
325     HashMap<PseudoIdentity,HashMap<String,Integer>> membersGeneraliztion =new
            HashMap<PseudoIdentity,HashMap<String,Integer>> ();
326     for(PseudoIdentity ML : info.EQclasses.keySet()){
327      membersGeneraliztion.put(ML, new HashMap<String,Integer>());
328      for(int i = 0; i < TA.length; i++){
329       String Attr = TA[i];
330       double temp = 0;
331       if(ML.PID.get(Attr).contains("|"))
332        temp = ((double)ML.PID.get(Attr).split("|").length)/((double)dataBase.
               GenTreesForAttr.get(Attr).supression.split("|").length);
333       else temp = 0;
334       int temp2 = (int)Math.floor(temp*100);
335       membersGeneraliztion.get(ML).put(Attr, temp2);
336      }
337
338     }
339     PseudoIdentity leastgen = null;
340     int leastgensize =Integer.MAX_VALUE;
341     PseudoIdentity Mostgen = null;
342     int Mostgensize =0;
343     int averagegensize = 0;
344     for(PseudoIdentity ML : info.EQclasses.keySet()){
345      int total =0;
346      for(int i = 0; i < TA.length; i++){
347       String Attr = TA[i];
348       total += membersGeneraliztion.get(ML).get(Attr).intValue();
349      }
350      total = (int)Math.floor((((double)total)/((double)dataBase.Data.PidAttrAndRanges
            .keySet().size())));
351      if(total <leastgensize ){
352       leastgensize = total;
353       leastgen = ML;
354      }
355      if(total > Mostgensize ){
356       Mostgensize = total;
357       Mostgen = ML;
358      }
359      averagegensize += total*info.EQclasses.get(ML).size();
360     }
361     averagegensize = (int)Math.floor((((double)averagegensize)/((double)dataBase.
```

```
        NumberofMembers)));
362     for(PseudoIdentity ML : info.EQclasses.keySet())
363      for(int i = 0; i < TA.length; i++){
364       String Attr = TA[i];
365       if(ML.PID.get(Attr).equals(MostGeneralized.get(Attr)))
366        if(MostGeneralizedAmount.keySet().contains(Attr)) MostGeneralizedAmount.put(
             Attr, new Integer(MostGeneralizedAmount.get(Attr)).intValue()+info.
             EQclasses.get(ML).size());
367        else MostGeneralizedAmount.put(Attr,info.EQclasses.get(ML).size());
368      }
369     String[] mostgen = new String[MostGeneralized.keySet().size()];
370     int mostgensize = 0;
371     int percent = 0;
372     for(int i = 0; i < TA.length; i++){
373      String Attr = TA[i];
374      double temp2 = MostGeneralized.get(Attr).split("|").length;
375      temp2 /=((double)dataBase.GenTreesForAttr.get(Attr).supression.split("|").length
           );
376      int temp = (int)Math.floor(temp2*100);
377      if(temp > percent ){
378       if(mostgensize > 0) mostgensize = 0;
379       mostgen[mostgensize] = Attr;
380       mostgensize++;
381       percent = temp;
382      }
383      else if(temp == percent){
384       mostgen[mostgensize] = Attr;
385       mostgensize++;
386      }
387     }
388     HashMap<Integer,Integer> NumberofEQclassesbysize = new HashMap<Integer,Integer>()
           ;
389     for(PseudoIdentity ML : info.EQclasses.keySet()){
390      if(NumberofEQclassesbysize.containsKey( info.EQclasses.get(ML).size())){
391       int temp = NumberofEQclassesbysize.get(info.EQclasses.get(ML).size());
392       NumberofEQclassesbysize.put(info.EQclasses.get(ML).size(), temp+1);
393      }
394      else NumberofEQclassesbysize.put(info.EQclasses.get(ML).size(),1);
395     }
396     ArrayList<Integer> theSizes = new ArrayList<Integer>(NumberofEQclassesbysize.
           keySet());
397     Integer[] TS =theSizes.toArray(new Integer[theSizes.size()]);
398     Arrays.sort(TS);
399     average/=(double)info.EQclasses.size();
400     System.out.println("Smallest Set: "+ Smallest);
401     if(!dogenout)TheFilehead+=Smallest+",";
402     System.out.println("Largest Set: "+ Largest );
```

```
403   if(!dogenout)TheFilehead+=Largest+",";
404   System.out.println("average Set: "+ average );
405   if(!dogenout)TheFilehead+=average+",";
406   System.out.println("Number of sets of size K("+k+"): "+ NumberSetsAtK );
407   if(!dogenout)TheFilehead+=k+",";
408   if(!dogenout)TheFilehead+=NumberSetsAtK+",";
409   for(int i = 0; i < TA.length; i++){
410    String Attr = TA[i];
411    System.out.println("The most generalized value used for Attribute "+Attr+" is "+
          MostGeneralized.get(Attr)+" and has a count of "+ MostGeneralizedAmount.get(
          Attr));
412    if(!dogenout)TheFilehead+="\""+MostGeneralized.get(Attr)+"\""+",";
413   }
414   for(int i = 0; i < TA.length; i++){
415    String Attr = TA[i];
416    if(!dogenout)TheFilehead+="\""+MostGeneralizedAmount.get(Attr)+"\""+",";
417   }
418   String output = "";
419   for(int i = 0; i < mostgensize; i++)output+= mostgen[i] + ",";
420   System.out.println("The most generalized Attribute(s) are "+output+" which are "
         + percent+"% generalized");
421   if(!dogenout)TheFilehead+="\""+output.substring(0, output.length()-1)+"\""+",";
422   if(!dogenout)TheFilehead+=percent+",";
423   for(int i = 0; i < TA.length; i++){
424    String Attr = TA[i];
425    double temp = (double)AveragePerAttr.get(Attr);
426    temp/=dataBase.NumberofMembers;
427    temp/= dataBase.GenTreesForAttr.get(Attr).supression.split("|").length;
428    int temp2 =(int)Math.floor(temp*100);
429    System.out.println("Attribute "+Attr+" has an average generalization percentage
          of "+ temp2+"%");
430    if(!dogenout) TheFilehead+=temp2+",";
431   }
432   for(int i = 0; i < TA.length; i++){
433    String Attr = TA[i];
434    System.out.println("Attribute "+Attr+" has " +NoGensPerAttr.get(Attr) + " values
          that have not been generalized");
435    if(!dogenout){
436     if(NoGensPerAttr.get(Attr) == null)TheFilehead+="0"+",";
437     else TheFilehead+=NoGensPerAttr.get(Attr)+",";
438    }
439   }
440   System.out.println("The number of members with at least one attribute that is not
          generalized is "+numMemberwithatleastnogen);
441   if(!dogenout)TheFilehead+=numMemberwithatleastnogen+",";
442   System.out.println("The number of members with no generalization is "+
          Memberwithnogen);
```

```
443     if(!dogenout)TheFilehead+=Memberwithnogen+",";
444     System.out.println("The least generalized Member over all the attributes is  "+
            leastgen.toString() + "  with an overall generalization of " +leastgensize+"%
            ");
445     if(!dogenout)TheFilehead+="\""+leastgen.toString()+"\""+",";
446     if(!dogenout)TheFilehead+=leastgensize+",";
447     System.out.println("The Most generalized Member over all the attributes is  "+
            Mostgen.toString() + "  with an overall generalization of"+ Mostgensize+"%")
            ;
448     if(!dogenout)TheFilehead+="\""+Mostgen.toString()+"\""+",";
449     if(!dogenout)TheFilehead+=Mostgensize+",";
450     System.out.println("The average generalization  over all the attributes is "+
            averagegensize+"%");
451     if(!dogenout)TheFilehead+=averagegensize;
452     for(Integer I : TS){
453      System.out.println("Number of sets of size ("+I+"): "+ NumberofEQclassesbysize.
             get(I) );
454     }
455     try{
456      if(dogenout) TheFilehead+=(cvsheader15);
457       oos.write((TheFilehead+"\n").toCharArray());
458      oos.close();
459     }catch (IOException e) {e.printStackTrace();}
460   }
461 }
```

B.9 SupportClassOne.java

This is the class used for eqclasses.

```
1  package kAnonymity;
2
3  import java.util.ArrayList;
4  import java.util.HashMap;
5
6  import dataBase.Members;
7  import dataBase.PseudoIdentity;
8
9  public class SupportClassOne {
10   SupportClassOne(){
11    EQclasses = null;
12    CurK = 0;
13    Str = 0;
14   }
15   public SupportClassOne(SupportClassOne info) {
16    EQclasses = new HashMap<PseudoIdentity, ArrayList<Members>>();
```

```
17    for(PseudoIdentity a : info.EQclasses.keySet()){
18     EQclasses.put(new PseudoIdentity(a), new ArrayList<Members>());
19     for(Members b : info.EQclasses.get(a) )
20      EQclasses.get(a).add(b);
21    }
22    CurK =info.CurK;
23    Str = info.Str;
24
25   }
26   public HashMap<PseudoIdentity, ArrayList<Members>> EQclasses;
27   public int CurK;
28   public int Str;
29  }
```

B.10 NoSplitLargeAndBoth.java

This is the class used for the algorithms that test the picked set constraint .

```
1   package kAnonymity.MyKAnonymity;
2   import java.util.ArrayList;
3   import java.util.Arrays;
4   import java.util.Date;
5   import java.util.HashMap;
6   import java.util.Random;
7
8   import kAnonymity.KAnonymitySupport;
9   import kAnonymity.SupportClassOne;
10
11  import dataBase.DB;
12  import dataBase.Members;
13  import dataBase.PseudoIdentity;
14  public class MyKAnonymityTwo {
15    static Random forpivots = new Random((new Date()).getTime());
16    public static DB MyKAnonymityAlgorithm(DB dataBase,int FinalK, int Metric,boolean
          whichchoice) {
17     int thechosen = -1;
18     PseudoIdentity TheChosenOne = null;
19     DB newdataBase = new DB(dataBase);
20     SupportClassOne info = KAnonymitySupport.GetEQClasses(newdataBase);
21     SupportClassOne info2 = new SupportClassOne(info);
22     while(info.CurK < FinalK){
23      if(thechosen != -1){
24       info = KAnonymitySupport.GetEQClasses(newdataBase);
25       info2 = new SupportClassOne(info);
26      }
27      if(whichchoice){
```

```
28        thechosen = Math.abs(forpivots.nextInt()) % info.EQclasses.keySet().size();
29        TheChosenOne = new PseudoIdentity((PseudoIdentity)info.EQclasses.keySet().
              toArray()[thechosen]);
30      }
31      else{
32       boolean flag = true;
33       while(flag){
34        thechosen = Math.abs(forpivots.nextInt()) % info.EQclasses.keySet().size();
35        TheChosenOne = new PseudoIdentity((PseudoIdentity)info.EQclasses.keySet().
              toArray()[thechosen]);
36        if(info.EQclasses.get(TheChosenOne).size()> info.CurK)continue;
37        else flag = false;
38       }
39      }
40      if(Metric == 0){
41       newdataBase.SetEntropy();
42        HashMap<PseudoIdentity,Double>  enthropies = null;
43       PseudoIdentity[] finals = new PseudoIdentity[info.EQclasses.keySet().size()
              -1];
44       enthropies = ComputeEntropies(newdataBase,info,TheChosenOne,finals);
45       if(finals.length != 0 )newdataBase = ImplementNewClustering(TheChosenOne,
              finals[0],newdataBase,info2);
46      }
47      else if(Metric == 1){
48       HashMap<PseudoIdentity, HashMap<PseudoIdentity,Integer>> ancestorCost = new
              HashMap<PseudoIdentity, HashMap<PseudoIdentity,Integer>>();
49       PseudoIdentity[] Listinsortedorder = new PseudoIdentity[info.EQclasses.keySet
              ().size()];
50       int position = 0;
51       for(PseudoIdentity a : info.EQclasses.keySet()){
52        if(!TheChosenOne.equals(a)){
53         int temp = 0;
54         PseudoIdentity tempPID = KAnonymitySupport.AncestorMetric(TheChosenOne,a,
              dataBase,temp);
55         ancestorCost.put(a, new HashMap<PseudoIdentity,Integer>());
56         ancestorCost.get(a).put(tempPID, temp);
57         Listinsortedorder[position] = a;
58         position++;
59         inlinesortofPseudobyentAncestor(Listinsortedorder,ancestorCost,position-1);
60
61        }
62       }
63
64       if( Listinsortedorder[0] != null){
65        PseudoIdentity finalchoice = new PseudoIdentity( Listinsortedorder[0]);
66        PseudoIdentity upgradefinal = new ArrayList<PseudoIdentity> (ancestorCost.get
              (Listinsortedorder[0]).keySet()).get(0);
```

```
67          int finalcost = ancestorCost.get(Listinsortedorder[0]).get(new ArrayList<
                PseudoIdentity> (ancestorCost.get(Listinsortedorder[0]).keySet()).get(0))
                ;
68         // System.out.println(TheChosenOne +"\n"+finalchoice+"\n"+upgradefinal);
69          newdataBase = ImplementNewClusteringForAncestor( TheChosenOne,  finalchoice,
                upgradefinal,newdataBase);
70         }
71        }
72        else if(Metric == 2){
73         PseudoIdentity[]  posretval = new PseudoIdentity[info.EQclasses.size()-1];
74         posretval = ComputeGCP(newdataBase,info,TheChosenOne);
75         if(posretval.length != 0 && posretval[0] != null){
76          PseudoIdentity tempPID = KAnonymitySupport.AncestorMetric(TheChosenOne,
                posretval[0], dataBase,0);
77          PseudoIdentity finalchoice = posretval[0];
78          PseudoIdentity upgradefinal = tempPID;
79          newdataBase = ImplementNewClusteringForAncestor( TheChosenOne,  finalchoice,
                upgradefinal,newdataBase);
80         }
81        }
82       }
83       return newdataBase;
84      }
85      private static PseudoIdentity[]  ComputeGCP(
86        DB newdataBase, SupportClassOne info, PseudoIdentity chosen) {
87       HashMap<PseudoIdentity, Double> retval = new HashMap<PseudoIdentity, Double>();
88       int position =0;
89       PseudoIdentity[] posretval = new PseudoIdentity[info.EQclasses.size()-1];
90       for(PseudoIdentity PID : info.EQclasses.keySet()){
91        if(!PID.equals(chosen)){
92         SupportClassOne tempinfo = new SupportClassOne(info);
93         PseudoIdentity tempPID = KAnonymitySupport.AncestorMetric(chosen,PID,
                newdataBase,0);
94         tempinfo.EQclasses.put(tempPID, new ArrayList<Members>(tempinfo.EQclasses.get(
                chosen)));
95         tempinfo.EQclasses.get(tempPID).addAll(tempinfo.EQclasses.get(PID));
96         tempinfo.EQclasses.remove(PID);
97         tempinfo.EQclasses.remove(chosen);
98         retval.put(PID, KAnonymitySupport.GCP(newdataBase, tempinfo.EQclasses));
99         posretval[position] = PID;
100        position++;
101        inlinesortofMembersbyentAncestor(posretval, retval,position -1);
102       }
103      }
104      return posretval;
105     }
106     private static void inlinesortofMembersbyentAncestor(
```

