Journal of Privacy and Confidentiality (2014) **6**, Number 2, 81–115

A Graph-based Approach to Key Variable Mapping

Duncan Smith* and Mark Elliot†

1 Introduction

1.1 Background

Statistical disclosure control (SDC) can be characterised as a three stage control process:

(1) Identify the key variables that might be used by an intruder to attack the data, via matching to members of the relevant population.

(2) Carry out risk analyses using the variables identified at stage 1.

(3) Employ disclosure control technqiues to reduce the risk identified at stage 2.

Extensive research on stages 2 and 3 of this process has been carried out over the last 30 years.[1] While there is still room for improvement, our understanding of the statistical processes involved in a statistical disclosure are now quite sophisticated. This is not true of stage 1, where our understanding is comparatively crude. This was the problem that drove the development of the metadata repository and the form of analysis presented here.

One formulation of the SDC problem has been in terms of intrusion scenarios (see [3]). An intrusion scenario is a description of the motivation that might lie behind an attack on data, and the means employed to launch the attack by a given type of *data intruder*. An intrusion scenario coupled with data (already released and under consideration for release) can be used to generate measures of disclosure risk by data stewardship organizations.[2] A risk exceeding some predefined threshold might trigger the application of disclosure limitation methods.

An attack scenario inevitably includes some form of matching between published data and information that is known, or is discoverable, about the targeted individuals or organizations. For instance, a target who is a university lecturer can be matched against the records within a database corresponding to people working in academia. A

*University of Manchester, United Kingdom, `mailto:duncan.g.smith@manchester.ac.uk`.
†University of Manchester, United Kingdom, `mailto:mark.elliot@manchester.ac.uk`.

[1]For recent reviews of the SDC area, see [1] and [2].

[2]Following Duncan et al. (2011) we use the term *Data Stewardship Organizations* to refer to any organization which has the joint responsibilities of disseminating data for statistical purposes and maintaining the confidentiality of the population units represented in that data. This includes, but is by no means limited to, National Statistical Institutes.

match might not be a correct match—there might be many matches within the database, the target might not be contained within the database, or the data within the database might be outdated or simply incorrect. These are all factors that might be considered when assessing disclosure risks for specific databases. Matching is essentially a synonym for identifying a subset of the records within a database that, under the assumptions that the matched record is within the database and the matching information is correct, must contain the matched record. Any member of this subset is a match, while only one match is the correct match.

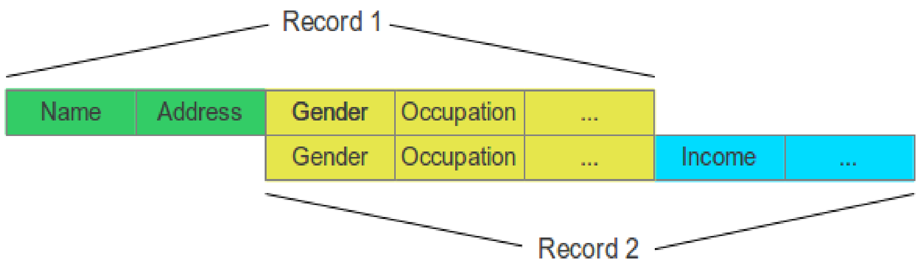

Figure 1: Record matching.

By matching on known levels of variables, an intruder might be able to associate a target with a specific record (identification) or a set of records that share some common, previously unknown attribute value(s) (attribution). The standard example is illustrated in Figure 1. Record 1 (which might simply be the information known about a target) is matched against Record 2 on a subset of the variables common to both records. If the match is known to be correct (e.g., the data are population data and it is the only possible match), then the intruder discovers the values of the other variables in Record 2.

For reasons of practicality, risk assessment often assumes a set of key variables and considers the risk associated with a proposed data release in isolation. The set of key variables will often be based on the variables that are assumed to be either directly observable or in the public domain. For example, gender and occupation might be key variables under a 'nosy neighbour' attack scenario. This type of ad hoc subjective key variable specification is better than an arbitrary selection, but nevertheless it falls well short of a fully specified scenario. An intruder might also have access to rich data that are neither directly observable nor in the public domain; for instance, data collected by a specific organization for its own administrative or business purposes. Furthermore, an intruder might be able to combine data from several databases in order to construct larger keys with which to attack a target dataset.

This paper is concerned with addressing the weakness in the attack scenario formulation—the identification of the key variables likely to be used for matching. In [4], Elliot et al. describe a process of *Data Environment Analysis* (DEA). DEA systematically captures metadata from data collection instruments such as questionnaires. This is used to construct a queryable metadata repository. The information in the repository is used to identify potential key variables via a process termed Key Variable Mapping (KVM).

2 The Key Variable Mapping System

The Key Variable Mapping System (KVMS) consists of a metadata repository, generated by DEA, and analysis tools to perform KVM. A primitive version of KVMS described in [4] was implemented in a spreadsheet using Visual Basic macros to generate outputs. An overview of this system will be presented here, before the revised system is described in detail in the subsequent sections.

Collection instruments (questionnaires, application forms, etc.) will be termed *forms*. Forms are in the public domain, and are collected via a number of means. Many forms are available online. The databases they give rise to are not generally in the public domain, but might be available to an intruder under a given attack scenario. Each form is essentially a list of questions with a list of possible responses for each question. Form field analysis is the process of entering form metadata, including the questions and possible responses, into the spreadsheet. Each set of responses is mapped to a *capture code*, consisting of a letter followed by an integer denoting the number of possible responses, followed by a dot, followed by another integer to disambiguate the code from other codes with an equal number of responses. Multiple forms might have identical possible responses to a common question and thus share a capture code.

For example, KVMS contains a number of forms with a question relating to 'General Health'. For one of these forms the possible responses are 'Good', 'Fairly Good', and 'Not Good'. The corresponding code is C3.001, the C denoting that the code is a *capture code* (distinguishing it from another type of code that we will describe later) with 3 levels and the 001 denoting that it is the first such code encountered. Capture codes for 'General Health' are shown in Table 1.

If each response in a set of responses A can be mapped to exactly one response in a set of responses B, then it is said that A *harmonizes* to B. Harmonization relations are the basis of the KVMS output, and are contained in a separate worksheet. The simplest way to visualize these relations is with a graph.

The graph in Figure 2 represents the (as we will subsequently see, flawed) logic in KVMS for General Health. H2.001 represents an unobserved set of responses to which C3.001 and C4.001 are assumed to harmonize. The H stands for *harmonization code*. Although this code is not actually contained in the KVMS repository, it is intended to convey information regarding the matching possibilities between A and B. It is the unique code to which A and B both harmonize that has a maximum number of levels, which we term the *harmonization* of A and B. For example, it might represent the notional 'GOOD' and 'POOR' responses illustrated in Figure 3. The harmonization of two (or more) questions represents a partitioning of records for matching purposes.

There is clearly a transitive relation, in that if A harmonizes to B, and B harmonizes

General Health	**C500000.001**	**15.001**	**C4.001**	**C3.001**
Good				X
Fairly Good				X
Not Good				X
excellent			X	
very good			X	
good			X	
poor			X	
hay fever / eczema		X		
chest condition		X		
...		X		
other illness		X		
write-in condition	X			

Table 1: Capture codes for ‘General Health’.

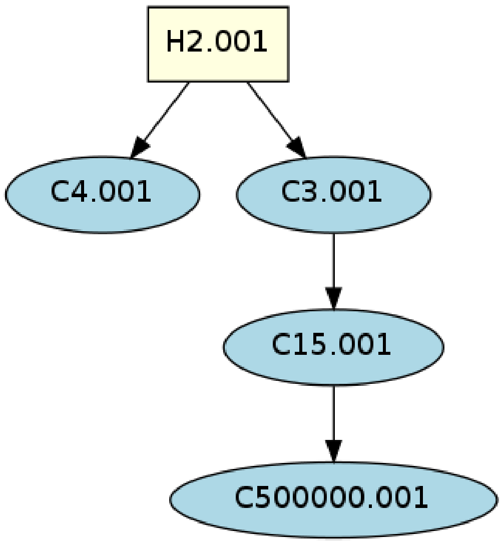

Figure 2: Harmonization graph for General Health as implicitly represented in the original KVMS.

to C, then A harmonizes to C. So in Figure 2, C500000.001 and C15.001 both harmonize to C3.001 (and H2.001). In Figure 2 the harmonization of C500000.001 and C15.001 is therefore C15.001. We can see from Figure 3 that the harmonization relation does not generally describe all matching possibilities—although C4.0001 does not harmonize to C3.001, 'Not Good' can be matched against 'poor' and *vice versa*. The harmonization relation describes cases where each possible response on one code can be mapped to (and therefore directly matched against) exactly one possible response on a second code.

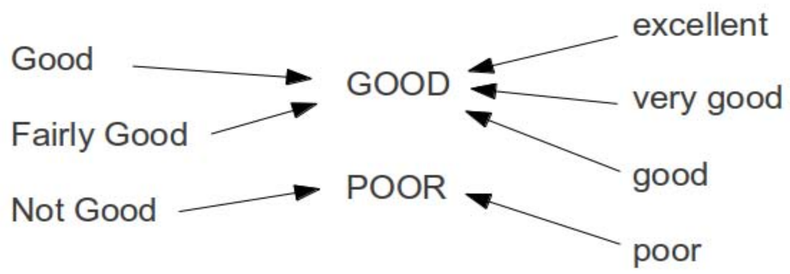

Figure 3: Possible mapping of responses to H2.001.

2.1 KVM Analysis

Analysis proceeds as follows; where $|\cdot|$ denotes set cardinality and $\lfloor\cdot\rfloor$ denotes the floor function.

1. Select a target form and a set E of forms for matching against the target form.
2. Specify a *prevalence* parameter.
3. For each question X in the target form:
    i. Let S be the set containing all the elements of E that contain an instance of X.
    ii. If $|S| < |E| \cdot p$ then the output for X is null.
    iii. Otherwise, the output for X is a maximal harmonization over the set containing the harmonizations for X for each combination of the elements of S of size $\lfloor |E| \cdot p \rfloor$ and the target form.

So p and E define a threshold number of forms, $N = \lfloor |E| \cdot p \rfloor$, that can exploit X for matching against the target form, above which X might be considered a key variable. Each combination, of size N, of the members of E will produce a harmonization code. The harmonization of this code with that of X from the target form will generate another code. The output for X is such a code with a maximal number of levels (there might be more than one such code). This code provides an indication of the risk associated with matching on X against the target form. A large numbers of levels would suggest greater risk.

2.2 Issues

The General Health example highlights several issues. The 'write-in condition' response allows an arbitrarily high number of possible responses, although the user has specified

500,000 levels. The harmonization graph implies that each of these can be mapped to one of the 15 categories in C15.001. This might not be the case, as the conditions in C15.001 are not exhaustive. Also, an individual might suffer from more than one of the conditions in C15.001, in which case a 'write-in condition' response might map to more than one condition in C15.001. The responses for C15.0001 are not mutually exclusive. Although it may be a single question on a form, C15.001 should really be treated as a number of distinct questions with yes/no responses. In reality there are 2^{15} possible responses to C15.001. It is also not clear that each response (or combination of responses) in C15.001 maps to a single response in C3.001. Does having hay fever imply poor health? Also, if C15.001 harmonizes to C3.001, then why does it not also harmonize to C4.001?

Although the individual performing DEA has clearly made assumptions of the nature shown in Figure 3, they are not recorded in KVMS. It is possible to enter subsequent capture codes for the question, and their relationships with other codes, using an alternative set of assumptions. This can lead to inconsistencies. In some cases these can be revealed by examining a graph such as that in Figure 2. For instance, the lack of a graph edge from C4.001 to C500000.001 suggests a problem. Not all inconsistencies can be identified in this way, and KVMS does not generate these graphs (so that they can be inspected) or perform such checks.

The basic underlying issue is that questions and sets of responses do not correspond to variables and categories. Matching takes place between the categories of a common variable, as illustrated in Figure 3. Although distinct variables might be stochastically dependent, inferences based on such a relationship are probabilistic rather than logical and could be dealt with using alternative approaches. The General Health example is an attempt to try to capture both logical and stochastic relationships simultaneously. In any case, KVMS is built on form metadata which provide no information regarding stochastic relationships between variables.

This is not to say that we can completely exclude uncertainty by focussing on individual variables. Even given our assumptions about the correctness of the data in the underlying databases, we still need to deal with contextual issues and semantics. In Figure 3 the word 'good' (with its various degrees of capitalization) means different things due to differences in context. It is also not certain that 'Not Good' would mean the same as 'poor' to all respondents. Although we could attempt to explicitly handle this uncertainty within KVMS, we simply note this as a possible extension of the approach. Currently this source of uncertainty is dealt with in a relatively ad hoc manner—we try to make reasonable choices. For instance, in Figure 3 it might have been decided that context suggested that 'excellent' and 'good' would map to 'Good', and that 'good' would map to 'Fairly Good'. In that case C4.001 would have harmonized to C3.001.

The most recent incarnation of KVMS seeks to address the above issues. It is firmly based on variables and categories. Data structures and algorithms are developed to make matching assumptions both explicit and consistent. The concept of a harmonization graph is formalized and an algorithm is presented for automating harmonization graph construction from form metadata and the specified matching assumptions. An efficient algorithm for performing KVM analysis is presented. This new version of KVMS is mainly implemented in the Python[3] programming language with a wxPython[4] user interface. It is far more flexible than the original spreadsheet implementation. Usability issues have been addressed, and KVM outputs can be generated much more efficiently. It can be easily extended to generate alternative outputs, and can be scripted in Python.

3 The Key Variable Mapping System II

Subsequent references to KVMS will refer to the new KVMS. Any references to the earlier version will be clear from the context. Much of the focus of subsequent sections is on data structures and algorithms. In part this is necessary in order to demonstrate the differences between the new and original KVMS, but it also provides insight into how the system might be extended via scripting. The existing output is used as an exemplar. For the purposes of subsequent sections several terms will be defined in detail, although some of these terms were introduced in the previous section.

Categorization – A categorization is a set of mutually exclusive and exhaustive subcategories for a variable—a partition of the variable's sample space. Distinct instances of the same variable might have distinct categorizations.

Code – A code is a unique identifier for a given categorization of a given variable. It consists of a letter denoting the type of code, followed by an integer denoting the number of categories, a dot, and another integer to disambiguate the code from other codes for the same variable with an equal number of categories. For instance, C3.002 would indicate a categorization for a given variable containing 3 categories. It would be distinct from at least one other categorization of the variable within the data environment that also contains 3 categories (i.e., C3.001). Leading zeros are used to accommodate up to 999 categorizations for the same variable with an equal number of categories.

Harmonization – The members of a set, C, of categorizations of a variable are said to harmonize to a categorization h if every category contained in the members of C is a subcategory of a category of h. The harmonization of a set of categorizations is the unique maximal categorization to which they harmonize. Here maximality is with respect to the number of categories. For example, the harmonization of {England, Scotland, Wales, Northern Ireland or Non-UK} and {England or Wales, Scotland, Northern Ireland, Non-UK} would be {England or Wales, Scotland, Northern Ireland or Non-UK}, even though both also harmonize to less detailed categorizations such as {England or Wales or Scotland, Northern Ireland or Non-UK}.

The first purpose of KVMS is to represent a data environment. The data stored in KVMS relating to a data environment will be referred to as a *virtual data environment.* At the moment this is restricted to metadata derived from explicit collection instruments (although it could in principle be expanded to include other types of information). A one to one correspondence is assumed to exist between collection instruments and the

[3]See `http://www.python.org/`.
[4]See `http://www.wxpython.org/`.

databases in a data environment. Thus a *form* within the virtual data environment represents a single collection instrument and is assumed to relate to a single database. A form contains arbitrary metadata (such as organization type, address etc.) and a collection of variable instances corresponding to the variables assumed to be present in the corresponding database. Each variable instance contains a set of categories corresponding to the categorization that is assumed to exist for the variable in the database. In some cases, the variables and their categorizations are clear, although the KVMS user must use his/her judgement in other cases. For instance, 'country of birth' and 'town of birth' relate to the same underlying variable 'place of birth'. The individual performing data entry must recognize this and enter the information appropriately in order to account for the matching possibilities. KVMS only recognizes matching possibilities between distinct instances of the same variable.

Matching between forms is possible only if they contain at least one common variable; the matching possibilities are dictated by their categorizations. Variables that are common to multiple forms are recognized by ensuring that variable instances share the same name. The data entry system presents the user with a list of existing variables to select from to minimize the chances of creating multiply-named instances of the same variable. Relating the categories of a variable is done using a *partition graph.* Thus, the virtual data environment consists of a collection of forms (each containing one or more variable instances/categorizations) and a collection of partition graphs—one partition graph for each distinct variable contained within the virtual data environment.

3.1 Partition Graphs

A categorization is easily defined (as above), but identifying matching possibilities across distinct categorizations is not quite so straightforward. The user is required to specify the relationships between categories by constructing a partition graph. The partition graph encodes the matching possibilities for a variable.