```
107     PseudoIdentity[] theLargeOrderedEnt, HashMap<PseudoIdentity, Double> ancestor,
108     int last) {
109    if(last != 0){
110     if(ancestor.get(theLargeOrderedEnt[last]) < ancestor.get(theLargeOrderedEnt[
            last-1])){
111      PseudoIdentity temp = theLargeOrderedEnt[last-1];
112      theLargeOrderedEnt[last-1] = theLargeOrderedEnt[last];
113      theLargeOrderedEnt[last] = temp;
114      inlinesortofMembersbyentAncestor(theLargeOrderedEnt,ancestor, last-1);
115     }
116    }
117   }
118   /**
119    * The return value from this function is a hash map where the key is the PID of
          a k-set and the
120    * second key is the PID of any cluster in the clusters. The value is the entropy
          when the two sets
121    * are combined.
122    * @param dataBase
123    * @param NeedToBeAno
124    * @param AllReadyHaveCurKAno
125    * @param info
126    * @return
127    */
128   public static HashMap<PseudoIdentity,Double>
129   ComputeEntropies(DB dataBase, SupportClassOne info,PseudoIdentity chosen,
          PseudoIdentity[] finals){
130    HashMap<PseudoIdentity,Double> enthropies = new HashMap<PseudoIdentity,Double>
          ();
131    int position = 0;
132    PseudoIdentity CurPid = null;
133    ArrayList<ArrayList<Members>> lister = new ArrayList<ArrayList<Members>>();
134    for(PseudoIdentity PID : info.EQclasses.keySet())
135     if(!PID.equals(chosen)) lister.add(info.EQclasses.get(PID));
136    for(ArrayList<Members> ListToBeKed : lister ){
137     CurPid = ListToBeKed.get(0).getPID(dataBase.Data);
138     ArrayList<Double> entrovalue = DiffCluster(ListToBeKed.size(),dataBase.
          MembersList.size(),chosen, CurPid,info,new HashMapCloner(dataBase.
          Occurances),dataBase);
139     double temppp = 0.0;
140     for(Double Ent :entrovalue) temppp+=Ent;
141     temppp/=entrovalue.size();
142     enthropies.put(CurPid, temppp);
143     finals[position] = CurPid;
144     position++;
145     inlinesortofMembersbyentAncestor(finals,enthropies,position-1);
146    }
```

```
147     return enthropies;
148   }
149
150   private static void inlinesortofPseudobyentAncestor(
151     PseudoIdentity[] theLargeOrderedEnt, HashMap<PseudoIdentity, HashMap<
          PseudoIdentity, Integer>> ancestor,
152     int last) {
153    if(last != 0){
154     if(ancestor.get(theLargeOrderedEnt[last]).get(new ArrayList<PseudoIdentity>(
          ancestor.get(theLargeOrderedEnt[last]).keySet()).get(0)) <
155       ancestor.get(theLargeOrderedEnt[last-1]).get(new ArrayList<PseudoIdentity>(
            ancestor.get(theLargeOrderedEnt[last-1]).keySet()).get(0)) ){
156      PseudoIdentity temp = theLargeOrderedEnt[last-1];
157      theLargeOrderedEnt[last-1] = theLargeOrderedEnt[last];
158      theLargeOrderedEnt[last] = temp;
159      inlinesortofPseudobyentAncestor(theLargeOrderedEnt,ancestor, last-1);
160     }
161    }
162   }
163   public static HashMap<PseudoIdentity,PseudoIdentity>
164   FindPivotsStarter(HashMap<PseudoIdentity, HashMap<PseudoIdentity,Double> >
        enthropies,PseudoIdentity TheChosen){
165    HashMap<PseudoIdentity,PseudoIdentity> pivots = new HashMap<PseudoIdentity,
         PseudoIdentity>();
166    for(PseudoIdentity k : enthropies.keySet()){
167     pivots.put(k, null);
168    }
169    PseudoIdentity[] A = new PseudoIdentity[enthropies.keySet().size()];
170    int i = 0;
171    for(PseudoIdentity a : enthropies.keySet()){
172     A[i] =a;
173     i++;
174    }
175    FindMaxPivots(A,enthropies,pivots,TheChosen);
176    return pivots;
177   }
178   public static double FindMaxPivots( PseudoIdentity[] identities,
179     HashMap<PseudoIdentity, HashMap<PseudoIdentity,Double> > enthropies,
180     HashMap<PseudoIdentity,PseudoIdentity> pivots,PseudoIdentity TheChosenID){
181     double cheapent = 0;
182     PseudoIdentity TheAnswer = null;
183     for(int position= 0; position < identities.length; position++){
184           if(enthropies.get(identities[position])!= null &&
185              enthropies.get(identities[position]).containsKey(TheChosenID) &&
186           enthropies.get(identities[position]).get(TheChosenID) >= cheapent &&
187           !identities[position].equals(TheChosenID)){
188         TheAnswer = new PseudoIdentity(identities[position]);
```

```
189         cheapent=enthropies.get(identities[position]).get(TheChosenID);
190        }
191        else if (enthropies.get(TheChosenID)!= null &&
192          enthropies.get(TheChosenID).containsKey(identities[position]) &&
193           enthropies.get(TheChosenID).get(identities[position]) >= cheapent){
194         TheAnswer = new PseudoIdentity(identities[position]);
195         cheapent=enthropies.get(TheChosenID).get(identities[position]) ;
196        }
197      }
198      for(PseudoIdentity a : pivots.keySet())pivots.put(a, a);
199      pivots.put(TheChosenID, TheAnswer);
200         return cheapent;
201   }
202   private static ArrayList<Double> DiffCluster(int CurPidSize,int DBSize,
          PseudoIdentity CurPid,
203   PseudoIdentity Others, SupportClassOne info, HashMapCloner cloner,DB database){
204    String NewValue = null;
205    HashMapCloner ThisOccurance = cloner;
206    ArrayList<Double> entrovalue = new ArrayList<Double>();
207    for(String Attr : CurPid.PID.keySet()){
208    // for(String val : ThisOccurance.MyMap.get(Attr).keySet() )
209    //  System.out.println("before      "+Attr+"    "+val+"      "+ ThisOccurance.MyMap
           .get(Attr).get(val));
210     if(CurPid.PID.get(Attr).compareTo(Others.PID.get(Attr)) != 0){
211      NewValue = KAnonymitySupport.MakeAttrStrings(CurPid.PID.get(Attr),Others.PID.
             get(Attr),database.Maps.BasicMappings.get(Attr),database.Maps.
             BasicInverseMappings.get(Attr));
212      //'{'+CurPid.PID.get(Attr)+'|'+Others.PID.get(Attr)+'}';
213   //Here we are putting the new attribute and new totals in  the occurance array
214      ThisOccurance.MyMap.get(Attr).put(NewValue, (CurPidSize+info.EQclasses.get(
             Others).size()));
215   //Here we are removing the old totals for each of the attributes
216      int temp = ThisOccurance.MyMap.get(Attr).get(CurPid.PID.get(Attr)).intValue();
217      temp -=CurPidSize;
218      ThisOccurance.MyMap.get(Attr).put(CurPid.PID.get(Attr), temp);
219      temp = ThisOccurance.MyMap.get(Attr).get(Others.PID.get(Attr)).intValue();
220      temp -= info.EQclasses.get(Others).size();
221      ThisOccurance.MyMap.get(Attr).put(Others.PID.get(Attr), temp);
222   //Here we are calculating the one dimensional entropy of the attribute we are
          currently working with
223     //  for(String val : ThisOccurance.MyMap.get(Attr).keySet() )
224      //  System.out.println("After      "+Attr+"    "+val+"      "+ ThisOccurance.MyMap
             .get(Attr).get(val));
225      entrovalue.add(KAnonymitySupport.enthropy(ThisOccurance.MyMap.get(Attr),
             DBSize));
226     }
227    }
```

```
228     return entrovalue;
229   }
230   public static DB ImplementNewClusteringForAncestor(PseudoIdentity TheChosen,
          PseudoIdentity TheOther,
231       PseudoIdentity TheUpgrade, DB database){
232       ArrayList<HashMap<String,String>> DBM = new ArrayList<HashMap<String,String>
              >(database.MembersList.keySet());
233       int totaladdition=0;
234       for(String Attr : database.Data.PidAttrAndRanges.keySet()){
235         if(!database.Maps.Mappings.get(Attr).containsKey(TheUpgrade.PID.get(Attr))){
236           Integer[] themappedtovalue = new Integer[database.Maps.Mappings.get(Attr).
                  size()];
237           themappedtovalue = database.Maps.Mappings.get(Attr).values().toArray(
                  themappedtovalue);
238           Arrays.sort(themappedtovalue);
239           int newintmap = themappedtovalue[themappedtovalue.length-1]+1;
240           database.Maps.Mappings.get(Attr).put(TheUpgrade.PID.get(Attr), newintmap);
241           database.Maps.InverseMappings.get(Attr).put(newintmap,TheUpgrade.PID.get(
                  Attr));
242           database.Maps.RangeMaps.put(Attr,newintmap+1);
243           database.Occurances.get(Attr).put(TheUpgrade.PID.get(Attr),totaladdition);
244           database.Data.PidAttrAndRanges.put(Attr, newintmap+1);
245         }
246       }
247       for(HashMap<String,String> M :DBM){
248         Members TM =database.MembersList.get(M);
249         PseudoIdentity TP = new PseudoIdentity(TM.getPID(database.Data));
250         if(TP.equals(TheChosen) || TP.equals(TheOther)){
251           for(String Attr : database.Data.PidAttrAndRanges.keySet()){
252             if(!TM.changeAnAttribute(TheUpgrade.PID.get(Attr), Attr, database)) System.
                    exit(0);
253             if(TP.equals(TheChosen) && !TheChosen.PID.get(Attr).equals(TheUpgrade.PID.
                    get(Attr))){
254               int t = database.Occurances.get(Attr).get(TheChosen.PID.get(Attr));
255               database.Occurances.get(Attr).put(TheChosen.PID.get(Attr),t--);
256             }
257             if(TP.equals(TheOther) && !TheOther.PID.get(Attr).equals(TheUpgrade.PID.get
                    (Attr))){
258               int t = database.Occurances.get(Attr).get(TheOther.PID.get(Attr));
259               database.Occurances.get(Attr).put(TheOther.PID.get(Attr),t--);
260             }
261           }
262           database.MembersList.put(M, TM);
263           totaladdition++;
264         }
265       }
266     return database;
```

```
267   }
268   public static DB ImplementNewClustering(HashMap<PseudoIdentity,PseudoIdentity>
          pivots,DB database){
269    HashMap<PseudoIdentity,PseudoIdentity> maptonewPID = new HashMap<PseudoIdentity,
           PseudoIdentity>();
270    for(PseudoIdentity keys : pivots.keySet()){
271     HashMap<String,String> temp = new HashMap<String,String>();
272     for(String Attr :keys.PID.keySet()){
273      if(pivots.get(keys) == null) return database;
274      else if(keys.PID.get(Attr).compareTo(pivots.get(keys).PID.get(Attr))==0){
275       temp.put(Attr, keys.PID.get(Attr));
276      }
277      else if(keys.PID.get(Attr).compareTo(pivots.get(keys).PID.get(Attr))!=0){
278       String newvalue = KAnonymitySupport.MakeAttrStrings(keys.PID.get(Attr),pivots
              .get(keys).PID.get(Attr),database.Maps.BasicMappings.get(Attr),database.
              Maps.BasicInverseMappings.get(Attr));
279       // new String('{'+keys.PID.get(Attr)+'|'+pivots.get(keys).PID.get(Attr)+'}');
280       String inverseNewValue = KAnonymitySupport.MakeAttrStrings(keys.PID.get(Attr)
              ,pivots.get(keys).PID.get(Attr),database.Maps.BasicMappings.get(Attr),
              database.Maps.BasicInverseMappings.get(Attr));
281       if(!database.Maps.Mappings.get(Attr).containsKey(newvalue)){
282        if(!database.Maps.Mappings.get(Attr).containsKey(inverseNewValue)){
283         Integer[] themappedtovalue = new Integer[database.Maps.Mappings.get(Attr).
               size()];
284         themappedtovalue = database.Maps.Mappings.get(Attr).values().toArray(
               themappedtovalue);
285         Arrays.sort(themappedtovalue);
286         int newintmap = themappedtovalue[themappedtovalue.length-1]+1;
287         database.Maps.Mappings.get(Attr).put(newvalue, newintmap);
288         database.Maps.InverseMappings.get(Attr).put(newintmap,newvalue);
289         database.Maps.RangeMaps.put(Attr,newintmap+1);
290         database.Occurances.get(Attr).put(newvalue, 0);
291         database.Data.PidAttrAndRanges.put(Attr, newintmap+1);
292         temp.put(Attr,newvalue);
293        }
294        else temp.put(Attr,inverseNewValue);
295       }
296       else temp.put(Attr,newvalue);
297      }
298     }
299     PseudoIdentity  temptotemp = new PseudoIdentity(temp);
300     maptonewPID.put(keys, temptotemp);
301    }
302   /* for(PseudoIdentity P : maptonewPID.keySet()){
303     System.out.println("this is for the new mapper "+P.toString()+"      "+
           maptonewPID.get(P).toString());
304    }*/
```

```
305     for(HashMap<String,String> UID : database.MembersList.keySet()){
306       PseudoIdentity MemPID = database.MembersList.get(UID).getPID(database.Data);
307       for(PseudoIdentity keys : pivots.keySet())
308        if(MemPID.equals(keys) || MemPID.equals(pivots.get(keys))){
309         for(String Attr : maptonewPID.get(keys).PID.keySet()){
310          if(database.MembersList.get(UID).GetAttributeValue(Attr).compareTo(
                 maptonewPID.get(keys).PID.get(Attr)) !=0){
311            Members m = database.MembersList.get(UID);
312            String tempooo = new String (m.GetAttributeValue(Attr));
313           m.changeAnAttribute(maptonewPID.get(keys).PID.get(Attr), Attr,database);
314            database.MembersList.put(UID, m);
315            int tempppp = database.Occurances.get(Attr).get(tempooo);
316            tempppp--;
317            database.Occurances.get(Attr).put(tempooo,tempppp);
318            tempppp =  database.Occurances.get(Attr).get(maptonewPID.get(keys).PID.
                   get(Attr));
319            tempppp++;
320            database.Occurances.get(Attr).put(maptonewPID.get(keys).PID.get(Attr),
                   tempppp);
321          }
322          else continue;
323         }
324        }
325     }
326     return database;
327   }
328   public static DB ImplementNewClustering(PseudoIdentity TC, PseudoIdentity TS, DB
        database, SupportClassOne info ){
329     PseudoIdentity newPID = new PseudoIdentity();
330     for(String Attr : database.Data.PidAttrAndRanges.keySet()){
331      String TCval = TC.PID.get(Attr);
332      String TSval = TS.PID.get(Attr);
333      if(TCval.equals(TSval)) newPID.PID.put(Attr, TCval);
334      else{
335       String newvalue = KAnonymitySupport.MakeAttrStrings(TCval,TSval,database.Maps.
            BasicMappings.get(Attr),database.Maps.BasicInverseMappings.get(Attr));
336       newPID.PID.put(Attr, newvalue);
337      }
338
339     }
340     return ImplementNewClusteringForAncestor(TC,TS,newPID,database);
341   }
342 }
```

B.11 SplitLargeAndAny.java

This is the class used for the algorithm that test split constraint and has no constraint on the picked set.

```
1  package kAnonymity.MyKAnonymity;
2  import java.util.ArrayList;
3  import java.util.Arrays;
4  import java.util.Date;
5  import java.util.HashMap;
6  import java.util.Random;
7
8  import kAnonymity.KAnonymitySupport;
9  import kAnonymity.SupportClassOne;
10 import dataBase.DB;
11 import dataBase.Members;
12 import dataBase.PseudoIdentity;
13
14
15 public class K_Split_Algorithm2 {
16  static Random forpivots = new Random((new Date()).getTime());
17  public static DB KAnonymityAlgorithm(DB dataBase, Integer FinalK, int Metric) {
18 //  first I want to take care of the distance function
19   SupportClassOne infoNeededForLonger = null;
20   int thechosen = -1;
21   PseudoIdentity TheChosenOne = null;
22   HashMap<PseudoIdentity,Double>  enthropies = null;
23   SupportClassOne info = null;
24   DB newdataBase = new DB(dataBase);
25   info = KAnonymitySupport.GetEQClasses(newdataBase);
26   int rand = 0;
27   while(info.CurK < FinalK){
28    if(thechosen != -1)info = KAnonymitySupport.GetEQClasses(newdataBase);
29    thechosen = Math.abs(forpivots.nextInt()) % info.EQclasses.keySet().size();
30    TheChosenOne = new PseudoIdentity((PseudoIdentity)info.EQclasses.keySet().
          toArray()[thechosen]);
31       infoNeededForLonger = new  SupportClassOne(info);
32    if(Metric == 0){
33     PseudoIdentity[] finals = new PseudoIdentity[info.EQclasses.keySet().size()-1];
34     enthropies = MyKAnonymityTwo.ComputeEntropies(newdataBase,info,TheChosenOne,
           finals);
35     PseudoIdentity pivot = TheChosenOne;
36     if(finals.length == 0 )continue;
37     if(infoNeededForLonger.EQclasses.get(pivot).size()+
38     infoNeededForLonger.EQclasses.get(finals[0]).size() < 2*FinalK){
39      newdataBase = MyKAnonymityTwo.ImplementNewClustering(TheChosenOne,finals[0],
           newdataBase,infoNeededForLonger);
```

```
40        infoNeededForLonger = null;
41       }
42       else{
43        if(infoNeededForLonger.EQclasses.get(pivot).size() >
44        infoNeededForLonger.EQclasses.get(finals[0]).size()){
45         Members[] ThechangedMembers = K_Split_Algorithm.FindTheValuesToTransfer(
               infoNeededForLonger.EQclasses.get(pivot),infoNeededForLonger.EQclasses.
               get(finals[0]),infoNeededForLonger, dataBase,finals[0]);
46         if(ThechangedMembers!=null){
47          for(Members a : ThechangedMembers) newdataBase.RemoveAMember(newdataBase.
                MembersList.get(a.GetUID(newdataBase.Data)));
48          for(Members a : ThechangedMembers) newdataBase.AddAMember(new Members(a));
49          for(Members a : ThechangedMembers) newdataBase = MyKAnonymityTwo.
                ImplementNewClustering(TheChosenOne,a.getPID(newdataBase.Data),
                newdataBase,infoNeededForLonger);
50         }
51        }
52        else if(infoNeededForLonger.EQclasses.get(pivot).size() <
53        infoNeededForLonger.EQclasses.get(finals[0]).size()){
54         Members[] ThechangedMembers = K_Split_Algorithm.FindTheValuesToTransfer(
               infoNeededForLonger.EQclasses.get(finals[0]),infoNeededForLonger.
               EQclasses.get(pivot),infoNeededForLonger, dataBase,pivot);
55         //There is a null point exception that happens once in awhile
56         if(ThechangedMembers != null){
57          for(Members a : ThechangedMembers) newdataBase.RemoveAMember(newdataBase.
                MembersList.get(a.GetUID(newdataBase.Data)));
58          for(Members a : ThechangedMembers) newdataBase.AddAMember(new Members(a));
59          for(Members a : ThechangedMembers) newdataBase = MyKAnonymityTwo.
                ImplementNewClustering(TheChosenOne,a.getPID(newdataBase.Data),
                newdataBase,infoNeededForLonger);
60         }
61        }
62       }
63      }
64      else if(Metric == 1){
65       HashMap<PseudoIdentity, HashMap<PseudoIdentity,Integer>> ancestorCost = new
             HashMap<PseudoIdentity, HashMap<PseudoIdentity,Integer>>();
66       PseudoIdentity[] Listinsortedorder = new PseudoIdentity[info.EQclasses.keySet()
             .size()-1];
67       int position = 0;
68       for(PseudoIdentity a : info.EQclasses.keySet()){
69        if(!TheChosenOne.equals(a)){
70         int temp = 0;
71         PseudoIdentity tempPID = KAnonymitySupport.AncestorMetric(TheChosenOne,a,
               dataBase,temp);
72         ancestorCost.put(a, new HashMap<PseudoIdentity,Integer>());
73         ancestorCost.get(a).put(tempPID, temp);
```

```
74          Listinsortedorder[position] = a;
75          position++;
76          inlinesortofPseudobyentAncestor(Listinsortedorder,ancestorCost,position-1);
77
78         }
79        }
80        if(Listinsortedorder.length == 0) continue;
81        PseudoIdentity finalchoice = new PseudoIdentity( Listinsortedorder[0]);
82        PseudoIdentity upgradefinal = new ArrayList<PseudoIdentity> (ancestorCost.get(
              Listinsortedorder[0]).keySet()).get(0);
83        int finalcost = ancestorCost.get(Listinsortedorder[0]).get(new ArrayList<
              PseudoIdentity> (ancestorCost.get(Listinsortedorder[0]).keySet()).get(0));
84        position = 1;
85        int smallestval = 0;
86        if(infoNeededForLonger.EQclasses.get(KAnonymitySupport.AncestorMetric(
              TheChosenOne,finalchoice, dataBase, 0))!= null)
87         smallestval =(int) (infoNeededForLonger.EQclasses.get(KAnonymitySupport.
               AncestorMetric(TheChosenOne,finalchoice, dataBase, 0)).size() +Math.ceil((
               infoNeededForLonger.EQclasses.get(TheChosenOne).size()+
88          infoNeededForLonger.EQclasses.get(finalchoice).size())/2 ));
89        int smallest = 0;
90        while(true){
91         if(position == Listinsortedorder.length ) break;;
92         PseudoIdentity Tempopo = KAnonymitySupport.AncestorMetric(TheChosenOne,
               finalchoice, dataBase, 0);
93         if(infoNeededForLonger.EQclasses.get(Tempopo) != null){
94          if(infoNeededForLonger.EQclasses.get(Tempopo).size()+Math.ceil((
                infoNeededForLonger.EQclasses.get(TheChosenOne).size()+
95            infoNeededForLonger.EQclasses.get(finalchoice).size())/2 ) >= 2*FinalK){
96           if(infoNeededForLonger.EQclasses.get(Tempopo).size()+Math.ceil((
                 infoNeededForLonger.EQclasses.get(TheChosenOne).size()+
97            infoNeededForLonger.EQclasses.get(finalchoice).size())/2 ) < smallestval){
98           smallest = position;
99           smallestval = (int) (infoNeededForLonger.EQclasses.get(Tempopo).size()+
                 Math.ceil((infoNeededForLonger.EQclasses.get(TheChosenOne).size()+
100            infoNeededForLonger.EQclasses.get(finalchoice).size())/2 ));
101          }
102          if(position >= Listinsortedorder.length /*||Listinsortedorder[position]==
                 null*/ ) System.out.println(Listinsortedorder.length+"  "+position);
103          finalchoice = new PseudoIdentity( Listinsortedorder[position]);
104           upgradefinal = new ArrayList<PseudoIdentity> (ancestorCost.get(
                  Listinsortedorder[position]).keySet()).get(0);
105          finalcost = ancestorCost.get(Listinsortedorder[position]).get(new ArrayList<
                 PseudoIdentity> (ancestorCost.get(Listinsortedorder[position]).keySet())
                 .get(0));
106          position++;
107          if(position >= Listinsortedorder.length -1 ){
```

```
108          finalchoice = new PseudoIdentity( Listinsortedorder[smallest]);
109           upgradefinal = new ArrayList<PseudoIdentity> (ancestorCost.get(
                  Listinsortedorder[smallest]).keySet()).get(0);
110          finalcost = ancestorCost.get(Listinsortedorder[smallest]).get(new ArrayList
                 <PseudoIdentity> (ancestorCost.get(Listinsortedorder[smallest]).keySet
                 ()).get(0));
111          break;
112         }
113        }
114        else break;
115       }
116       else break;
117      }
118
119      //System.out.println(TheChosenOne +"\n"+finalchoice+"\n"+upgradefinal);
120      if(infoNeededForLonger.EQclasses.get(TheChosenOne).size()+
121      infoNeededForLonger.EQclasses.get(finalchoice).size() < 2*FinalK){
122       newdataBase = ImplementNewClusteringForAncestor( TheChosenOne,  finalchoice,
              upgradefinal,newdataBase);
123      }
124      //we essentially need to get the old values from the database and find which
             would have
125      //fit best. This can be written tomorrow
126      else{
127       if(infoNeededForLonger.EQclasses.get(TheChosenOne).size() >
128       infoNeededForLonger.EQclasses.get(finalchoice).size()){
129        //This finds the members to completely move to the other class.
130        Members[] ThechangedMembers = FindTheValuesToTransferAncestor(
               infoNeededForLonger.EQclasses.get(TheChosenOne),infoNeededForLonger.
               EQclasses.get(finalchoice),infoNeededForLonger, dataBase,finalchoice);
131        if(ThechangedMembers!=null){
132         for(Members a : ThechangedMembers) newdataBase.RemoveAMember(newdataBase.
                MembersList.get(a.GetUID(newdataBase.Data)));
133         for(Members a : ThechangedMembers) newdataBase.AddAMember(new Members(a));
134         for(Members a : ThechangedMembers)
135          newdataBase = ImplementNewClusteringForAncestor(new PseudoIdentity( a.
                getPID(dataBase.Data)),  finalchoice, finalchoice,newdataBase);
136        }
137       }
138       else if(infoNeededForLonger.EQclasses.get(TheChosenOne).size() <
139       infoNeededForLonger.EQclasses.get(finalchoice).size()){
140        Members[] ThechangedMembers = FindTheValuesToTransferAncestor(
               infoNeededForLonger.EQclasses.get(finalchoice),infoNeededForLonger.
               EQclasses.get(TheChosenOne),infoNeededForLonger, dataBase,TheChosenOne);
141        if(ThechangedMembers != null){
142         for(Members a : ThechangedMembers) newdataBase.RemoveAMember(newdataBase.
                MembersList.get(a.GetUID(newdataBase.Data)));
```

```
143         for(Members a : ThechangedMembers) newdataBase.AddAMember(new Members(a));
144         for(Members a : ThechangedMembers)
145          newdataBase = ImplementNewClusteringForAncestor( new PseudoIdentity( a.
                 getPID(dataBase.Data)),  TheChosenOne, TheChosenOne,newdataBase);
146        }
147       }
148      }
149     }
150     else if( Metric == 2){
151      HashMap<PseudoIdentity, HashMap<PseudoIdentity,Integer>>  ancestorCost = new
             HashMap<PseudoIdentity, HashMap<PseudoIdentity,Integer>>();
152      PseudoIdentity[] Listinsortedorder = new PseudoIdentity[info.EQclasses.keySet()
             .size()-1];
153      Listinsortedorder = ComputeGCP(newdataBase,info,TheChosenOne);
154      if(Listinsortedorder.length == 0)continue;
155      PseudoIdentity finalchoice = new PseudoIdentity( Listinsortedorder[0]);
156      PseudoIdentity upgradefinal = KAnonymitySupport.AncestorMetric(TheChosenOne,
             Listinsortedorder[0], dataBase,0);
157      int position = 1;
158      int smallestval = 0;
159      if(infoNeededForLonger.EQclasses.get(KAnonymitySupport.AncestorMetric(
             TheChosenOne,finalchoice, dataBase, 0))!= null)
160       smallestval =(int) (infoNeededForLonger.EQclasses.get(KAnonymitySupport.
              AncestorMetric(TheChosenOne,finalchoice, dataBase, 0)).size() +Math.ceil((
              infoNeededForLonger.EQclasses.get(TheChosenOne).size()+
161        infoNeededForLonger.EQclasses.get(finalchoice).size())/2 ));
162      int smallest = 0;
163      while(true){
164       if(position ==Listinsortedorder.length ) break;
165       PseudoIdentity Tempopo = KAnonymitySupport.AncestorMetric(TheChosenOne,
              finalchoice, dataBase, 0);
166       if(infoNeededForLonger.EQclasses.get(Tempopo) != null){
167        if(infoNeededForLonger.EQclasses.get(Tempopo).size()+Math.ceil((
               infoNeededForLonger.EQclasses.get(TheChosenOne).size()+
168          infoNeededForLonger.EQclasses.get(finalchoice).size())/2 ) >= 2*FinalK){
169         if(infoNeededForLonger.EQclasses.get(Tempopo).size()+Math.ceil((
                infoNeededForLonger.EQclasses.get(TheChosenOne).size()+
170           infoNeededForLonger.EQclasses.get(finalchoice).size())/2 ) < smallestval){
171          smallest = position;
172          smallestval = (int) (infoNeededForLonger.EQclasses.get(Tempopo).size()+
                Math.ceil((infoNeededForLonger.EQclasses.get(TheChosenOne).size()+
173            infoNeededForLonger.EQclasses.get(finalchoice).size())/2 ));
174         }
175         finalchoice = new PseudoIdentity( Listinsortedorder[position]);
176          upgradefinal = KAnonymitySupport.AncestorMetric(TheChosenOne,
                Listinsortedorder[position], dataBase, 0);
177         position++;
```