A partition graph for a variable X is a directed acyclic graph $G(V, E)$ where each $v \in V$ is a non-empty category of X. The children of a node v are a set of subcategories of v that constitute a partition of the sample space of v. The set of sink nodes (nodes with zero children) of G are a discrete sample space for X. For convenience, let us say that a directed acyclic graph with these properties has the *partition graph property.* Figure 4 shows such a graph for a hypothetical variable *Place of Residence.*

This clearly has the partition graph property. Britain comprises of England, Scotland, and Wales; the UK comprises of Britain and Northern Ireland; and non Britain comprises of non UK and Northern Ireland. The sink nodes include nodes for all countries in the UK and a non UK node.

The partition graph property implies a simple way for checking that a categorization is valid, that is, comprises of a set of mutually exclusive and exhaustive categories:

A categorization C is valid with respect to a partition graph G if, and only if, all sink nodes in G are reachable from exactly one node in C.

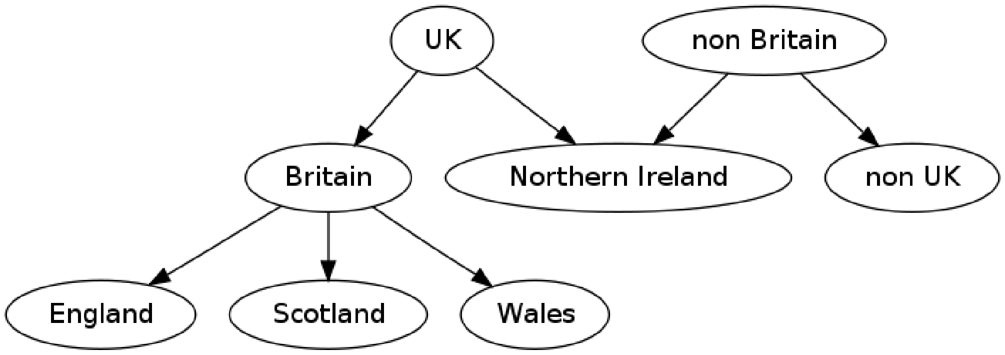

Figure 4: A partition graph for Place of Residence.

> *If any sink node is not reachable from a node in C, then the categorization is not exhaustive.*
>
> *If any sink node is reachable from more than one node in C, then the categories are not mutually exclusive.*

Partition graphs are not generally unique. In Figure 4, UK could have been made a parent of England, Scotland, Wales, and Northern Ireland. This graph is shown in Figure 5. The graph still possesses the partition graph property and admits the same set of categorizations.

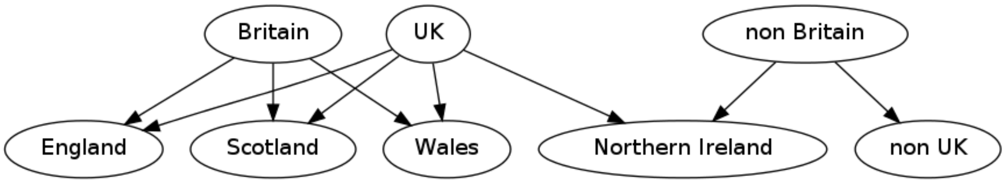

Figure 5: Alternative partition graph for Place of Residence.

Thus a given set of categorizations of a variable corresponds to an equivalence class of partition graphs. It is possible to permute graphs according to rules that maintain the partition graph property. Such permutations are not detailed here, but it is possible to produce graphs that are more pleasing visually or more efficient for analytical purposes. It is important to be able to identify relationships between categories. In Figure 4 the {UK, Britain} edge implies that Britain is a subcategory of UK. However, there is no path from UK to Britain in Figure 5. A more general criterion for identifying subcategories is used within KVMS.

> *A node v is a subcategory of a node w if, and only if, the sink nodes reachable from v are a subset of the sink nodes reachable from w.*

Partition Graph Construction

A partition graph can be constructed incrementally as additional collection instruments are captured and new databases represented. For example, an initial categorization {England, Scotland, Wales, non Britain} would simply result in a graph with 4 nodes and no edges. Accommodating a second categorization {Britain, non Britain} would just require the addition of Britain as a parent of England, Scotland, and Wales. Adding {UK, non UK} adds the node UK as a parent of Britain and the node Northern Ireland as a child of UK. A non UK node would also be added and non Britain would be made a parent of Northern Ireland and non UK. These incremental additions would produce the graph in Figure 4.

Although it might not result in the most pleasing looking graph, the algorithm *add_node* will add a node v to a non-empty partition graph G so that the partition graph property is maintained (see Appendices for pseudocode).

The algorithm only considers the sink nodes of the existing graph. No existing nodes or edges are removed, and no edges are added from existing non-sink nodes.

A graph can be initialized by adding a single node for the variable. If not, it must be ensured that the first categorization added contains categories that are a partition of the sample space of the relevant variable. Once a valid non-empty graph is generated *add_node* can be used to extend it, and the validity of any subsequently added categorizations can be checked using the reachability criterion described earlier.

The *add_node* algorithm cannot generally be used for automatic construction of a graph. It is mainly intended as a procedural guide for users who wish to construct a graph manually. However, it can be used to automatically construct graphs for variables on interval scales that have been categorized into subintervals. Figure 6 shows the partition graph produced by adding the categorizations {'0-18', '18-infinity'} and {'0-16', '16-21', '21-infinity'} for the variable Age using the algorithm. Assume we added a new form to the data environment with categorization {'0-18', '18-21', '21-infinity'}. As the relevant nodes are already present, the partition graph does not require updating, and it is a trivial exercise to check that the categorization is also valid.

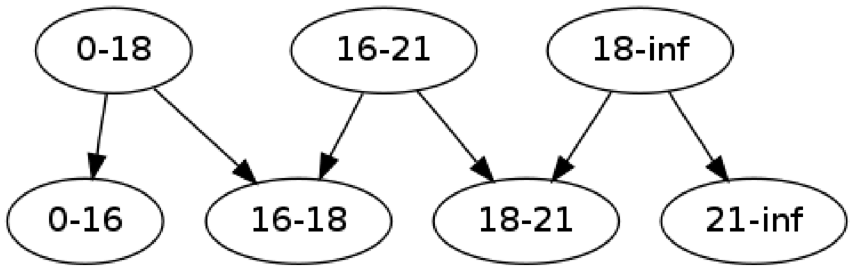

Figure 6: A partition graph for Age generated using a simple algorithm.

The user is responsible for creating and updating a partition graph for each variable within the virtual data environment. An editor is provided which allows nodes and edges to be added via a simple graphical interface. Restrictions are placed upon the edits, so that an edited graph remains consistent with previous incarnations—a new graph will always admit all the categorizations admitted by previous versions. For example, the user is not permitted to delete any existing nodes or edges, or add new edges from existing non-sink nodes. The *add_node* algorithm demonstrates that such edits are never required. These restrictions do not require the user to employ the algorithm, and the user is still free to produce alternative layouts. But they do prevent the user from introducing inconsistent information.[5]

During form entry a categorization for a variable is specified by selecting a subset of nodes in the relevant partition graph. The user either selects an existing variable or creates a new variable before (perhaps) editing the corresponding partition graph and selecting the relevant nodes. Validity of the categorization is automatically checked using the reachability criterion. Selecting an empty set of nodes implies that the form allows an arbitrary level of precision. For example, a form might ask the respondent to write in their age, rather than selecting an age range. Such a categorization is assumed to harmonize to all other categorizations of the relevant variable.

In summary, the virtual data environment consists of forms, variables/categorizations, and partition graphs to relate the categorizations. A partition graph encodes all the possibilities for matching between the categories of a variable. A relatively simple algorithm can be used to construct a partition graph incrementally as new categories are met. Restrictions on allowable edits enable simple consistency checks, ensuring that specified categorizations are both mutually exclusive and exhaustive.

The discussion so far has mainly related to the construction of the virtual data environment. This is a continuing process—new forms, variables, and categories being added as they are encountered. The following sections relate to the existing output, although some of the data structures and algorithms might be of wider interest. Currently all these data structures are generated from the information in the virtual data environment each time an analysis is conducted, reflecting the fact that the virtual data environment is subject to continuous change.

4 Categorizations and Harmonization

A partition graph admits a family of discrete sample spaces for a node v in G. The children of v represent one member of this family. However, the sink nodes reachable from v constitute an alternative sample space. If the latter is generated for each category in a valid categorization of X, then the categorization can be represented as a set of discrete sample spaces, the union of which contains exactly the sink nodes of G. It is this form of sample space that reliably encodes matching possibilities for all valid partition graphs—because it contains the sink nodes. This is the representation that is used in KVMS for generating harmonizations. Thus the categorization {Britain, Northern

[5] A separate unrestricted editor is also provided, but this is intended primarily to enable users to correct their own errors.