```
178         if(position >= Listinsortedorder.length -1 ){
179          finalchoice = new PseudoIdentity( Listinsortedorder[smallest]);
180           upgradefinal = KAnonymitySupport.AncestorMetric(TheChosenOne,
                  Listinsortedorder[position-1], dataBase, 0);
181          break;
182         }
183        }
184        else break;
185       }
186       else break;
187      }
188      //System.out.println(TheChosenOne +"\n"+finalchoice+"\n"+upgradefinal);
189      if(infoNeededForLonger.EQclasses.get(TheChosenOne).size()+
190      infoNeededForLonger.EQclasses.get(finalchoice).size() < 2*FinalK /*&& flaggy*/)
             {
191       newdataBase = ImplementNewClusteringForAncestor( TheChosenOne,  finalchoice,
             upgradefinal,newdataBase);
192      }
193      //we essentially need to get the old values from the database and find which
             would have
194      //fit best. This can be written tomorrow
195      else{
196       if(infoNeededForLonger.EQclasses.get(TheChosenOne).size() >
197       infoNeededForLonger.EQclasses.get(finalchoice).size() /*&& !flaggy*/){
198        //This finds the members to completely move to the other class.
199        Members[] ThechangedMembers = FindTheValuesToTransferAncestor(
               infoNeededForLonger.EQclasses.get(TheChosenOne),infoNeededForLonger.
               EQclasses.get(finalchoice),infoNeededForLonger, dataBase,finalchoice);
200        if(ThechangedMembers!=null){
201         for(Members a : ThechangedMembers) newdataBase.RemoveAMember(newdataBase.
                MembersList.get(a.GetUID(newdataBase.Data)));
202         for(Members a : ThechangedMembers) newdataBase.AddAMember(new Members(a));
203         for(Members a : ThechangedMembers)
204          newdataBase = ImplementNewClusteringForAncestor(new PseudoIdentity( a.
                getPID(dataBase.Data)),  finalchoice, finalchoice,newdataBase);
205         info = KAnonymitySupport.GetEQClasses(newdataBase);
206        }
207       }
208       else if(infoNeededForLonger.EQclasses.get(TheChosenOne).size() <
209       infoNeededForLonger.EQclasses.get(finalchoice).size() /*&& !flaggy*/){
210        Members[] ThechangedMembers = FindTheValuesToTransferAncestor(
               infoNeededForLonger.EQclasses.get(finalchoice),infoNeededForLonger.
               EQclasses.get(TheChosenOne),infoNeededForLonger, dataBase,TheChosenOne);
211        if(ThechangedMembers != null){
212         for(Members a : ThechangedMembers) newdataBase.RemoveAMember(newdataBase.
                MembersList.get(a.GetUID(newdataBase.Data)));
213         for(Members a : ThechangedMembers) newdataBase.AddAMember(new Members(a));
```

```
214         for(Members a : ThechangedMembers)
215          newdataBase = ImplementNewClusteringForAncestor( new PseudoIdentity( a.
                 getPID(dataBase.Data)), TheChosenOne, TheChosenOne,newdataBase);
216         info = KAnonymitySupport.GetEQClasses(newdataBase);
217        }
218       }
219      }
220
221    }
222    infoNeededForLonger = null;
223   }
224   return newdataBase;
225  }
226  private static PseudoIdentity[]  ComputeGCP(
227    DB newdataBase, SupportClassOne info, PseudoIdentity chosen) {
228   HashMap<PseudoIdentity, Double> retval = new HashMap<PseudoIdentity, Double>();
229   int position =0;
230   PseudoIdentity[] posretval = new PseudoIdentity[info.EQclasses.size()-1];
231   for(PseudoIdentity PID : info.EQclasses.keySet()){
232    if(!PID.equals(chosen)){
233     SupportClassOne tempinfo = new SupportClassOne(info);
234     PseudoIdentity tempPID = KAnonymitySupport.AncestorMetric(chosen,PID,
             newdataBase,0);
235     tempinfo.EQclasses.put(tempPID, new ArrayList<Members>(tempinfo.EQclasses.get(
             chosen)));
236     tempinfo.EQclasses.get(tempPID).addAll(tempinfo.EQclasses.get(PID));
237     tempinfo.EQclasses.remove(PID);
238     tempinfo.EQclasses.remove(chosen);
239     retval.put(PID, KAnonymitySupport.GCP(newdataBase, tempinfo.EQclasses));
240     posretval[position] = PID;
241     position++;
242     inlinesortofMembersbyentAncestor(posretval, retval,position -1);
243    }
244   }
245   return posretval;
246  }
247  private static void inlinesortofMembersbyentAncestor(
248    PseudoIdentity[] theLargeOrderedEnt, HashMap<PseudoIdentity, Double> ancestor,
249    int last) {
250   if(last != 0){
251    if(ancestor.get(theLargeOrderedEnt[last]) < ancestor.get(theLargeOrderedEnt[last
           -1])){
252     PseudoIdentity temp = theLargeOrderedEnt[last-1];
253     theLargeOrderedEnt[last-1] = theLargeOrderedEnt[last];
254     theLargeOrderedEnt[last] = temp;
255     inlinesortofMembersbyentAncestor(theLargeOrderedEnt,ancestor, last-1);
256    }
```

```
257     }
258   }
259   private static Members[] FindTheValuesToTransfer(ArrayList<Members> large,
260     ArrayList<Members> small, SupportClassOne info,DB dataBase,
261     HashMap<PseudoIdentity,PseudoIdentity> temporary, PseudoIdentity pivot) {
262     //We want to get the member from the original database here
263     ArrayList<Members> TheLargeSetMembers = new ArrayList<Members>();
264     ArrayList<Members> TheSmallSetMembers = new ArrayList<Members>();
265     //calculating the number of values we should have in the larger set
266     int SizeOfLarge = (int)Math.ceil(((double)large.size()+
267       (double)small.size())/2);
268     int valuesToMove =large.size()-SizeOfLarge;
269     //if no change can be done
270     if(valuesToMove == 0) return null;
271     //gets the smaller sets members
272     for(Members meq : small)TheSmallSetMembers.add(dataBase.MembersList.get(meq.
          GetUID((dataBase.Data))));
273     //gets the larger sets members
274     for(Members meq : large) TheLargeSetMembers.add(dataBase.MembersList.get(meq.
          GetUID((dataBase.Data))));
275     //Make the new hashtable to compute the entropy
276     HashMap<String, HashMap<String,Integer>> TempOccurances = new HashMap<String,
          HashMap<String,Integer>>();
277     FillNewHashWithOld(dataBase,small,pivot,TempOccurances, dataBase.Occurances);
278     HashMap<Members,Double> Entrophy = new HashMap<Members,Double>();
279     Members[] TheLargeOrderedEnt = new Members[large.size()];
280     int position = 0;
281     for(int i =0 ; i < TheLargeOrderedEnt.length; i++) TheLargeOrderedEnt[i] = null;
282     for(Members mlg : TheLargeSetMembers){
283       TempOccurances = MakeOccuranceTable(dataBase,TempOccurances,mlg,pivot);
284       double tempent = 0.0;
285       for(String Attr : TempOccurances.keySet())
286         tempent += KAnonymitySupport.enthropy(TempOccurances.get(Attr),dataBase.
              MembersList.size());
287       tempent/= TempOccurances.size();
288       Entrophy.put(mlg,tempent);
289       TempOccurances = revertOccuranceTable(dataBase,TempOccurances,mlg,pivot);
290       TheLargeOrderedEnt[position] = mlg;
291       position++;
292       inlinesortofMembersbyent(TheLargeOrderedEnt,Entrophy,position-1);
293     }
294     Members[] retval= new Members[valuesToMove];
295     for(int i = 0; i < valuesToMove; i++) retval[i] = TheLargeOrderedEnt[i];
296     return retval;
297   }
298   private static Members[] FindTheValuesToTransferAncestor(ArrayList<Members> large,
299     ArrayList<Members> small, SupportClassOne info,DB dataBase, PseudoIdentity pivot
```

```
        ) {
300     //We want to get the member from the original database here
301     ArrayList<Members> TheLargeSetMembers = new ArrayList<Members>();
302     ArrayList<Members> TheSmallSetMembers = new ArrayList<Members>();
303     //calculating the number of values we should have in the larger set
304     int SizeOfLarge = (int)Math.ceil(((double)large.size()+
305       (double)small.size())/2);
306     int valuesToMove =large.size()-SizeOfLarge;
307     //if no change can be done
308     if(valuesToMove == 0) return null;
309     //gets the smaller sets members
310     for(Members meq : small)TheSmallSetMembers.add(dataBase.MembersList.get(meq.
            GetUID((dataBase.Data))));
311     //gets the larger sets members
312     for(Members meq : large) TheLargeSetMembers.add(dataBase.MembersList.get(meq.
            GetUID((dataBase.Data))));
313     //Make the new hashtable to compute the entropy
314     HashMap<String, HashMap<String,Integer>> TempOccurances = new HashMap<String,
            HashMap<String,Integer>>();
315     FillNewHashWithOld(dataBase,small,pivot,TempOccurances, dataBase.Occurances);
316     HashMap<Members,Integer> Ancestor = new HashMap<Members,Integer>();//This will
            change TODO
317     Members[] TheLargeOrderedEnt = new Members[large.size()];
318     int position = 0;
319     for(int i =0 ; i < TheLargeOrderedEnt.length; i++) TheLargeOrderedEnt[i] = null;
320     for(Members mlg : TheLargeSetMembers){
321      TempOccurances = MakeOccuranceTable(dataBase,TempOccurances,mlg,pivot);
322      int tempent = 0;
323      KAnonymitySupport.AncestorMetric(new PseudoIdentity(mlg.getPID(dataBase.Data)),
            pivot, dataBase, tempent);
324      Ancestor.put(mlg,tempent);
325      TempOccurances = revertOccuranceTable(dataBase,TempOccurances,mlg,pivot);
326      TheLargeOrderedEnt[position] = mlg;
327      position++;
328      inlinesortofMembersbyentAncestor(TheLargeOrderedEnt,Ancestor,position-1);
329     }
330     Members[] retval= new Members[valuesToMove];
331     for(int i = 0; i < valuesToMove; i++) retval[i] = TheLargeOrderedEnt[i];
332     return retval;
333   }
334   private static void inlinesortofPseudobyentAncestor(
335     PseudoIdentity[] theLargeOrderedEnt, HashMap<PseudoIdentity, HashMap<
            PseudoIdentity, Integer>> ancestor,
336     int last) {
337    if(last != 0){
338     if(ancestor.get(theLargeOrderedEnt[last]).get(new ArrayList<PseudoIdentity>(
            ancestor.get(theLargeOrderedEnt[last]).keySet()).get(0)) <
```

```
339       ancestor.get(theLargeOrderedEnt[last-1]).get(new ArrayList<PseudoIdentity>(
              ancestor.get(theLargeOrderedEnt[last-1]).keySet()).get(0)) ){
340      PseudoIdentity temp = theLargeOrderedEnt[last-1];
341      theLargeOrderedEnt[last-1] = theLargeOrderedEnt[last];
342      theLargeOrderedEnt[last] = temp;
343      inlinesortofPseudobyentAncestor(theLargeOrderedEnt,ancestor, last-1);
344     }
345    }
346   }
347   private static void inlinesortofMembersbyentAncestor(
348     Members[] theLargeOrderedEnt, HashMap<Members, Integer> ancestor,
349     int last) {
350    if(last != 0){
351     if(ancestor.get(theLargeOrderedEnt[last]) < ancestor.get(theLargeOrderedEnt[last
          -1])){
352      Members temp = theLargeOrderedEnt[last-1];
353      theLargeOrderedEnt[last-1] = theLargeOrderedEnt[last];
354      theLargeOrderedEnt[last] = temp;
355      inlinesortofMembersbyentAncestor(theLargeOrderedEnt,ancestor, last-1);
356     }
357    }
358   }
359   private static void inlinesortofMembersbyent(Members[] theLargeOrderedEnt,
360     HashMap<Members, Double> entrophy, int last) {
361     if(last != 0){
362      if(entrophy.get(theLargeOrderedEnt[last]) < entrophy.get(theLargeOrderedEnt[
           last-1])){
363       Members temp = theLargeOrderedEnt[last-1];
364       theLargeOrderedEnt[last-1] = theLargeOrderedEnt[last];
365       theLargeOrderedEnt[last] = temp;
366       inlinesortofMembersbyent(theLargeOrderedEnt,entrophy, last-1);
367      }
368     }
369   }
370   //put a member back into the occurances when it is removed
371   private static HashMap<String, HashMap<String, Integer>> revertOccuranceTable(
372     DB database, HashMap<String, HashMap<String, Integer>> tempOccurances,
373     Members mlg, PseudoIdentity pivot) {
374    for(String Attr : tempOccurances.keySet()){
375     ArrayList<String> valremovalList = new ArrayList<String>();
376     if(database.Data.PidAttrAndRanges.containsKey(Attr)){
377      for(String val : tempOccurances.get(Attr).keySet() ){
378       if(val.equals(pivot.PID.get(Attr))){
379        int oval =  tempOccurances.get(Attr).get(pivot.PID.get(Attr));
380        oval--;
381        if(oval == 0)valremovalList.add(pivot.PID.get(Attr));
382        else tempOccurances.get(Attr).put(pivot.PID.get(Attr),oval);
```

```
383         }
384        }
385        for(String a : valremovalList) tempOccurances.get(Attr).remove(a);
386        if(tempOccurances.get(Attr).keySet().contains(mlg.GetAttributeValue(Attr))){
387         int oval = tempOccurances.get(Attr).get(mlg.GetAttributeValue(Attr));
388         oval++;
389         tempOccurances.get(Attr).put(mlg.GetAttributeValue(Attr),oval);
390        }
391        else tempOccurances.get(Attr).put(mlg.GetAttributeValue(Attr),1);
392       }
393      }
394      for(String Attr : tempOccurances.keySet()){
395       if(database.Data.PidAttrAndRanges.containsKey(Attr)){
396        if(tempOccurances.get(Attr).containsKey(mlg.GetAttributeValue(Attr))){
397         int oval =  tempOccurances.get(Attr).get(mlg.GetAttributeValue(Attr));
398         oval++;
399         tempOccurances.get(Attr).put(mlg.GetAttributeValue(Attr),oval);
400        }
401        else tempOccurances.get(Attr).put(mlg.GetAttributeValue(Attr),1);
402       }
403      }
404      return tempOccurances;
405     }
406     //This function will copy the hashmap from the database which holds all the
           occurances
407     //in the database.
408     private static void FillNewHashWithOld(DB database, ArrayList<Members> SmallSet,
           PseudoIdentity pivot,
409       HashMap<String, HashMap<String, Integer>> tempOccurances,
410       HashMap<String, HashMap<String, Integer>> occurances) {
411       for(String Attr : occurances.keySet()){
412        tempOccurances.put(Attr, new HashMap<String, Integer>());
413        for(String val : occurances.get(Attr).keySet())
414         tempOccurances.get(Attr).put(new String(val),new Integer(occurances.get(Attr).
               get(val).intValue()));
415       }
416       //This will go through the occurences hashmap and remove all occurances of the
             members in the
417       //smaller set. They will be reset with the values in the pivot pseudoidentity.
418       for(String Attr : tempOccurances.keySet()){
419        ArrayList<String> valremovalList = new ArrayList<String>();
420        if(database.Data.PidAttrAndRanges.containsKey(Attr)){
421         for(String val : tempOccurances.get(Attr).keySet() ){
422          for(Members sm: SmallSet ){
423           if(sm.GetAttributeValue(Attr).equals(val)){
424            int oval = tempOccurances.get(Attr).get(val);
425            oval--;
```

```
426             if(oval == 0) valremovalList.add(val);
427             else tempOccurances.get(Attr).put(val, oval);
428            }
429           }
430          }
431          for(String a : valremovalList) tempOccurances.get(Attr).remove(a);
432         }
433        }
434        //Now we need to put in the values for the pivot
435        for(String Attr : tempOccurances.keySet()){
436         if(database.Data.PidAttrAndRanges.containsKey(Attr)){
437          if(tempOccurances.get(Attr).keySet().contains(pivot.PID.get(Attr))){
438           int oval = tempOccurances.get(Attr).get((pivot.PID.get(Attr))) + SmallSet.
                  size();
439           tempOccurances.get(Attr).put(pivot.PID.get(Attr), oval);
440          }
441          else{
442           tempOccurances.get(Attr).put(pivot.PID.get(Attr),  SmallSet.size());
443          }
444
445         }
446        }
447    }
448    private static HashMap<String, HashMap<String, Integer>>
449    MakeOccuranceTable(DB database,HashMap<String, HashMap<String, Integer>>
         tempOccurances,
450      Members mlg, PseudoIdentity pivot) {
451     int oval =0;
452     for(String Attr : database.Data.PidAttrAndRanges.keySet()){
453      ArrayList<String> valremovelist = new ArrayList<String>();
454       for(String val : tempOccurances.get(Attr).keySet()){
455        if(val.equals(mlg.GetAttributeValue(Attr))){
456         oval =  tempOccurances.get(Attr).get(val)-1;
457         if(oval == 0) valremovelist.add(val);
458         else tempOccurances.get(Attr).put(val,oval);
459         oval =  tempOccurances.get(Attr).get(pivot.PID.get(Attr))+1;
460         tempOccurances.get(Attr).put(pivot.PID.get(Attr),oval);
461        }
462       }
463       for(String a : valremovelist) tempOccurances.get(Attr).remove(a);
464     }
465     return tempOccurances;
466    }
467    public static DB ImplementNewClustering(HashMap<PseudoIdentity,PseudoIdentity>
         pivots,DB database){
468     HashMap<PseudoIdentity,PseudoIdentity> maptonewPID = new HashMap<PseudoIdentity,
         PseudoIdentity>();
```

```
469   for(PseudoIdentity keys : pivots.keySet()){
470    HashMap<String,String> temp = new HashMap<String,String>();
471    for(String Attr :keys.PID.keySet()){
472     if(keys.PID.get(Attr).compareTo(pivots.get(keys).PID.get(Attr))==0) temp.put(
            Attr, keys.PID.get(Attr));
473     else if(keys.PID.get(Attr).compareTo(pivots.get(keys).PID.get(Attr))!=0){
474      String newvalue = KAnonymitySupport.MakeAttrStrings(keys.PID.get(Attr),pivots.
           get(keys).PID.get(Attr),database.Maps.BasicMappings.get(Attr),database.
           Maps.BasicInverseMappings.get(Attr));
475      // new String('{'+keys.PID.get(Attr)+'|'+pivots.get(keys).PID.get(Attr)+'}');
476      String inverseNewValue = KAnonymitySupport.MakeAttrStrings(keys.PID.get(Attr),
           pivots.get(keys).PID.get(Attr),database.Maps.BasicMappings.get(Attr),
           database.Maps.BasicInverseMappings.get(Attr));
477      if(!database.Maps.Mappings.get(Attr).containsKey(newvalue)){
478       if(!database.Maps.Mappings.get(Attr).containsKey(inverseNewValue)){
479        Integer[] themappedtovalue = new Integer[database.Maps.Mappings.get(Attr).
             size()];
480        themappedtovalue = database.Maps.Mappings.get(Attr).values().toArray(
             themappedtovalue);
481        Arrays.sort(themappedtovalue);
482        int newintmap = themappedtovalue[themappedtovalue.length-1]+1;
483        database.Maps.Mappings.get(Attr).put(newvalue, newintmap);
484        database.Maps.InverseMappings.get(Attr).put(newintmap,newvalue);
485        database.Maps.RangeMaps.put(Attr,newintmap+1);
486        database.Occurances.get(Attr).put(newvalue, 0);
487        database.Data.PidAttrAndRanges.put(Attr, newintmap+1);
488        temp.put(Attr,newvalue);
489       }
490       else temp.put(Attr,inverseNewValue);
491      }
492      else temp.put(Attr,newvalue);
493     }
494    }
495    PseudoIdentity  temptotemp = new PseudoIdentity(temp);
496    maptonewPID.put(keys, temptotemp);
497   }
498   for(HashMap<String,String> UID : database.MembersList.keySet()){
499     PseudoIdentity MemPID = database.MembersList.get(UID).getPID(database.Data);
500     for(PseudoIdentity keys : pivots.keySet())
501      if(MemPID.equals(keys) || MemPID.equals(pivots.get(keys))){
502       for(String Attr : maptonewPID.get(keys).PID.keySet()){
503        if(database.MembersList.get(UID).GetAttributeValue(Attr).compareTo(
             maptonewPID.get(keys).PID.get(Attr)) !=0){
504          Members m = database.MembersList.get(UID);
505          String tempooo = new String (m.GetAttributeValue(Attr));
506          m.changeAnAttribute(maptonewPID.get(keys).PID.get(Attr), Attr,database);
507          database.MembersList.put(UID, m);
```

```
508             int tempppp = database.Occurances.get(Attr).get(tempooo);
509             tempppp--;
510             database.Occurances.get(Attr).put(tempooo,tempppp);
511             tempppp =  database.Occurances.get(Attr).get(maptonewPID.get(keys).PID.get
                    (Attr));
512             tempppp++;
513             database.Occurances.get(Attr).put(maptonewPID.get(keys).PID.get(Attr),
                    tempppp);
514           }
515           else continue;
516         }
517       }
518   }
519   return database;
520 }
521 public static DB ImplementNewClusteringForAncestor(PseudoIdentity TheChosen,
      PseudoIdentity TheOther,
522     PseudoIdentity TheUpgrade, DB database){
523     ArrayList<HashMap<String,String>> DBM = new ArrayList<HashMap<String,String>
          >(database.MembersList.keySet());
524     int totaladdition=0;
525     for(HashMap<String,String> M :DBM){
526       Members TM =database.MembersList.get(M);
527       PseudoIdentity TP = new PseudoIdentity(TM.getPID(database.Data));
528       if(TP.equals(TheChosen) || TP.equals(TheOther)){
529         for(String Attr : database.Data.PidAttrAndRanges.keySet()){
530           TM.changeAnAttribute(TheUpgrade.PID.get(Attr), Attr, database);
531           database.MembersList.put(M, TM);
532           if(TP.equals(TheChosen) && !TheChosen.PID.get(Attr).equals(TheUpgrade.PID.
                  get(Attr))){
533             int t = database.Occurances.get(Attr).get(TheChosen.PID.get(Attr));
534             database.Occurances.get(Attr).put(TheChosen.PID.get(Attr),t--);
535           }
536           if(TP.equals(TheOther) && !TheOther.PID.get(Attr).equals(TheUpgrade.PID.get(
                  Attr))){
537             int t = database.Occurances.get(Attr).get(TheOther.PID.get(Attr));
538             database.Occurances.get(Attr).put(TheOther.PID.get(Attr),t--);
539           }
540         }
541         totaladdition++;
542       }
543     }
544     for(String Attr : database.Data.PidAttrAndRanges.keySet()){
545       if(!database.Maps.Mappings.get(Attr).containsKey(TheUpgrade.PID.get(Attr))){
546         Integer[] themappedtovalue = new Integer[database.Maps.Mappings.get(Attr).
                size()];
547         themappedtovalue = database.Maps.Mappings.get(Attr).values().toArray(
```

```
            themappedtovalue);
548         Arrays.sort(themappedtovalue);
549         int newintmap = themappedtovalue[themappedtovalue.length-1]+1;
550         database.Maps.Mappings.get(Attr).put(TheUpgrade.PID.get(Attr), newintmap);
551         database.Maps.InverseMappings.get(Attr).put(newintmap,TheUpgrade.PID.get(
                Attr));
552         database.Maps.RangeMaps.put(Attr,newintmap+1);
553         database.Occurances.get(Attr).put(TheUpgrade.PID.get(Attr),totaladdition);
554         database.Data.PidAttrAndRanges.put(Attr, newintmap+1);
555       }
556     }
557     return database;
558   }
559   public static HashMap<PseudoIdentity,PseudoIdentity>
560   FindPivotsStarter(HashMap<PseudoIdentity, HashMap<PseudoIdentity,Double> >
          enthropies,PseudoIdentity TheChosen){
561    HashMap<PseudoIdentity,PseudoIdentity> pivots = new HashMap<PseudoIdentity,
           PseudoIdentity>();
562    for(PseudoIdentity k : enthropies.keySet()){
563     pivots.put(k, null);
564    }
565    PseudoIdentity[] A = new PseudoIdentity[enthropies.keySet().size()];
566    int i = 0;
567    for(PseudoIdentity a : enthropies.keySet()){
568     A[i] =a;
569     i++;
570    }
571    FindMaxPivots(A,enthropies,pivots,TheChosen);
572    return pivots;
573   }
574   public static double FindMaxPivots( PseudoIdentity[] identities,
575     HashMap<PseudoIdentity, HashMap<PseudoIdentity,Double> > enthropies,
576     HashMap<PseudoIdentity,PseudoIdentity> pivots,PseudoIdentity TheChosenID){
577     double cheapent = 0;
578     PseudoIdentity TheAnswer = null;
579     for(int position= 0; position < identities.length; position++){
580           if(enthropies.get(identities[position])!= null &&
581              enthropies.get(identities[position]).get(TheChosenID) != null &&
582           enthropies.get(identities[position]).get(TheChosenID) >= cheapent &&
583           identities[position] != TheChosenID){
584         TheAnswer = new PseudoIdentity(identities[position]);
585         cheapent=enthropies.get(identities[position]).get(TheChosenID);
586        }
587        else if (enthropies.get(TheChosenID)!= null && enthropies.get(TheChosenID).
               get(identities[position]) != null &&
588           enthropies.get(TheChosenID).get(identities[position]) >= cheapent){
589         TheAnswer = new PseudoIdentity(identities[position]);
```

```
590         cheapent=enthropies.get(TheChosenID).get(identities[position]) ;
591        }
592      }
593      for(PseudoIdentity a : pivots.keySet())pivots.put(a, a);
594      pivots.put(TheChosenID, TheAnswer);
595         return cheapent;
596  }
597  /**
598   * The return value from this function is a hash map where the key is the PID of a
         k-set and the
599   * second key is the PID of any cluster in the clusters. The value is the entropy
       when the two sets
600   * are combined.
601   * @param dataBase
602   * @param NeedToBeAno
603   * @param AllReadyHaveCurKAno
604   * @param info
605   * @return
606   */
607  public static HashMap<PseudoIdentity, HashMap<PseudoIdentity,Double> >
608  ComputeEntropies(DB dataBase, SupportClassOne info){
609   HashMap<PseudoIdentity, HashMap<PseudoIdentity,Double> > enthropies = new HashMap
         <PseudoIdentity, HashMap<PseudoIdentity,Double> >();
610   PseudoIdentity CurPid = null;
611   ArrayList<ArrayList<Members>> lister = new ArrayList<ArrayList<Members>>();
612   for(PseudoIdentity PID : info.EQclasses.keySet()) lister.add(info.EQclasses.get(
         PID));
613   for(ArrayList<Members> ListToBeKed :  lister ){
614    CurPid = ListToBeKed.get(0).getPID(dataBase.Data);
615    enthropies.put(CurPid, new HashMap<PseudoIdentity,Double>());
616    info.EQclasses.remove(CurPid);
617    for( PseudoIdentity Others : info.EQclasses.keySet()){
618     ArrayList<Double> entrovalue = DiffCluster(ListToBeKed.size(),dataBase.
           MembersList.size(),CurPid, Others,info,new HashMapCloner(dataBase.
           Occurances),dataBase);
619     double temppp = 0.0;
620     for(Double Ent :entrovalue) temppp+=Ent;
621     temppp/=entrovalue.size();
622     enthropies.get(CurPid).put(Others, temppp);
623    }
624   }
625     return enthropies;
626  }
627  private static ArrayList<Double> DiffCluster(int CurPidSize,int DBSize,
       PseudoIdentity CurPid,
628    PseudoIdentity Others, SupportClassOne info, HashMapCloner cloner,DB database){
629     String NewValue = null;
```

```
630     HashMapCloner ThisOccurance = cloner;
631     ArrayList<Double> entrovalue = new ArrayList<Double>();
632     for(String Attr : CurPid.PID.keySet()){
633      if(CurPid.PID.get(Attr).compareTo(Others.PID.get(Attr)) != 0){
634       NewValue = KAnonymitySupport.MakeAttrStrings(CurPid.PID.get(Attr),Others.PID.
              get(Attr),database.Maps.BasicMappings.get(Attr),database.Maps.
              BasicInverseMappings.get(Attr));
635       ThisOccurance.MyMap.get(Attr).put(NewValue, (CurPidSize+ info.EQclasses.get(
              Others).size()));
636       int temp = ThisOccurance.MyMap.get(Attr).get(CurPid.PID.get(Attr)).intValue()
              ;
637       temp -=CurPidSize;
638       ThisOccurance.MyMap.get(Attr).put(CurPid.PID.get(Attr), temp);
639       temp = ThisOccurance.MyMap.get(Attr).get(Others.PID.get(Attr)).intValue();
640       temp -= info.EQclasses.get(Others).size();
641       ThisOccurance.MyMap.get(Attr).put(Others.PID.get(Attr), temp);
642       entrovalue.add(KAnonymitySupport.enthropy(ThisOccurance.MyMap.get(Attr),
              DBSize));
643      }
644     }
645     return entrovalue;
646    }
647 }
```

B.12 SplitLargeAndOnlySmall.java

This is the class used for the algorithm that test split constraint and constrains the picked set.