Ireland, non UK} would produce the set partition representation {{England, Scotland, Wales}, {Northern Ireland}, {non UK}} with respect to the partition graph in Figure 4.

If every element of a set partition α is a subset of some element of a set partition β, then α is termed a *refinement* of β; α is said to be finer than β and β is said to be coarser than α. If α is finer than β and not equal to β, then α is said to be strictly finer than β and β is said to be strictly coarser than α.

Thus the set partition representation allows harmonization to be characterized as follows:

> *The harmonization h of two categorizations C1, C2 (in terms of the sink node set partition representation) is the unique set partition that is the finest of all the set partitions that are coarser than both C1 and C2.*

If α is a refinement of β then each element of β is the union of one or more elements of α. Thus, the elements of α can be aggregated (using set union) to produce β. Therefore, the harmonization, h, of two categorizations, C1 and C2, can be expressed as set operations on their set partition representations.

Given this set partition, representation each sink node, v, must be contained in exactly one element of C1, exactly one element of C2, and exactly one element of h. Thus the element in h which contains v must be a superset of the element in C1 containing v and the element of C2 containing v. If not, then C1 and C2 are not refinements of h. If this holds for all sink nodes, then all elements of C1 and C2 are subsets of some element of h and they are therefore refinements of h. So, performing set unions on the elements of C1 and C2 which have non-empty intersection will produce their harmonization, h. KVMS implements these operations by mapping them to operations on graphs.

In order to compute harmonizations the categorizations are represented as a disjoint set forest. A categorization with n categories is represented as a forest containing n trees. Each tree contains the sink nodes that are reachable from the corresponding category in the partition graph. The nodes within a tree can be connected arbitrarily. Disjoint set forests support two basic functions. The *find* function finds the source node of the tree containing a node v by searching along the path from v to the source. The *union* function joins the trees containing distinct nodes v and w by finding their sources and making one of the sources a child of the other source (if the sources are distinct). A sensible implementation flattens a tree during a call to the *find* function by disconnecting all the non-source nodes on the searched path from their parents and making them children of the source node. It is a simple task to maintain a reference to the tree depths, so that *union* attaches a tree with lower depth to the source node of a tree with higher depth. Tarjan [5] shows that such an implementation results in *find* and *union* functions that take amortized constant execution time.

Consider the categorizations {Britain, Northern Ireland, non UK}, {UK, non UK}, and the partition graph in Figure 4. Represented as sink node partitions these would be

{{England, Scotland, Wales}, {Northern Ireland}, {non UK}}, and {{England, Scotland, Wales, Northern Ireland}, {non UK}} respectively. Corresponding disjoint set forests are shown in Figures 7 and 8. Of course, disjoint set forests are not generally unique, as a collection of nodes can be organized into a rooted tree in multiple ways.

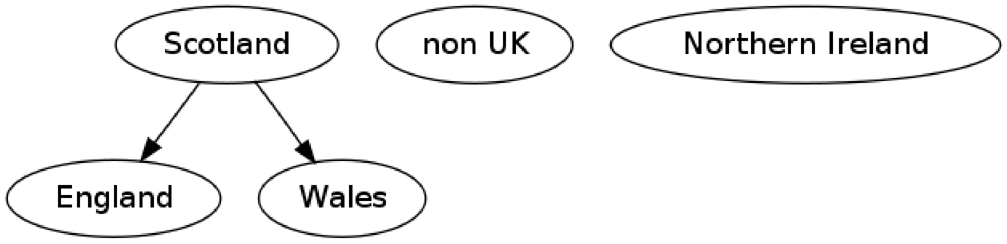

Figure 7: Disjoint set forest for the categorization {Britain, Northern Ireland, non UK}.

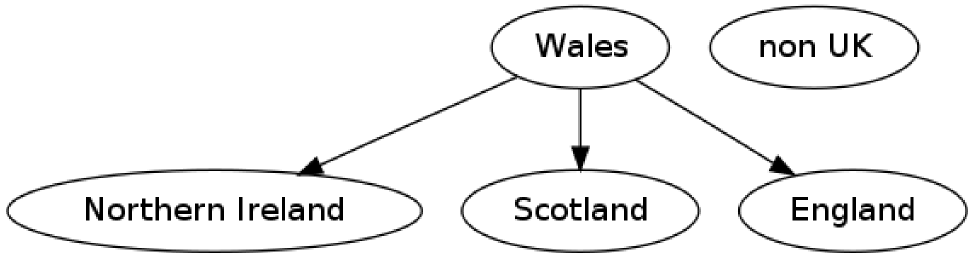

Figure 8: Disjoint set forest for the categorization {UK, non UK}.

Creating a disjoint set forest from a set partition requires each node to be added as a distinct tree (containing a single root node) and a call to *union* for each edge in the forest. Thus creation has amortized O(n) complexity for a categorization in a partition graph with n sink nodes.

The algorithm *harmonize* will produce the harmonization of a set of categorizations (see Appendices for pseudocode). Its time complexity is amortized O(mn) for harmonizing a set containing m categorizations from a partition graph with n sink nodes.

The harmonization of the categorizations in Figures 7 and 8 is shown in Figure 9. Note that the harmonization of these categorizations is the same as the categorization shown in Figure 8. This is because the categorization in Figure 7 is a refinement of the categorization in Figure 8. Also note that the graph in Figure 9 is not the same as the graph in Figure 8. Yet they represent the same categorization because they contain the same connected components.

4.1 Harmonization Graphs

The refinement relation admits a partial ordering on the possible set partitions, and

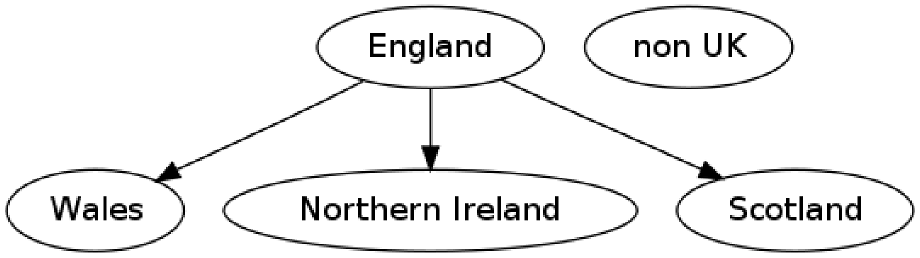

Figure 9: The harmonization of the categorizations in Figures 7 and 8, {{England, Scotland, Wales, Northern Ireland}, {non UK}}.

therefore on the set of categorizations for a given variable within a virtual data environment. The set of categorizations for a given variable within a virtual data environment and their possible harmonizations can be used to construct a directed acyclic graph $G(V, E)$ where V is the set of all observed categorizations and their possible harmonizations, and such that a path $v \rightarrow w$, $\{v, w\} \subset V$ implies that w is a refinement of v. These graphs are constructed automatically from the data in the repository, and additional restrictions are imposed on their structure.

Let $\mathrm{H}(\cdot)$ denote the function which returns the harmonization of its arguments.

1. The harmonization graph G for a set of categorizations, C, of a given variable has node set equal to the union of C and the set containing the harmonizations for all non-empty subsets of C.
2. For each pair of nodes $\{v, w\} \subset V$, if v is coarser than w, then there exists a path $v \rightarrow w$ in G.
3. G is the unique graph that satisfies 1 and 2 and has a minimum number of edges.

It is possible to construct a graph that possesses Properties 1 and 2 above, but that does not have Property 3. The transitive reduction of such a graph possesses all three properties ([6]). The transitive reduction of a directed acyclic graph is unique. Therefore, a set of categorizations necessarily gives rise to a unique harmonization graph.

The size of the power set of C increases exponentially with the size of C, so generating all possible harmonizations can be computationally expensive. In practice the number of distinct harmonizations tends to be much smaller than the power set of C, so an algorithm is developed that ensures that all the necessary harmonizations are contained in the graph without having to generate a harmonization for each member of the power set of C.

The algorithm adds categorizations from C to a directed acyclic graph $G(V, E)$ in an arbitrary order, ensuring that the graph structure is updated to produce a valid harmonization graph (with respect to the subset of the virtual data environment containing the added categorizations) on each node addition.