```
1  package kAnonymity.MyKAnonymity;
2  import java.util.ArrayList;
3  import java.util.Arrays;
4  import java.util.Date;
5  import java.util.HashMap;
6  import java.util.Random;
7
8  import kAnonymity.KAnonymitySupport;
9  import kAnonymity.SupportClassOne;
10 import dataBase.DB;
11 import dataBase.Members;
12 import dataBase.PseudoIdentity;
13
14
15 public class K_Split_Algorithm {
16  static Random forpivots = new Random((new Date()).getTime());
```

```
17  public static DB KAnonymityAlgorithm(DB dataBase, Integer FinalK, int Metric) {
18  //  first I want to take care of the distance function
19    SupportClassOne infoNeededForLonger = null;
20    int thechosen = -1;
21    PseudoIdentity TheChosenOne = null;
22    HashMap<PseudoIdentity,Double>  enthropies = null;
23    SupportClassOne info = null;
24    DB newdataBase = new DB(dataBase);
25    info = KAnonymitySupport.GetEQClasses(newdataBase);
26    int rand = 0;
27    while(info.CurK < FinalK){
28     if(thechosen != -1) info = KAnonymitySupport.GetEQClasses(newdataBase);
29     boolean flag = true;
30     while(flag){
31      rand = forpivots.nextInt();
32      thechosen = Math.abs(rand) % info.EQclasses.keySet().size();
33      TheChosenOne = new PseudoIdentity((PseudoIdentity)info.EQclasses.keySet().
            toArray()[thechosen]);
34      if(info.EQclasses.get(TheChosenOne).size()> info.CurK)continue;
35      else flag = false;
36     }
37     infoNeededForLonger = new  SupportClassOne(info);
38     if(Metric == 0){
39      PseudoIdentity[] finals = new PseudoIdentity[info.EQclasses.keySet().size()-1];
40      enthropies = MyKAnonymityTwo.ComputeEntropies(newdataBase,info,TheChosenOne,
            finals);
41      PseudoIdentity pivot = TheChosenOne;
42      if(finals.length == 0 )continue;
43      if(infoNeededForLonger.EQclasses.get(pivot).size()+
44      infoNeededForLonger.EQclasses.get(finals[0]).size() < 2*FinalK){
45       newdataBase = MyKAnonymityTwo.ImplementNewClustering(TheChosenOne,finals[0],
             newdataBase,infoNeededForLonger);
46       infoNeededForLonger = null;
47      }
48      else{
49       if(infoNeededForLonger.EQclasses.get(pivot).size() >
50       infoNeededForLonger.EQclasses.get(finals[0]).size()){
51        Members[] ThechangedMembers = FindTheValuesToTransfer(infoNeededForLonger.
              EQclasses.get(pivot),infoNeededForLonger.EQclasses.get(finals[0]),
              infoNeededForLonger, dataBase,finals[0]);
52        if(ThechangedMembers!=null){
53         for(Members a : ThechangedMembers) newdataBase.RemoveAMember(newdataBase.
               MembersList.get(a.GetUID(newdataBase.Data)));
54         for(Members a : ThechangedMembers) newdataBase.AddAMember(new Members(a));
55         for(Members a : ThechangedMembers) newdataBase = MyKAnonymityTwo.
               ImplementNewClustering(TheChosenOne,a.getPID(newdataBase.Data),
               newdataBase,infoNeededForLonger);
```

```
56          }
57         }
58         else if(infoNeededForLonger.EQclasses.get(pivot).size() <
59         infoNeededForLonger.EQclasses.get(finals[0]).size()){
60          Members[] ThechangedMembers = FindTheValuesToTransfer(infoNeededForLonger.
                EQclasses.get(finals[0]),infoNeededForLonger.EQclasses.get(pivot),
                infoNeededForLonger, dataBase,pivot);
61          //There is a null point exception that happens once in awhile
62          if(ThechangedMembers != null){
63           for(Members a : ThechangedMembers) newdataBase.RemoveAMember(newdataBase.
                MembersList.get(a.GetUID(newdataBase.Data)));
64           for(Members a : ThechangedMembers) newdataBase.AddAMember(new Members(a));
65           for(Members a : ThechangedMembers) newdataBase = MyKAnonymityTwo.
                ImplementNewClustering(TheChosenOne,a.getPID(newdataBase.Data),
                newdataBase,infoNeededForLonger);
66          }
67         }
68        }
69       }
70       else if(Metric == 1){
71        HashMap<PseudoIdentity, HashMap<PseudoIdentity,Integer>> ancestorCost = new
              HashMap<PseudoIdentity, HashMap<PseudoIdentity,Integer>>();
72        PseudoIdentity[] Listinsortedorder = new PseudoIdentity[info.EQclasses.keySet()
              .size()-1];
73        int position = 0;
74        for(PseudoIdentity a : info.EQclasses.keySet()){
75         if(!TheChosenOne.equals(a)){
76          int temp = 0;
77          PseudoIdentity tempPID = KAnonymitySupport.AncestorMetric(TheChosenOne,a,
                dataBase,temp);
78          ancestorCost.put(a, new HashMap<PseudoIdentity,Integer>());
79          ancestorCost.get(a).put(tempPID, temp);
80          Listinsortedorder[position] = a;
81          position++;
82          inlinesortofPseudobyentAncestor(Listinsortedorder,ancestorCost,position-1);
83
84         }
85        }
86        if(Listinsortedorder.length == 0) continue;
87        PseudoIdentity finalchoice = new PseudoIdentity( Listinsortedorder[0]);
88        PseudoIdentity upgradefinal = new ArrayList<PseudoIdentity> (ancestorCost.get(
              Listinsortedorder[0]).keySet()).get(0);
89        int finalcost = ancestorCost.get(Listinsortedorder[0]).get(new ArrayList<
              PseudoIdentity> (ancestorCost.get(Listinsortedorder[0]).keySet()).get(0));
90        position = 1;
91        int smallestval = 0;
92        if(infoNeededForLonger.EQclasses.get(KAnonymitySupport.AncestorMetric(
```

```
        TheChosenOne,finalchoice, dataBase, 0))!= null)
93      smallestval =(int) (infoNeededForLonger.EQclasses.get(KAnonymitySupport.
            AncestorMetric(TheChosenOne,finalchoice, dataBase, 0)).size() +Math.ceil((
            infoNeededForLonger.EQclasses.get(TheChosenOne).size()+
94        infoNeededForLonger.EQclasses.get(finalchoice).size())/2 ));
95      int smallest = 0;
96      while(true){
97       if(position == Listinsortedorder.length ) break;;
98       PseudoIdentity Tempopo = KAnonymitySupport.AncestorMetric(TheChosenOne,
            finalchoice, dataBase, 0);
99       if(infoNeededForLonger.EQclasses.get(Tempopo) != null){
100       if(infoNeededForLonger.EQclasses.get(Tempopo).size()+Math.ceil((
              infoNeededForLonger.EQclasses.get(TheChosenOne).size()+
101         infoNeededForLonger.EQclasses.get(finalchoice).size())/2 ) >= 2*FinalK){
102        if(infoNeededForLonger.EQclasses.get(Tempopo).size()+Math.ceil((
              infoNeededForLonger.EQclasses.get(TheChosenOne).size()+
103          infoNeededForLonger.EQclasses.get(finalchoice).size())/2 ) < smallestval){
104         smallest = position;
105         smallestval = (int) (infoNeededForLonger.EQclasses.get(Tempopo).size()+
              Math.ceil((infoNeededForLonger.EQclasses.get(TheChosenOne).size()+
106           infoNeededForLonger.EQclasses.get(finalchoice).size())/2 ));
107        }
108        if(position >= Listinsortedorder.length /*||Listinsortedorder[position]==
              null*/ ) System.out.println(Listinsortedorder.length+"  "+position);
109        finalchoice = new PseudoIdentity( Listinsortedorder[position]);
110         upgradefinal = new ArrayList<PseudoIdentity> (ancestorCost.get(
              Listinsortedorder[position]).keySet()).get(0);
111        finalcost = ancestorCost.get(Listinsortedorder[position]).get(new ArrayList<
              PseudoIdentity> (ancestorCost.get(Listinsortedorder[position]).keySet())
              .get(0));
112        position++;
113        if(position >= Listinsortedorder.length -1 ){
114         finalchoice = new PseudoIdentity( Listinsortedorder[smallest]);
115          upgradefinal = new ArrayList<PseudoIdentity> (ancestorCost.get(
                Listinsortedorder[smallest]).keySet()).get(0);
116         finalcost = ancestorCost.get(Listinsortedorder[smallest]).get(new ArrayList
              <PseudoIdentity> (ancestorCost.get(Listinsortedorder[smallest]).keySet
              ()).get(0));
117         break;
118        }
119       }
120       else break;
121      }
122      else break;
123     }
124
125     //System.out.println(TheChosenOne +"\n"+finalchoice+"\n"+upgradefinal);
```

```
126     if(infoNeededForLonger.EQclasses.get(TheChosenOne).size()+
127     infoNeededForLonger.EQclasses.get(finalchoice).size() < 2*FinalK /*&& flaggy*/)
            {
128      newdataBase = ImplementNewClusteringForAncestor( TheChosenOne,  finalchoice,
             upgradefinal,newdataBase);
129     }
130     //we essentially need to get the old values from the database and find which
            would have
131     //fit best. This can be written tomorrow
132     else{
133      if(infoNeededForLonger.EQclasses.get(TheChosenOne).size() >
134      infoNeededForLonger.EQclasses.get(finalchoice).size() /*&& !flaggy*/){
135       //This finds the members to completely move to the other class.
136       Members[] ThechangedMembers = FindTheValuesToTransferAncestor(
              infoNeededForLonger.EQclasses.get(TheChosenOne),infoNeededForLonger.
              EQclasses.get(finalchoice),infoNeededForLonger, dataBase,finalchoice);
137       if(ThechangedMembers!=null){
138        for(Members a : ThechangedMembers) newdataBase.RemoveAMember(newdataBase.
               MembersList.get(a.GetUID(newdataBase.Data)));
139        for(Members a : ThechangedMembers) newdataBase.AddAMember(new Members(a));
140        for(Members a : ThechangedMembers)
141         newdataBase = ImplementNewClusteringForAncestor(new PseudoIdentity( a.
               getPID(dataBase.Data)),  finalchoice, finalchoice,newdataBase);
142        info = KAnonymitySupport.GetEQClasses(newdataBase);
143       }
144      }
145      else if(infoNeededForLonger.EQclasses.get(TheChosenOne).size() <
146      infoNeededForLonger.EQclasses.get(finalchoice).size() /*&& !flaggy*/){
147       Members[] ThechangedMembers = FindTheValuesToTransferAncestor(
              infoNeededForLonger.EQclasses.get(finalchoice),infoNeededForLonger.
              EQclasses.get(TheChosenOne),infoNeededForLonger, dataBase,TheChosenOne);
148       if(ThechangedMembers != null){
149        for(Members a : ThechangedMembers) newdataBase.RemoveAMember(newdataBase.
               MembersList.get(a.GetUID(newdataBase.Data)));
150        for(Members a : ThechangedMembers) newdataBase.AddAMember(new Members(a));
151        for(Members a : ThechangedMembers)
152         newdataBase = ImplementNewClusteringForAncestor( new PseudoIdentity( a.
               getPID(dataBase.Data)),  TheChosenOne, TheChosenOne,newdataBase);
153        info = KAnonymitySupport.GetEQClasses(newdataBase);
154       }
155      }
156     }
157
158    }
159    else if( Metric == 2){
160     HashMap<PseudoIdentity, HashMap<PseudoIdentity,Integer>>  ancestorCost = new
            HashMap<PseudoIdentity, HashMap<PseudoIdentity,Integer>>();
```

```
161     PseudoIdentity[] Listinsortedorder = new PseudoIdentity[info.EQclasses.keySet()
            .size()-1];
162     Listinsortedorder = ComputeGCP(newdataBase,info,TheChosenOne);
163     if(Listinsortedorder.length == 0)continue;
164     PseudoIdentity finalchoice = new PseudoIdentity( Listinsortedorder[0]);
165     PseudoIdentity upgradefinal = KAnonymitySupport.AncestorMetric(TheChosenOne,
            Listinsortedorder[0], dataBase,0);
166     int position = 1;
167     int smallestval = 0;
168     if(infoNeededForLonger.EQclasses.get(KAnonymitySupport.AncestorMetric(
            TheChosenOne,finalchoice, dataBase, 0))!= null)
169      smallestval =(int) (infoNeededForLonger.EQclasses.get(KAnonymitySupport.
             AncestorMetric(TheChosenOne,finalchoice, dataBase, 0)).size() +Math.ceil((
             infoNeededForLonger.EQclasses.get(TheChosenOne).size()+
170       infoNeededForLonger.EQclasses.get(finalchoice).size())/2 ));
171     int smallest = 0;
172     while(true){
173      if(position ==Listinsortedorder.length ) break;
174      PseudoIdentity Tempopo = KAnonymitySupport.AncestorMetric(TheChosenOne,
             finalchoice, dataBase, 0);
175      if(infoNeededForLonger.EQclasses.get(Tempopo) != null){
176       if(infoNeededForLonger.EQclasses.get(Tempopo).size()+Math.ceil((
              infoNeededForLonger.EQclasses.get(TheChosenOne).size()+
177         infoNeededForLonger.EQclasses.get(finalchoice).size())/2 ) >= 2*FinalK){
178        if(infoNeededForLonger.EQclasses.get(Tempopo).size()+Math.ceil((
               infoNeededForLonger.EQclasses.get(TheChosenOne).size()+
179         infoNeededForLonger.EQclasses.get(finalchoice).size())/2 ) < smallestval){
180        smallest = position;
181        smallestval = (int) (infoNeededForLonger.EQclasses.get(Tempopo).size()+
               Math.ceil((infoNeededForLonger.EQclasses.get(TheChosenOne).size()+
182          infoNeededForLonger.EQclasses.get(finalchoice).size())/2 ));
183        }
184        finalchoice = new PseudoIdentity( Listinsortedorder[position]);
185         upgradefinal = KAnonymitySupport.AncestorMetric(TheChosenOne,
               Listinsortedorder[position], dataBase, 0);
186        position++;
187        if(position >= Listinsortedorder.length -1 ){
188         finalchoice = new PseudoIdentity( Listinsortedorder[smallest]);
189          upgradefinal = KAnonymitySupport.AncestorMetric(TheChosenOne,
                 Listinsortedorder[position-1], dataBase, 0);
190         break;
191        }
192       }
193       else break;
194      }
195      else break;
196     }
```