For any given pair of *distinct* categorizations $\{v,w\}$ the refinement relationship can be tested thus:

> *$H(v,w) = v$ implies that v is strictly coarser than w, and $H(v,w) = w$ implies that v is strictly finer than w.*

Node Insertion

Assume that there exists a harmonization graph G (possibly empty) and we wish to add a new node u. The immediate issue is how the edges must be updated to satisfy Properties 2 and 3 after the addition of u. Property 2 implies that there must exist paths to u from all nodes, and only those nodes, that are coarser than u. Furthermore, there must exist paths from u to all nodes, and only those nodes, that are finer than u. The nodes that are coarser than u and the nodes that are finer than u induce subgraphs of G. If we consider the subgraph induced by the nodes coarser than u, then it is clear that making its sink nodes parents of u ensures that exactly the required paths to u are present. It is also clear that adding any edges between the non-sink nodes of this subgraph to u would introduce redundant edges. A similar argument demonstrates that the children of u must be exactly the source nodes of the subgraph induced by the nodes that are finer than u. The remaining issue is whether the addition of u has rendered any of the existing edges redundant.

An edge (v,w) is redundant if, and only if, there exists a path $v \rightarrow w$ in G of length greater than 1. All the transitive relationships that are implied by (v,w) are also implied by the path $v \rightarrow w$ of length greater than 1, so (v,w) must be removed.

So we only need to consider each edge (v,w) that was not redundant before the addition of u and has been rendered redundant by the creation of a $v \rightarrow w$ path of length greater than 1. Thus only edges from ancestors of u to descendants of u need be considered. Property 2 implies that before the addition of u there must have existed a path from each parent of u to each child of u. Thus there are no edges from any non parental ancestors of u to descendants of u or from ancestors of u to non child descendants of u. These would have been redundant before the insertion of u. This leaves only any edges between parents of u and children of u. These are clearly redundant and must be removed.

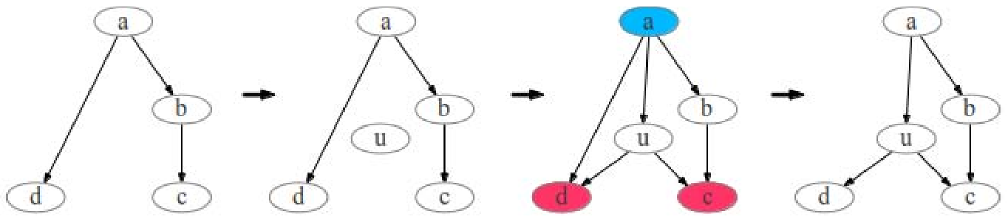

Figure 10: Addition of a new node to an existing harmonization graph.

The sequence of steps for node addition are shown in Figure 10. The node is added, then the parents and children of u are identified and edges to/from u added, then superfluous edges are removed.

This still leaves two issues—the identification of the parents and children of an inserted node u, and the identification of the nodes (harmonizations) to be inserted to satisfy Property 1.

Identification of Parents and Children

The parents and children of u could be identified via standard traversal algorithms such as depth first search (see e.g., [7]). This would involve generating the harmonization of each visited node x with u to establish whether the node was coarser or finer than u, or neither. A more efficient approach is to exploit the transitive relationships in the graph to reduce the number of harmonization generations that are required.

Firstly we discuss how to partition the nodes of G into nodes that are coarser than u, nodes that are finer than u, and nodes that are neither coarser nor finer than u. The nodes of G are visited and processed in an arbitrary order. For each unvisited node x in G we generate $\mathrm{H}(x,u)$. This allows us to add x to the relevant set. If x is coarser than u, then we can add it and each of its unvisited ancestors to the set of nodes coarser than u. Similarly, if x is finer than u, then we can add it and each of its unvisited descendants to the set of nodes finer than u. If x is neither coarser nor finer than u, then we add it to the set of nodes that are neither coarser nor finer than u. Each processed node is marked as visited, as are the nodes visited during the ancestral and descendant searches. Thus we only calculate $\mathrm{H}(x,u)$ for a subset of the nodes in G.

In order to identify the parents of u we note that parents of u are the only nodes in G that result in an ancestral search, but are never visited in an ancestral search. An ancestral search starting at x stops when it meets previously visited nodes. These nodes and their ancestors have already been added to the set of nodes that are coarser than u, so processing can stop. Each ancestral node that is not a parent of u will either be initially marked as visited during an ancestral search from a descendant that is also coarser than u or be met as a previously visited node indicating that the search should stop. A similar argument applies to the identification of the children of u.

The only remaining issue for constructing a harmonization graph is the generation of the nodes to add. The harmonization graph must contain the harmonization for each non-empty subset of the observed categorizations.

Harmonization Generation

Clearly, the observed categorizations need to be added, and these are placed in an initial list of nodes to be added. On adding a node u we know that $\mathrm{H}(x,u)$ already exists in G if x is either an ancestor or descendant of u after the addition of u. $\mathrm{H}(x,u)$ can only be a new harmonization if x is neither coarser nor finer than u. These harmonizations are generated during the addition of u. If we also add these nodes to G after the addition of an observed categorization, then we can ensure that all the necessary harmonizations have been added to produce a valid harmonization graph after the addition of an observed categorization.

So after addition of an observed categorization the set of generated harmonizations are also added to G. For addition of these harmonizations the node insertion algorithm can be somewhat simplified. Only these harmonizations are required to generate a valid harmonization graph. Any new harmonizations generated during the node addition algorithm can be ignored—in fact, any such harmonizations are already queued for addition.

General Algorithm

The *insert_node* algorithm combines the above to generate a harmonization graph from a set of observed categorizations (see Appendices for pseudocode). The unvisited descendants of a node are found via a standard depth first traversal using previously visited nodes as sentinels to limit the search. The unvisited ancestors are found using an essentially identical approach but searching along the edges in the reverse direction.

Harmonization Graphs - Discussion

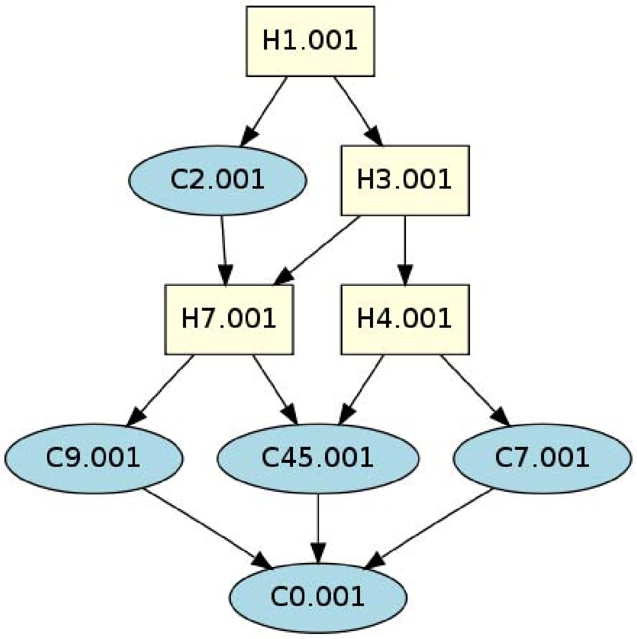

Figure 11: Harmonization graph for Age based on a data environment containing 191 forms.

Figure 11 is the harmonization graph for the variable Age from a virtual data environment containing 191 UK datasets, the vast majority of which contain a variable instance for Age. The corresponding partition graph contains 103 nodes in total with 61 sink nodes. Yet there are only 5 distinct codes for Age within the virtual data environment, and only 4 additional harmonization codes required to construct the harmonization graph. Codes that appear within the data environment are shown as blue ellipses, to distinguish them from the generated harmonization codes which are shown as yellow rectangles. The two types of code are also distinguished by their leading letter. As previously indicated, the convention for distinguishing different categorizations of the same type and with equal numbers of categories is to use the number of categories and a unique (zero padded) integer separated by a dot. Thus C45.001 is a code, for a categorization of Age, which is found within the data environment and has 45 categories. If there were another categorization of the same type and with the same number of categories it would be assigned the identifier C45.002.

The C0.001 category is a special category corresponding to datasets containing 'Date of Birth' or where there is no specified categorization. Forms where a respondent is asked to write in a value result in the same type of code. It is assumed that the user could provide an arbitrary level of detail, and that harmonization with any other categorization c will reproduce c. Although the code seems to imply 0 categories, it actually means 0 specified categories; in reality it is an arbitrarily high number of categories, and has to be special cased for some purposes. Note that any harmonization graph that contains an arbitrary precision node will necessarily have that node as a single sink node. All harmonization graphs have a single source node corresponding to the harmonization of all observed codes.