```
197     //System.out.println(TheChosenOne +"\n"+finalchoice+"\n"+upgradefinal);
198     if(infoNeededForLonger.EQclasses.get(TheChosenOne).size()+
199     infoNeededForLonger.EQclasses.get(finalchoice).size() < 2*FinalK /*&& flaggy*/)
            {
200      newdataBase = ImplementNewClusteringForAncestor( TheChosenOne,  finalchoice,
             upgradefinal,newdataBase);
201     }
202     //we essentially need to get the old values from the database and find which
            would have
203     //fit best. This can be written tomorrow
204     else{
205      if(infoNeededForLonger.EQclasses.get(TheChosenOne).size() >
206      infoNeededForLonger.EQclasses.get(finalchoice).size() /*&& !flaggy*/){
207       //This finds the members to completely move to the other class.
208       Members[] ThechangedMembers = FindTheValuesToTransferAncestor(
              infoNeededForLonger.EQclasses.get(TheChosenOne),infoNeededForLonger.
              EQclasses.get(finalchoice),infoNeededForLonger, dataBase,finalchoice);
209       if(ThechangedMembers!=null){
210        for(Members a : ThechangedMembers) newdataBase.RemoveAMember(newdataBase.
             MembersList.get(a.GetUID(newdataBase.Data)));
211        for(Members a : ThechangedMembers) newdataBase.AddAMember(new Members(a));
212        for(Members a : ThechangedMembers)
213         newdataBase = ImplementNewClusteringForAncestor(new PseudoIdentity( a.
              getPID(dataBase.Data)),  finalchoice, finalchoice,newdataBase);
214        info = KAnonymitySupport.GetEQClasses(newdataBase);
215       }
216      }
217      else if(infoNeededForLonger.EQclasses.get(TheChosenOne).size() <
218      infoNeededForLonger.EQclasses.get(finalchoice).size() /*&& !flaggy*/){
219       Members[] ThechangedMembers = FindTheValuesToTransferAncestor(
              infoNeededForLonger.EQclasses.get(finalchoice),infoNeededForLonger.
              EQclasses.get(TheChosenOne),infoNeededForLonger, dataBase,TheChosenOne);
220       if(ThechangedMembers != null){
221        for(Members a : ThechangedMembers) newdataBase.RemoveAMember(newdataBase.
             MembersList.get(a.GetUID(newdataBase.Data)));
222        for(Members a : ThechangedMembers) newdataBase.AddAMember(new Members(a));
223        for(Members a : ThechangedMembers)
224         newdataBase = ImplementNewClusteringForAncestor( new PseudoIdentity( a.
              getPID(dataBase.Data)),  TheChosenOne, TheChosenOne,newdataBase);
225        info = KAnonymitySupport.GetEQClasses(newdataBase);
226       }
227      }
228     }
229
230    }
231    infoNeededForLonger = null;
232   }
```

```
233     return newdataBase;
234   }
235   private static PseudoIdentity[]  ComputeGCP(
236     DB newdataBase, SupportClassOne info, PseudoIdentity chosen) {
237    HashMap<PseudoIdentity, Double> retval = new HashMap<PseudoIdentity, Double>();
238    int position =0;
239    PseudoIdentity[] posretval = new PseudoIdentity[info.EQclasses.size()-1];
240    for(PseudoIdentity PID : info.EQclasses.keySet()){
241     if(!PID.equals(chosen)){
242      SupportClassOne tempinfo = new SupportClassOne(info);
243      PseudoIdentity tempPID = KAnonymitySupport.AncestorMetric(chosen,PID,
             newdataBase,0);
244      tempinfo.EQclasses.put(tempPID, new ArrayList<Members>(tempinfo.EQclasses.get(
             chosen)));
245      tempinfo.EQclasses.get(tempPID).addAll(tempinfo.EQclasses.get(PID));
246      tempinfo.EQclasses.remove(PID);
247      tempinfo.EQclasses.remove(chosen);
248      retval.put(PID, KAnonymitySupport.GCP(newdataBase, tempinfo.EQclasses));
249      posretval[position] = PID;
250      position++;
251      inlinesortofMembersbyentAncestor(posretval, retval,position -1);
252     }
253    }
254    return posretval;
255   }
256   private static void inlinesortofMembersbyentAncestor(
257     PseudoIdentity[] theLargeOrderedEnt, HashMap<PseudoIdentity, Double> ancestor,
258     int last) {
259    if(last != 0){
260     if(ancestor.get(theLargeOrderedEnt[last]) < ancestor.get(theLargeOrderedEnt[last
           -1])){
261      PseudoIdentity temp = theLargeOrderedEnt[last-1];
262      theLargeOrderedEnt[last-1] = theLargeOrderedEnt[last];
263      theLargeOrderedEnt[last] = temp;
264      inlinesortofMembersbyentAncestor(theLargeOrderedEnt,ancestor, last-1);
265     }
266    }
267   }
268   static Members[] FindTheValuesToTransfer(ArrayList<Members> large,
269     ArrayList<Members> small, SupportClassOne info,DB dataBase, PseudoIdentity pivot
           ) {
270    //We want to get the member from the original database here
271    ArrayList<Members> TheLargeSetMembers = new ArrayList<Members>();
272    ArrayList<Members> TheSmallSetMembers = new ArrayList<Members>();
273    //calculating the number of values we should have in the larger set
274    int SizeOfLarge = (int)Math.ceil(((double)large.size()+
275      (double)small.size())/2);
```

```
276     int valuesToMove =large.size()-SizeOfLarge;
277     //if no change can be done
278     if(valuesToMove == 0) return null;
279     //gets the smaller sets members
280     for(Members meq : small)TheSmallSetMembers.add(dataBase.MembersList.get(meq.
            GetUID((dataBase.Data))));
281     //gets the larger sets members
282     for(Members meq : large) TheLargeSetMembers.add(dataBase.MembersList.get(meq.
            GetUID((dataBase.Data))));
283     //Make the new hashtable to compute the entropy
284     HashMap<String, HashMap<String,Integer>> TempOccurances = new HashMap<String,
            HashMap<String,Integer>>();
285     FillNewHashWithOld(dataBase,small,pivot,TempOccurances, dataBase.Occurances);
286     HashMap<Members,Double> Entrophy = new HashMap<Members,Double>();
287     Members[] TheLargeOrderedEnt = new Members[large.size()];
288     int position = 0;
289     for(int i =0 ; i < TheLargeOrderedEnt.length; i++) TheLargeOrderedEnt[i] = null;
290     for(Members mlg : TheLargeSetMembers){
291      TempOccurances = MakeOccuranceTable(dataBase,TempOccurances,mlg,pivot);
292      double tempent = 0.0;
293      for(String Attr : TempOccurances.keySet())
294       tempent += KAnonymitySupport.enthropy(TempOccurances.get(Attr),dataBase.
              MembersList.size());
295      tempent/= TempOccurances.size();
296      Entrophy.put(mlg,tempent);
297      TempOccurances = revertOccuranceTable(dataBase,TempOccurances,mlg,pivot);
298      TheLargeOrderedEnt[position] = mlg;
299      position++;
300      inlinesortofMembersbyent(TheLargeOrderedEnt,Entrophy,position-1);
301     }
302     Members[] retval= new Members[valuesToMove];
303     for(int i = 0; i < valuesToMove; i++) retval[i] = TheLargeOrderedEnt[i];
304     return retval;
305   }
306   private static Members[] FindTheValuesToTransferAncestor(ArrayList<Members> large,
307     ArrayList<Members> small, SupportClassOne info,DB dataBase, PseudoIdentity pivot
          ) {
308     //We want to get the member from the original database here
309     ArrayList<Members> TheLargeSetMembers = new ArrayList<Members>();
310     ArrayList<Members> TheSmallSetMembers = new ArrayList<Members>();
311     //calculating the number of values we should have in the larger set
312     int SizeOfLarge = (int)Math.ceil(((double)large.size()+
313       (double)small.size())/2);
314     int valuesToMove =large.size()-SizeOfLarge;
315     //if no change can be done
316     if(valuesToMove == 0) return null;
317     //gets the smaller sets members
```

```
318     for(Members meq : small)TheSmallSetMembers.add(dataBase.MembersList.get(meq.
            GetUID((dataBase.Data))));
319     //gets the larger sets members
320     for(Members meq : large) TheLargeSetMembers.add(dataBase.MembersList.get(meq.
            GetUID((dataBase.Data))));
321     //Make the new hashtable to compute the entropy
322     HashMap<String, HashMap<String,Integer>> TempOccurances = new HashMap<String,
            HashMap<String,Integer>>();
323     FillNewHashWithOld(dataBase,small,pivot,TempOccurances, dataBase.Occurances);
324     HashMap<Members,Integer> Ancestor = new HashMap<Members,Integer>();//This will
            change TODO
325     Members[] TheLargeOrderedEnt = new Members[large.size()];
326     int position = 0;
327     for(int i =0 ; i < TheLargeOrderedEnt.length; i++) TheLargeOrderedEnt[i] = null;
328     for(Members mlg : TheLargeSetMembers){
329      TempOccurances = MakeOccuranceTable(dataBase,TempOccurances,mlg,pivot);
330      int tempent = 0;
331      KAnonymitySupport.AncestorMetric(new PseudoIdentity(mlg.getPID(dataBase.Data)),
             pivot, dataBase, tempent);
332      Ancestor.put(mlg,tempent);
333      TempOccurances = revertOccuranceTable(dataBase,TempOccurances,mlg,pivot);
334      TheLargeOrderedEnt[position] = mlg;
335      position++;
336      inlinesortofMembersbyentAncestor(TheLargeOrderedEnt,Ancestor,position-1);
337     }
338     Members[] retval= new Members[valuesToMove];
339     for(int i = 0; i < valuesToMove; i++) retval[i] = TheLargeOrderedEnt[i];
340     return retval;
341   }
342   private static void inlinesortofPseudobyentAncestor(
343     PseudoIdentity[] theLargeOrderedEnt, HashMap<PseudoIdentity, HashMap<
            PseudoIdentity, Integer>> ancestor,
344     int last) {
345     if(last != 0){
346      if(ancestor.get(theLargeOrderedEnt[last]).get(new ArrayList<PseudoIdentity>(
             ancestor.get(theLargeOrderedEnt[last]).keySet()).get(0)) <
347        ancestor.get(theLargeOrderedEnt[last-1]).get(new ArrayList<PseudoIdentity>(
               ancestor.get(theLargeOrderedEnt[last-1]).keySet()).get(0)) ){
348       PseudoIdentity temp = theLargeOrderedEnt[last-1];
349       theLargeOrderedEnt[last-1] = theLargeOrderedEnt[last];
350       theLargeOrderedEnt[last] = temp;
351       inlinesortofPseudobyentAncestor(theLargeOrderedEnt,ancestor, last-1);
352      }
353     }
354   }
355   private static void inlinesortofMembersbyentAncestor(
356     Members[] theLargeOrderedEnt, HashMap<Members, Integer> ancestor,
```

```
357     int last) {
358   if(last != 0){
359     if(ancestor.get(theLargeOrderedEnt[last]) < ancestor.get(theLargeOrderedEnt[last
            -1])){
360       Members temp = theLargeOrderedEnt[last-1];
361       theLargeOrderedEnt[last-1] = theLargeOrderedEnt[last];
362       theLargeOrderedEnt[last] = temp;
363       inlinesortofMembersbyentAncestor(theLargeOrderedEnt,ancestor, last-1);
364     }
365   }
366 }
367 private static void inlinesortofMembersbyent(Members[] theLargeOrderedEnt,
368   HashMap<Members, Double> entrophy, int last) {
369     if(last != 0){
370       if(entrophy.get(theLargeOrderedEnt[last]) < entrophy.get(theLargeOrderedEnt[
              last-1])){
371         Members temp = theLargeOrderedEnt[last-1];
372         theLargeOrderedEnt[last-1] = theLargeOrderedEnt[last];
373         theLargeOrderedEnt[last] = temp;
374         inlinesortofMembersbyent(theLargeOrderedEnt,entrophy, last-1);
375       }
376     }
377 }
378 //put a member back into the occurances when it is removed
379 private static HashMap<String, HashMap<String, Integer>> revertOccuranceTable(
380   DB database, HashMap<String, HashMap<String, Integer>> tempOccurances,
381   Members mlg, PseudoIdentity pivot) {
382   for(String Attr : tempOccurances.keySet()){
383     ArrayList<String> valremovalList = new ArrayList<String>();
384     if(database.Data.PidAttrAndRanges.containsKey(Attr)){
385       for(String val : tempOccurances.get(Attr).keySet() ){
386         if(val.equals(pivot.PID.get(Attr))){
387           int oval =  tempOccurances.get(Attr).get(pivot.PID.get(Attr));
388           oval--;
389           if(oval == 0)valremovalList.add(pivot.PID.get(Attr));
390           else tempOccurances.get(Attr).put(pivot.PID.get(Attr),oval);
391         }
392       }
393       for(String a : valremovalList) tempOccurances.get(Attr).remove(a);
394       if(tempOccurances.get(Attr).keySet().contains(mlg.GetAttributeValue(Attr))){
395         int oval = tempOccurances.get(Attr).get(mlg.GetAttributeValue(Attr));
396         oval++;
397         tempOccurances.get(Attr).put(mlg.GetAttributeValue(Attr),oval);
398       }
399       else tempOccurances.get(Attr).put(mlg.GetAttributeValue(Attr),1);
400     }
401   }
```

```
402     for(String Attr : tempOccurances.keySet()){
403      if(database.Data.PidAttrAndRanges.containsKey(Attr)){
404       if(tempOccurances.get(Attr).containsKey(mlg.GetAttributeValue(Attr))){
405        int oval =  tempOccurances.get(Attr).get(mlg.GetAttributeValue(Attr));
406        oval++;
407        tempOccurances.get(Attr).put(mlg.GetAttributeValue(Attr),oval);
408       }
409       else tempOccurances.get(Attr).put(mlg.GetAttributeValue(Attr),1);
410      }
411     }
412     return tempOccurances;
413    }
414    //This function will copy the hashmap from the database which holds all the
           occurances
415    //in the database.
416    private static void FillNewHashWithOld(DB database, ArrayList<Members> SmallSet,
           PseudoIdentity pivot,
417      HashMap<String, HashMap<String, Integer>> tempOccurances,
418      HashMap<String, HashMap<String, Integer>> occurances) {
419      for(String Attr : occurances.keySet()){
420       tempOccurances.put(Attr, new HashMap<String, Integer>());
421       for(String val : occurances.get(Attr).keySet())
422        tempOccurances.get(Attr).put(new String(val),new Integer(occurances.get(Attr).
               get(val).intValue()));
423      }
424      //This will go through the occurences hashmap and remove all occurances of the
           members in the
425      //smaller set. They will be reset with the values in the pivot pseudoidentity.
426      for(String Attr : tempOccurances.keySet()){
427       ArrayList<String> valremovalList = new ArrayList<String>();
428       if(database.Data.PidAttrAndRanges.containsKey(Attr)){
429        for(String val : tempOccurances.get(Attr).keySet() ){
430         for(Members sm: SmallSet ){
431          if(sm.GetAttributeValue(Attr).equals(val)){
432           int oval = tempOccurances.get(Attr).get(val);
433           oval--;
434           if(oval == 0) valremovalList.add(val);
435           else tempOccurances.get(Attr).put(val, oval);
436          }
437         }
438        }
439        for(String a : valremovalList) tempOccurances.get(Attr).remove(a);
440       }
441      }
442      //Now we need to put in the values for the pivot
443      for(String Attr : tempOccurances.keySet()){
444       if(database.Data.PidAttrAndRanges.containsKey(Attr)){
```

```
445       if(tempOccurances.get(Attr).keySet().contains(pivot.PID.get(Attr))){
446        int oval = tempOccurances.get(Attr).get((pivot.PID.get(Attr))) + SmallSet.
               size();
447        tempOccurances.get(Attr).put(pivot.PID.get(Attr), oval);
448       }
449       else{
450        tempOccurances.get(Attr).put(pivot.PID.get(Attr),  SmallSet.size());
451       }
452
453      }
454     }
455   }
456   private static HashMap<String, HashMap<String, Integer>>
457   MakeOccuranceTable(DB database,HashMap<String, HashMap<String, Integer>>
          tempOccurances,
458     Members mlg, PseudoIdentity pivot) {
459    int oval =0;
460    for(String Attr : database.Data.PidAttrAndRanges.keySet()){
461     ArrayList<String> valremovelist = new ArrayList<String>();
462      for(String val : tempOccurances.get(Attr).keySet()){
463       if(val.equals(mlg.GetAttributeValue(Attr))){
464        oval =  tempOccurances.get(Attr).get(val)-1;
465        if(oval == 0) valremovelist.add(val);
466        else tempOccurances.get(Attr).put(val,oval);
467        oval =  tempOccurances.get(Attr).get(pivot.PID.get(Attr))+1;
468        tempOccurances.get(Attr).put(pivot.PID.get(Attr),oval);
469       }
470      }
471      for(String a : valremovelist) tempOccurances.get(Attr).remove(a);
472    }
473    return tempOccurances;
474   }
475   public static DB ImplementNewClusteringForAncestor(PseudoIdentity TheChosen,
          PseudoIdentity TheOther,
476     PseudoIdentity TheUpgrade, DB database){
477      ArrayList<HashMap<String,String> > DBM = new ArrayList<HashMap<String,String>
           >(database.MembersList.keySet());
478      int totaladdition=0;
479      for(HashMap<String,String> M :DBM){
480       Members TM =database.MembersList.get(M);
481       PseudoIdentity TP = new PseudoIdentity(TM.getPID(database.Data));
482       if(TP.equals(TheChosen) || TP.equals(TheOther)){
483        for(String Attr : database.Data.PidAttrAndRanges.keySet()){
484         TM.changeAnAttribute(TheUpgrade.PID.get(Attr), Attr, database);
485         database.MembersList.put(M, TM);
486         if(TP.equals(TheChosen) && !TheChosen.PID.get(Attr).equals(TheUpgrade.PID.
               get(Attr))){
```

```
487             int t = database.Occurances.get(Attr).get(TheChosen.PID.get(Attr));
488             database.Occurances.get(Attr).put(TheChosen.PID.get(Attr),t--);
489            }
490            if(TP.equals(TheOther) && !TheOther.PID.get(Attr).equals(TheUpgrade.PID.get(
                   Attr))){
491             int t = database.Occurances.get(Attr).get(TheOther.PID.get(Attr));
492             database.Occurances.get(Attr).put(TheOther.PID.get(Attr),t--);
493            }
494           }
495           totaladdition++;
496          }
497         }
498         for(String Attr : database.Data.PidAttrAndRanges.keySet()){
499          if(!database.Maps.Mappings.get(Attr).containsKey(TheUpgrade.PID.get(Attr))){
500            Integer[] themappedtovalue = new Integer[database.Maps.Mappings.get(Attr).
                   size()];
501            themappedtovalue = database.Maps.Mappings.get(Attr).values().toArray(
                   themappedtovalue);
502            Arrays.sort(themappedtovalue);
503            int newintmap = themappedtovalue[themappedtovalue.length-1]+1;
504            database.Maps.Mappings.get(Attr).put(TheUpgrade.PID.get(Attr), newintmap);
505            database.Maps.InverseMappings.get(Attr).put(newintmap,TheUpgrade.PID.get(
                   Attr));
506            database.Maps.RangeMaps.put(Attr,newintmap+1);
507            database.Occurances.get(Attr).put(TheUpgrade.PID.get(Attr),totaladdition);
508            database.Data.PidAttrAndRanges.put(Attr, newintmap+1);
509          }
510        }
511        return database;
512    }
513    public static DB ImplementNewClustering(HashMap<PseudoIdentity,PseudoIdentity>
           pivots,DB database){
514     HashMap<PseudoIdentity,PseudoIdentity> maptonewPID = new HashMap<PseudoIdentity,
            PseudoIdentity>();
515     for(PseudoIdentity keys : pivots.keySet()){
516      HashMap<String,String> temp = new HashMap<String,String>();
517      for(String Attr :keys.PID.keySet()){
518       if(keys.PID.get(Attr).compareTo(pivots.get(keys).PID.get(Attr))==0) temp.put(
              Attr, keys.PID.get(Attr));
519       else if(keys.PID.get(Attr).compareTo(pivots.get(keys).PID.get(Attr))!=0){
520        String newvalue = KAnonymitySupport.MakeAttrStrings(keys.PID.get(Attr),pivots.
               get(keys).PID.get(Attr),database.Maps.BasicMappings.get(Attr),database.
               Maps.BasicInverseMappings.get(Attr));
521        // new String('{'+keys.PID.get(Attr)+'|'+pivots.get(keys).PID.get(Attr)+'}');
522        String inverseNewValue = KAnonymitySupport.MakeAttrStrings(keys.PID.get(Attr),
               pivots.get(keys).PID.get(Attr),database.Maps.BasicMappings.get(Attr),
               database.Maps.BasicInverseMappings.get(Attr));
```

```
523       if(!database.Maps.Mappings.get(Attr).containsKey(newvalue)){
524        if(!database.Maps.Mappings.get(Attr).containsKey(inverseNewValue)){
525         Integer[] themappedtovalue = new Integer[database.Maps.Mappings.get(Attr).
                size()];
526         themappedtovalue = database.Maps.Mappings.get(Attr).values().toArray(
                themappedtovalue);
527         Arrays.sort(themappedtovalue);
528         int newintmap = themappedtovalue[themappedtovalue.length-1]+1;
529         database.Maps.Mappings.get(Attr).put(newvalue, newintmap);
530         database.Maps.InverseMappings.get(Attr).put(newintmap,newvalue);
531         database.Maps.RangeMaps.put(Attr,newintmap+1);
532         database.Occurances.get(Attr).put(newvalue, 0);
533         database.Data.PidAttrAndRanges.put(Attr, newintmap+1);
534         temp.put(Attr,newvalue);
535        }
536        else temp.put(Attr,inverseNewValue);
537       }
538       else temp.put(Attr,newvalue);
539      }
540     }
541     PseudoIdentity  temptotemp = new PseudoIdentity(temp);
542     maptonewPID.put(keys, temptotemp);
543    }
544    for(HashMap<String,String> UID : database.MembersList.keySet()){
545      PseudoIdentity MemPID = database.MembersList.get(UID).getPID(database.Data);
546      for(PseudoIdentity keys : pivots.keySet())
547       if(MemPID.equals(keys) || MemPID.equals(pivots.get(keys))){
548        for(String Attr : maptonewPID.get(keys).PID.keySet()){
549         if(database.MembersList.get(UID).GetAttributeValue(Attr).compareTo(
                maptonewPID.get(keys).PID.get(Attr)) !=0){
550           Members m = database.MembersList.get(UID);
551           String tempooo = new String (m.GetAttributeValue(Attr));
552           m.changeAnAttribute(maptonewPID.get(keys).PID.get(Attr), Attr,database);
553           database.MembersList.put(UID, m);
554           int tempppp = database.Occurances.get(Attr).get(tempooo);
555           tempppp--;
556           database.Occurances.get(Attr).put(tempooo,tempppp);
557           tempppp =  database.Occurances.get(Attr).get(maptonewPID.get(keys).PID.get
                (Attr));
558           tempppp++;
559           database.Occurances.get(Attr).put(maptonewPID.get(keys).PID.get(Attr),
                tempppp);
560         }
561         else continue;
562        }
563       }
564    }
```

```
565     return database;
566   }
567   public static HashMap<PseudoIdentity,PseudoIdentity>
568   FindPivotsStarter(HashMap<PseudoIdentity, HashMap<PseudoIdentity,Double> >
          enthropies,PseudoIdentity TheChosen){
569    HashMap<PseudoIdentity,PseudoIdentity> pivots = new HashMap<PseudoIdentity,
           PseudoIdentity>();
570    for(PseudoIdentity k : enthropies.keySet()){
571     pivots.put(k, null);
572    }
573    PseudoIdentity[] A = new PseudoIdentity[enthropies.keySet().size()];
574    int i = 0;
575    for(PseudoIdentity a : enthropies.keySet()){
576     A[i] =a;
577     i++;
578    }
579    FindMaxPivots(A,enthropies,pivots,TheChosen);
580    return pivots;
581   }
582   public static double FindMaxPivots( PseudoIdentity[] identities,
583     HashMap<PseudoIdentity, HashMap<PseudoIdentity,Double> > enthropies,
584     HashMap<PseudoIdentity,PseudoIdentity> pivots,PseudoIdentity TheChosenID){
585     double cheapent = 0;
586     PseudoIdentity TheAnswer = null;
587     for(int position= 0; position < identities.length; position++){
588           if(enthropies.get(identities[position])!= null &&
589              enthropies.get(identities[position]).get(TheChosenID) != null &&
590           enthropies.get(identities[position]).get(TheChosenID) >= cheapent &&
591           identities[position] != TheChosenID){
592         TheAnswer = new PseudoIdentity(identities[position]);
593         cheapent=enthropies.get(identities[position]).get(TheChosenID);
594        }
595        else if (enthropies.get(TheChosenID)!= null && enthropies.get(TheChosenID).
               get(identities[position]) != null &&
596           enthropies.get(TheChosenID).get(identities[position]) >= cheapent){
597         TheAnswer = new PseudoIdentity(identities[position]);
598         cheapent=enthropies.get(TheChosenID).get(identities[position]) ;
599        }
600      }
601     for(PseudoIdentity a : pivots.keySet())pivots.put(a, a);
602     pivots.put(TheChosenID, TheAnswer);
603        return cheapent;
604   }
605   /**
606    * The return value from this function is a hash map where the key is the PID of a
           k-set and the
607    * second key is the PID of any cluster in the clusters. The value is the entropy
```

```
        when the two sets
608      * are combined.
609      * @param dataBase
610      * @param NeedToBeAno
611      * @param AllReadyHaveCurKAno
612      * @param info
613      * @return
614      */
615     public static HashMap<PseudoIdentity, HashMap<PseudoIdentity,Double> >
616     ComputeEntropies(DB dataBase, SupportClassOne info){
617      HashMap<PseudoIdentity, HashMap<PseudoIdentity,Double> > enthropies = new HashMap
            <PseudoIdentity, HashMap<PseudoIdentity,Double> >();
618      PseudoIdentity CurPid = null;
619      ArrayList<ArrayList<Members>> lister = new ArrayList<ArrayList<Members>>();
620      for(PseudoIdentity PID : info.EQclasses.keySet()) lister.add(info.EQclasses.get(
            PID));
621      for(ArrayList<Members> ListToBeKed :  lister ){
622       CurPid = ListToBeKed.get(0).getPID(dataBase.Data);
623       enthropies.put(CurPid, new HashMap<PseudoIdentity,Double>());
624       info.EQclasses.remove(CurPid);
625       for( PseudoIdentity Others : info.EQclasses.keySet()){
626        ArrayList<Double> entrovalue = DiffCluster(ListToBeKed.size(),dataBase.
               MembersList.size(),CurPid, Others,info,new HashMapCloner(dataBase.
               Occurances),dataBase);
627        double temppp = 0.0;
628        for(Double Ent :entrovalue) temppp+=Ent;
629        temppp/=entrovalue.size();
630        enthropies.get(CurPid).put(Others, temppp);
631       }
632      }
633       return enthropies;
634     }
635     private static ArrayList<Double> DiffCluster(int CurPidSize,int DBSize,
            PseudoIdentity CurPid,
636       PseudoIdentity Others, SupportClassOne info, HashMapCloner cloner,DB database){
637        String NewValue = null;
638        HashMapCloner ThisOccurance = cloner;
639        ArrayList<Double> entrovalue = new ArrayList<Double>();
640        for(String Attr : CurPid.PID.keySet()){
641         if(CurPid.PID.get(Attr).compareTo(Others.PID.get(Attr)) != 0){
642          NewValue = KAnonymitySupport.MakeAttrStrings(CurPid.PID.get(Attr),Others.PID.
                 get(Attr),database.Maps.BasicMappings.get(Attr),database.Maps.
                 BasicInverseMappings.get(Attr));
643          ThisOccurance.MyMap.get(Attr).put(NewValue, (CurPidSize+ info.EQclasses.get(
                 Others).size()));
644          int temp = ThisOccurance.MyMap.get(Attr).get(CurPid.PID.get(Attr)).intValue()
                 ;
```

```
645         temp -=CurPidSize;
646         ThisOccurance.MyMap.get(Attr).put(CurPid.PID.get(Attr), temp);
647         temp = ThisOccurance.MyMap.get(Attr).get(Others.PID.get(Attr)).intValue();
648         temp -= info.EQclasses.get(Others).size();
649         ThisOccurance.MyMap.get(Attr).put(Others.PID.get(Attr), temp);
650         entrovalue.add(KAnonymitySupport.enthropy(ThisOccurance.MyMap.get(Attr),
                DBSize));
651       }
652      }
653      return entrovalue;
654     }
655 }
```