Figure 11 shows that there are only 5 distinct categorizations for age across all the datasets within the data environment, and one of these allows the user to provide an arbitrary degree of precision. If trying to match on Age between two datasets with categorizations C9.001 and C45.001, then any potential match can be mapped to one of 7 categories. These categories are given by the unique sink node in the graph intersection of their ancestral graphs, H7.001. Similarly, trying to match across C9.001, C45.001, and C7.001 will allow any matches (across all three datasets) to be mapped to one of three categories, given by H3.001. Closer inspection of the categorizations would be required to identify these categories. Some might provide more detail than others (narrow age bands) and their sensitivity might vary. In terms of usefulness for matching the number of forms containing these categorizations is also relevant. Having many forms with categorizations C9.001 and C45.001 would suggest that Age is more useful for matching than if most of the forms had categorizations C2.001 and C7.001 (which harmonize to H1.001).

It should be noted that harmonization graphs can be a little misleading regarding matching possibilities. The codes C2.001 and C7.001 correspond to the categorizations {'0-16', '16-inf'} and {'0-21', '21-25', '25-35', '35-45', '45-55', '55-65', '65-inf'}, respectively. The category '0-16' can certainly be matched against '0-21'. All but the first category in C7.001 can be matched against '16-inf'. We could meet a more degenerate situation where, say, we had two categorizations for Age with very narrow age bands but no shared boundaries. For instance, {'0-2', '2-4', '4-6', ..., '78-80', '80-inf'} and

{'0-1', '1-3', '3-5', ..., '77-79', '79-inf'} would also harmonize to H1.001, yet there are many matching possibilities. For example, '2-4' can be matched against an aggregated category '1-5'. Although such possibilities tend to be disguised in the harmonization graph, they are present in the partition graph.

The work described in this paper relates to the re-implementation of the original KVMS. Although the opportunity was taken to place the system on a sounder footing by focussing on variables and categorizations, alternative outputs were not explored. Harmonization relates to the exemplar output. Alternative outputs might more fully exploit the more detailed information in partition graphs.

5 Analysis

KVMS output was described in Section 2.1. It is essentially unchanged from that in the original KVMS, although the data structures and algorithms used to generate them are very different. As the outputs relate directly to variables rather than questions they are more meaningful. The purpose of the analysis is to provide an empirically grounded set of key variables which a data stewardship organization could then use as an input to a disclosure risk assessment exercise. This section describes how harmonization graphs can be exploited to generate the KVMS output efficiently.

To recap:

Analysis proceeds as follows; where $|\cdot|$ denotes set cardinality and $\lfloor\cdot\rfloor$ denotes the floor function.

1. Select a target form and a set E of forms for matching against the target form.
2. Specify a parameter, $0 \le p \le 1$.
3. For each variable X in the target form:
   i. Let S be the set containing all the elements of E that contain an instance of X.
   ii. If $|S| < |E| \cdot p$ then the output for X is null.
   iii. Otherwise, the output for X is a maximal harmonization over the set containing the harmonizations for X for each combination of the elements of S of size $\lfloor |E| \cdot p \rfloor$ and the target form.

Essentially a prevalence parameter, p, is specified and this is used to calculate an integer parameter, $N = \lfloor |E| \cdot p \rfloor$. Then a maximal harmonization $h = \mathrm{H}(x_0, x_1, \ldots, x_N)$ is sought, where $x_1, \ldots, x_N$ are categorizations of the relevant variable from any N distinct datasets in the data environment and x_0 is the categorization within the target dataset. Note: the categorizations are not assumed to be distinct.

Naively iterating through all the combinations of forms of a given size that contain the relevant variable can be costly. A more focused search can be employed using the harmonization graph for the variable in question, constructed using all the forms in the data environment, E.[6]

Assume we have a valid harmonization graph, G, for X (note: a harmonization graph constructed from a superset of the data environment is still valid). We know that the output for X is a node in G. We also know that the maximum number of observed categorizations that can harmonize to a node v is the number of forms in E containing v plus the number of forms in E containing the descendants of v. If we were to associate each node with the relevant maximum number of forms, N_v, then we could remove those nodes for which $N_v < N$. The nodes in the resulting subgraph of G are exactly the nodes that satisfy the threshold. Only the sink nodes in this subgraph could be valid outputs—these are candidate solutions, the actual solution being a candidate with the largest number of categories. This reasoning exploits two transitive relationships—the number of forms associated with a node and its descendants is at least as great as the number of forms associated with any of its descendants, and, the number of categories associated with a node is greater than the number of categories associated with any of its ancestors.

The virtual data environment, E, is scanned to produce a mapping of categorizations (graph nodes) to counts. The well-known *post order* depth first graph traversal algorithm (which can be found in many introductory texts; see e.g., [7]) will ensure that the children of a node, v, are visited before v. A post order traversal is conducted and the set of descendants of a node v is calculated as the union of the children of v and the children's descendant sets. N_v is calculated via summing the count of forms for v with those for the descendants of v. If $N_v \geq N$ then v satisfies the threshold. When a node that satisfies the threshold is visited, then the unvisited ancestors of this node are marked as visited (in a similar manner to the node addition algorithm described in Section 4.1.1). This prevents superfluous calculations for nodes that are not candidate solutions. Each candidate solution found is compared to the best solution found up to that point. Once the traversal is complete the best solution is returned. Pseudocode is contained in the Appendices.

[6]Note: here the data environment might be a subset of a larger virtual data environment, perhaps conditioning the analysis on geographical location or some other attribute.

Example:

Assume we had the following mapping for the harmonization graph in Figure 11. (Counts for harmonization codes are zero.)

```
mapping = {'C0.001': 21, 'C2.001': 36, 'C7.001': 7,
'C9.001': 30, 'C45.001': 10}
```

For $N \leq 21$ we can choose N forms with categorization C0.001. These harmonize to C0.001 which is the output for the analysis. In the algorithm above, C0.001 is the only candidate solution.

For $N = 22$ C0.001 is not a candidate solution as the number of forms associated with it and its descendants is less than N. The algorithm finds 3 candidate solutions, C9.001, C45.001, and C7.001. C45.001 is returned as it contains the larger number of levels.

As N gets larger the candidate solutions change. For $N = 29$ C7.001 is no longer a candidate solution as the number of forms associated with it and its descendants is 28. C45.001 is still returned as it is still a candidate solution. For $N = 32$ C45.001 is no longer a candidate solution and of the two candidate solutions (H4.001 and C9.001) the returned solution is C9.001.

Finding the descendant sets for the nodes in G is equivalent to computing the transitive closure of G. The transitive closure G^T of G is the graph that contains the same nodes as G and that has an edge from each node v to each of its descendants in G. Thus the descendants of v in G are the children of v in G^T. There are several algorithms for computing transitive closures (see [6] for discussion and further references). The above algorithm only computes the transitive closure for the subgraph induced by the candidate solutions and their descendants.

Computing the complete closure would allow solutions to be generated for an arbitrary range of thresholds. Pre-computed closures could be stored. In practice it has not been found that computing the closures leads to computational or space issues as the harmonization graphs are relatively small. If the data environment being analysed does not contain all the observed codes, then the graph could be pruned before conducting the analysis. The details will not be presented here as the computational benefits are negligible for the size of graphs that have been met in practice.

5.1 An Example with Real World Data

The data environment analysed consisted of 191 forms. These were extracted from the original KVMS spreadsheet implementation and relate to a variety of organizations such as universities, insurance companies, and supermarkets. A matching group was created containing the 83 forms relating to commercial organizations. The target was the 2001 Individual Sample of Anonymized Records (SAR).[7]

```
Variable                                      Code

Age                                           C45.001
Country of residence                          C4.001
Education                                     H15.001
Family type                                   None
General health over last 12 months            H16.001
House type                                    H8.001
Marital status                                H8.001
Migration                                     None
NS-SEC socio-economic classification          None
No.  of carers in household                   None
No.  of families in household                 None
No.  of household members in poor health      None
No.  of persons in household 65 or over       None
No.  of residents in household                C6.001
Workplace                                     C8.001
```

Setting the prevalence to zero generates the output above. The null results signify that the corresponding variables do not appear on any of the match forms.[8]

There is an option to conduct analyses so that inconsistencies in categorizations are highlighted. In this case the output for an inconsistently categorized variable would be a message indicating a (but possibly not the only) pair of forms with inconsistent categorizations. Although the data entry system is designed to limit the opportunities for creating such inconsistencies, this particular data environment was created via a 'warts and all' extraction of data from the original KVMS. Thus it can be useful to conduct the analysis twice—to generate outputs and to identify inconsistencies so that the corresponding outputs can be treated cautiously. The outputs above are under the option of not identifying inconsistencies.

[7]The 2001 Individal SAR is a 3% sample of data from individuals contained in the 2001 UK census.

[8]'None' is always returned in such cases, even though the threshold (at least 0 match forms) is strictly satisfied for all variables. Essentially, a prevalence of zero is treated as a prevalence equivalent to a single form.

As stated earlier in this paper, the standard output is almost identical to that in the original KVMS. The user would generally repeat an analysis for a range of prevalences to gain an impression of risk. However, it is possible to produce more informative outputs such as that suggested in the previous section—a plot of harmonization code sizes against prevalence. Non-standard outputs must be generated at the command line—using the built in Python shell.

The plot in Figure 12 shows code sizes against prevalence for the SAR matched against the commercial group. This was generated at the command line using the Matplotlib[9] third party library for the plot. Age has been excluded so that the results for the other variables can be seen more clearly. (All forms contain Age, and all but 2 forms have strictly more detailed categorizations than the SAR.) A little 'jitter' in the form of random noise has been added to the coordinates to reduce the problem of lines masking each other.

We can see from the plot that although we have a maximum code size of 16 for General Health the variable appears on very few forms. We can contrast this with Marital Status where the maximum code size is 8, but has code sizes of at least 3 for prevalences up to around 0.25. This suggests, perhaps, using the area under the curve as a general measure of risk/usefulness for matching.[10]

In this case the 'area under the curve' suggests Age, Workplace, Marital Status, and Education as key variables in descending order of importance. However, inspection of the categories for Workplace shows that it is mainly useful for distinguishing those that work at home, have no fixed place of work, or have a fixed place of work (other than home). Other categories such as 'Northern Ireland' clearly overlap categories such as 'at home'. This is an issue inherited from the original KVMS which will be corrected in the future. But a pragmatic approach would be to treat Workplace as only having 3 categories and demote it to a level of importance closer to that of Education.

Note that the issue with Workplace could not be highlighted during analysis because, although the categories were not mutually exclusive, the categorizations were consistent within the original KVMS. Another potential issue is that we might be dealing with a group (subpopulation) where the ability for a variable to distinguish individuals is reduced or eliminated. For instance, organizations might use standard forms containing Gender but only deal with males. This is an issue with having knowledge of categorizations, but no actual data. These points illustrate the importance of 'clerical review'. The outputs are suggestive of where the risk lies, but manual inspection/interpretation remains important.

5.2 An Overview of the Use of KVMS in Practice

To reiterate the point made in the introduction, the purpose of the KVMS output is to provide a crucial input into a disclosure risk assessment process—the specification

[9]See `http://matplotlib.org/`.

[10]Although these could be generated using the Python shell, they are not (as yet) standard outputs.

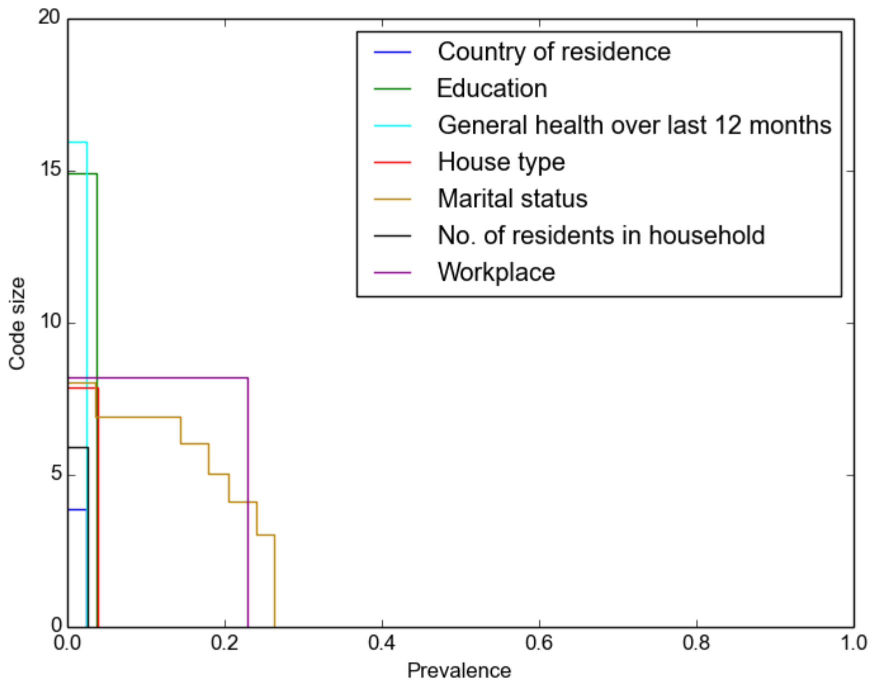

Figure 12: Code sizes against prevalence (Age not shown for clarity).

of the set of key variables that an intruder might feasibly use to attempt to reidentify data units within anonymised microdata. The output focuses on comparisons at the individual variable level. In practice, however, these are used to produce multi-variable, composite keys. The foregoing outlined the general approach to this, but in practice there are three ways in which the system is used:

(1) Specific data source cross match approach: Produce a harmonized set of key variables for a target dataset and a single specific attack dataset. This corresponds to an attack where the intruder has access to a single database and uses this to attempt to identify individuals in the target dataset.

(2) Typical data source cross match approach: Produce a harmonized set of key variables for a target dataset and a typical representation of what an intruder might have access to.

(3) Multiple data source cross match approach: Produce a harmonized set of key variables for a target dataset and several attack datasets. This corresponds to the situation where an intruder has access to several data sources each providing different variables.

The specific database cross match approach is fairly straightforward. One simply enters the case numbers of the target and attack datasets and the analysis produces a list of common variables and their harmonization codes. The harmonization codes with larger numbers of levels are generally assumed to be more useful for matching. If the relatively unusual situation described in Section 4.1 is suspected (many categories/but few categories in harmonization code), then the underlying categorizations can be examined.

Although easy to understand, this approach is generally over-specific. The intruder might use a different attack dataset to the one tested with a different set of variables. One could in principle run many analyses and produce many different key sets, but that would increase the complexity of the subsequent risk assessment task.

An alternative approach is to use the typical database cross match. Here we can consider a set of possible attack data sources simultaneously. We do this by specifying a required prevalence and generate maximal harmonizations over the subsets of data sources for which the prevalence is satisfied. If the threshold is set to 1, then only variables that are present in all selected data sources will be included in the outputs. If the threshold is set to 0 then the outputs are equivalent to generating the harmonization codes for each attack dataset and choosing the largest. Harmonization codes are a means of trying to capture the overall matching potential (against a target dataset) of a set of attack datasets with a variety of categorizations. Low prevalences will tend to produce bigger, more conservative keys; but if groups of forms are well defined this can be a way of producing future proofing key variable choices. If the grouping was 'datasets held by banks on their customers' and a single bank is collecting information on variable X, then the future proofing assumption is that all banks will eventually collect this information on their customers.

The multiple data source cross match approach reflects the concern that an intruder may increasingly be able to draw on multiple types of data source to construct their attack dataset. The process is the same as the typical database cross match case, but with forms drawn from multiple groupings.

There is a lot of information that is not captured by the analysis. It is an attempt to produce a general measure of risk from mutiple attack datasets containing different variables with different categorizations. Examination of harmonization graphs (and the frequencies generating during the analysis process) can provide more information regarding the possibilities for matching between attack data sources.

Note that a target dataset need not be a database. It could be a source of publicly available information such as a register or a representation of what people commonly know about their neighbours.

A combination of the above approaches was used in determining the attack scenarios for the assessment of disclosure risk associated with the microdata outputs from the 2011 UK census. Proposals for the specifications of the microdata were input as target data sources in a 300 data source repository. This led to the construction of a suite of scenarios which were used by the UK's Office for National Statistics in defining the 2011 census outputs.

6 Discussion

In this paper we have described the Key Variable Mapping approach to capturing critical information about a data environment. KVMS only requires that the user specify a categorization, and that the categorization is consistent with the corresponding partition graph. The system largely prevents inconsistencies by limiting how an existing partition graph can be edited. A general procedure for constructing partition graphs has been presented, and this provides an algorithm for constructing partition graphs automatically for a given set of categorizations if the relevant variable is on a numeric interval scale.

A representation of a categorization as a set partition over the set of sink nodes in a partition graph has also been presented. This demonstrates that harmonization of categorizations can be viewed as a sequence of set operations on set partitions. Disjoint set forests provide an attractive object model, where harmonization becomes a series of simple union operations on disjoint set forests.

Elliot et al. [4] show how the relationships between categorizations and harmonizations can be shown in a directed graph. However, Elliot et al. do not provide a formal definition of harmonization graphs and they are not automatically constructed or used in the early version of KVMS presented there. In this paper a formal definition has been provided along with an algorithm for constructing a harmonization graph from a set of categorizations. All the relevant harmonizations are generated automatically. Furthermore, it has been shown how the use of such a graph can be used to generate the outputs for the exemplar analysis in a particularly efficient manner.

The most innovative aspect of this work (with respect to the original KVMS) is the use of graphs and graph algorithms to generate outputs. In particular, the partition graph might be used to generate alternative outputs relating to key variables, and be useful for addressing other disclosure control problems. Partition graphs, as presented, fulfil the requirements of KVM—a partition graph encodes all logical matching possibilities based on alternative categorizations of a common variable. A partition graph does not encode information relating to the scale of a variable. This information might be relevant in other problem areas, such as recoding, where variables on an ordinal scale might be treated differently to those on a nominal scale. An ordinal scale would imply an ordering of the categories for each categorization. An ordering of the sink nodes of a partition graph would encode orderings for all other valid categorizations. So working with ordinal variables only requires a simple secondary data structure (a list of sink nodes). This could also be used to further restrict allowable edits and to add extra consistency checks. For example, a new category for Age with reachable sink nodes {'0-18', '21-infinity'} would be considered invalid as the sink nodes would not constitute a contiguous sublist of the ordered sink nodes. Alternative data structures might be more efficient for specific analytical purposes, but encoding the necessary additional information is straightforward. It is also possible to associate additional information with nodes and edges; such as node sensitivities, or edge weights to accommodate semantic uncertainty or conditional probabilities. It is also possible to construct multivariate partition graphs. These can become very large, and it is not clear that there exist use cases where these would be significantly more useful than a collection of univariate graphs. An exception would be where the graphs were augmented with additional information (probabilities, etc.) that could not be deterministically extended to higher dimensions without unreasonable assumptions (e.g., naive Bayes). These cases could largely be dealt with via secondary data structures. Otherwise, extending to the multivariate case only offers the explicit handling of zero measure joint categories (structural zeros).

The prospect of a multivariate partition graph suggests the possibility of multivariate harmonization graphs. The tensor product of the relevant univariate graphs would possess the essential properties of a harmonization graph, but would usually contain superfluous nodes—due to combinations of observed categories not being observed jointly. A multivariate graph would be a richer data structure than the underlying collection of univariate graphs, and would permit alternative outputs. But harmonization graphs are designed to efficiently generate the existing output, which is designed to allow the construction of composite keys on the basis of the usefulness (for linkage purposes) of individual keys. This paper is generally concerned with the construction of a queryable (meta-)database of which partition graphs form an integrable part. Harmonization graphs relate to a specific output, which is presented as an exemplar. Alternative outputs are not considered here— although they could involve secondary data structures such as multivariate harmonization graphs.

KVMS is primarily concerned with the possibilities for record linkage—the underlying databases are private. However, the encoding of matching possibilities could be useful for probabilistic record linkage between known databases (see e.g., [8]). Firstly, partition graphs can accommodate equal, but distinctly named categories. So they can be useful for data harmonization. They might also be useful for record linkage approaches that use similarity scores (e.g., [9]) where, for example, 'small' categories with a large degree of overlap might be assigned higher scores than 'large' categories with a small degree of overlap. Blocking (assigning pairs of records that do not match on a particular variable to the set of non matches) would be simple to implement using partition graphs (and the reachability criterion).

There are several potential areas for further research and ways of extending the current system. Additional work is required in order to produce a solid code base that could then be (at least partially) open sourced. This would allow other researchers to use the data environment, to extend it—or create new data environments—and to add further analytical outputs via scripts/plugins. An open source partition graph implementation is available at `https://github.com/DuncanSmith147/KVMS`.

References

[1] Duncan, G. T., Elliot, M. J. and Salazar-Gonzalez, J-J. (2011). *Statistical Confidentiality.* New York: Springer.

[2] Hundepool, A., Domingo-Ferrer, J., Franconi, L., Giessing, S., Schulte Nordholt E., Spicer, K. and deWolf, P.-P. (2012). *Statistical Disclosure Control.* Chichester, UK: Wiley.

[3] Elliot, M.J. and Dale, A. (1999) Scenarios of attack: The data intruder's perspective on statistical disclosure risk. *Netherlands Official Statistics 14*. 6–10.

[4] Elliot, M., Lomax, S., Mackey, E. and Purdam, K. (2010). Data environment analysis and the key variable mapping system. In J. Domingo-Ferrer and E. Magkos, eds, *PSD 2010*, vol. 6344 of LNCS. 138–147.

[5] Tarjan, R.E. (1975). Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225.

[6] Aho, A., Garey, M. and Ullman, J. (1972). The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137.

[7] Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C. (2001). *Introduction to Algorithms.* Second edition. MIT Press and McGraw-Hill.

[8] Fellegi, I.P. and Sunter, A.B. (1969) A theory for record linkage. *Journal of the American Statistical Association*, 64(238):1183–1210.

[9] Smith, D. and Shlomo, N. (2014). Privacy preserving record linkage. *Report for the Data without Boundaries project.* `http://www.ccsr.ac.uk/documents/Data_without_Boundaries_Report.pdf`.

Appendices

1 Pseudocode

Comments are preceded by a #.

1.1 Partition Graph Construction

```
def add_node(v, G):
    if v is not a node in G:
        add v to G
        for each sink node w in G:
            if $v \subset w$:
                u = $v \setminus w$
                add (w,v) to G
                add u to G
                add (w,u) to G
                break
            elif $w \subset v$:
                add (v,w) to G
            elif $v \cap w \neq \emptyset$:
                u = $v \cap w$
                x = $w \setminus v$
                add u to G
                add x to G
                add (v,u) to G
                add (w,u) to G
                add (w,x) to G
```

1.2 Harmonization

```
def harmonize(cats):
    # cats is a set of disjoint set forests
    harm = empty disjoint set forest
    cat = an arbitrary categorization from cats
    for node in cat:
        add node to harm
    for cat in cats:
        for node in cat:
            source = find(cat, node)
            union(harm, source, node)
    return harm
```

The correctness of the algorithm should be apparent. The *union* function is called on the harmonization for a pair of nodes if, and only if, the two nodes are in the same tree/category for some categorization. For any pair of nodes in the same tree/category for some categorization the equivalent of a *union* call on the harmonization is performed via separate calls with each node and the nodes' common source node.

1.3 Harmonization Graph Construction

```
def insert_node(u, G, is_observed):
    # is_observed is True if u is an observed categorization
    parents = an empty set
    children = an empty set
    visited = an empty set
    harmonizations = an empty set
    for x in the nodes of G:
        if x is not a member of visited:
            h = H(x, u)
            if h == x:
                add x to parents
                for n in the unvisited ancestors of x:
                    add n to visited
            elif h == u:
                add x to children
                for n in the unvisited descendants of x:
                    add n to visited
            else:
                if is_observed:
                    add h to harmonizations
    # remove non parents from candidate parents
    parents = parents \ visited
    # remove non children from candidate children
    children = children \ visited
    # now u can be added and edges removed and added
    add the node u to G
    for p in parents:
        # remove any edges between p and nodes in children
        for each child c of p:
            if c in children
                remove the edge (p,c) from G
        add the edge (p,u) to G
    for c in children:
        add the edge (u,c) to G
    for h in harmonizations:
        insert_node(h, G, False)

def harmonization_graph(cats):
    # cats is a set of observed categorizations
    # i.e.  (disjoint set forests)
    G = an empty directed acyclic graph
    for u in cats:
        insert_node(u, G, True)
    return G
```

1.4 Analysis

Note that Python allows the nesting of functions, and enclosed functions have access to objects in the enclosing function's scope. Assignment within an enclosed function creates a local name, so the candidate solution is contained in a list which can simply be mutated.

```
def analyse(G, mapping, N): # N is calculated as described
above
    # initialize containers
    visited = empty set
    descendants = empty mapping
    result = [None]

    # define nested functions
    def dfs(v):
        if v is not a member of visited:
            for c in the children of v:
                if c is not a member of visited:
                    dfs(c) # might add v to visited
            if v is not a member of visited:
                process(v)
            add v to visited

    def process(v):
        desc = empty set
        for c in the children of v:
            add c to desc
            desc = desc ∪ descendants[c]
        descendants[v] = desc
        Nv = mapping[v]
        for d in desc:
            Nv = Nv + mapping[d]
        if Nv ≥ N: # v is a candidate solution
            if result[0] is None:
                result[0] = v
            else:
                if v has more categories than result[0]:
                    result[0] = v
            # visit ancestors
            for n in the unvisited ancestors of v:
                add n to visited

    # run algorithm
    for v in the nodes of G:
        dfs(v)
    return result[0]
```