Table Summarization with the Help of Domain Lattices

Table Summarization with the Help of Domain Lattices *

K. Selçuk Candan, Huiping Cao, Yan Qi
Arizona State Univ.
Tempe, AZ 85283, USA
{candan,hcao11,yan.qi}@asu.edu

Maria Luisa Sapino
Univ. di Torino
Torino, Italy
mlsapino@di.unito.it

ABSTRACT

Table summarization is needed in various scenarios where it is hard to display a large data set. For instance, a scientist who is exploring possible databases for further analysis may want to see concise yet informative summaries of large tables. Similarly, small devices, such as PDAs, cannot effectively present a large table of results with their small screens. In this paper, we first argue that table summarization can benefit from knowledge about acceptable value clustering alternatives for clustering the values in the database. We formulate the problem of table summarization with the help of domain lattices. We then provide a framework to express alternative clustering strategies and to account for various utility measures (such as information loss) in assessing different summarization alternatives. Based on this interpretation, we introduce three preference criteria, max-min-util, pareto-util, and max-sum-util for the problem of table summarization. To tackle with the inherent complexity, we rely on the properties of the fuzzy interpretation to further develop a novel ranked set cover based evaluation mechanism (RSC). Experimental evaluations showed that RSC improves both execution times and the summary qualities, by pruning the search space more effectively than the existing solutions.

1. INTRODUCTION

Summarization of large data tables is required in many scenarios where it is hard to display complete data sets. For instance, small devices, such as PDAs, cannot effectively present a large table of results with their small screens. TabSum was introduced in [17, 32] as a flexible and dynamic approach to create and maintain table summarization for mobile commerce applications. Table summarization is also useful in various other scenarios, where it is hard or highly-impractical to visualize large data sets. Consider, for example, a scientist exploring *the Digital Archaeologial Record (tDAR/FICSR)* [26, 27] (a digital library which archives and provides access to a large number of (and diverse) data sets, collected by different researchers within the context of different projects and deposited to *tDAR* for sharing. When this scientist poses a search query through *FICSR*, her query might match many potentially relevant databases and data tables. For this scientist to be able to explore the multitude of candidate data resources identified by *tDAR* as quickly and effectively as possible, novel data reduction techniques (such as table summarization) are needed.

In this paper, we first note that table summarization process can benefit significantly from knowledge about acceptable value clustering alternatives. For example, Table 1(a) shows a data table consisting of 4 rows. If the user is interested in summarizing the table based on the attribute pair, $\langle Age, Location\rangle$ in such a way that, each row corresponds to at least 2 rows in the original table (i.e, the table size is

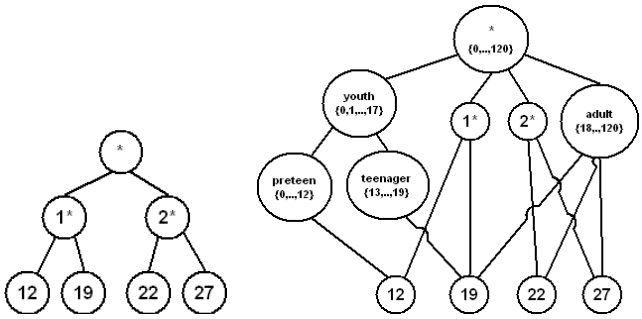
ID	Name	Age	Location
A	V	12	loc1
B	W	19	loc1
C	Y	22	loc2
D	Z	27	loc2

(a) data table

Count	**Age**	**Location**
2	**1***	**loc1**
2	**2***	**loc2**

(b) data table after 2-summarization on $\langle Age, Location\rangle$

Table 1: (a) A database and (b) a 2-summarized version on the $\langle Age, Location\rangle$ attribute pairs



(a) Hierarchy (b) Lattice

Figure 1: (a) Hierarchy- and (b) Lattice-based value clustering graphs for the attribute *age*

*Authors are listed in the alphabetical order; Supported by NSF Grant "Archaeological Data Integration for the Study of Long-Term Human and Social Dynamics" (0624341)

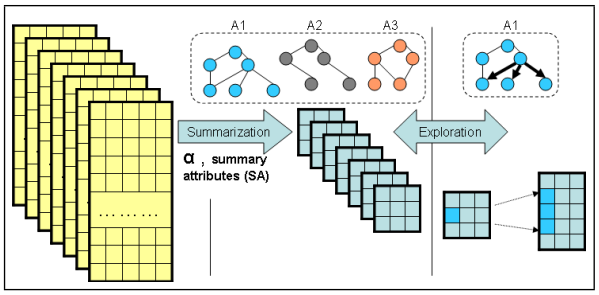


Figure 2: Lattice based α-summarization helps reduce the table size for quick exploration. Lattices can also help the user further explore the summary as in OLAP

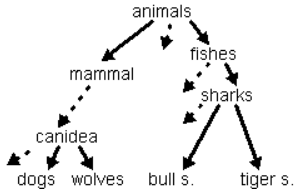


Figure 3: A partial taxonomy of animals

at least halved – we refer to this as *2-summarization*), this can be best achieved if additional domain knowledge, which can help choose how to cluster the values, is available. For example, given the value hierarchy in Figure 1(a), one can 2-summarize this table as shown in Table 1(b).

Value hierarchies (such as the one in Figure 1(a)) are commonly used for user-driven data exploration *within* large data sets. For example, OLAP operators (such as *drill-down*, navigating from less specific to more detailed data by stepping down on a given hierarchy, and *roll-up*, which performs aggregation on by climbing up a hierarchy of concepts) commonly assume value hierarchies are given [6]. We note that, when summaries are created leveraging on an underlying value clustering hierarchy, this hierarchy can then be used for helping the user refine the relevant portions of the summary results by moving up-down on the hierarchy as in OLAP's *drill-down* and *roll-up* (Figure 2).

More recently, there is an increasing awareness of the fact that, in many applications (even in more traditional ones, such as OLAP [28]), there are generally multiple acceptable value clustering *alternatives*. For example, the value lattice shown in Figure 1(b) can provide more clustering alternatives than the simpler hierarchy in Figure 1(a), enabling the applications to be more precise and informative.

When there are alternative clustering strategies for the values in a database, however, not all alternatives might be equally desirable. Let us reconsider Figure 1(b):

- Clustering the value "12" under "*preteen*" (which corresponds to values $0 \leq age < 13$) versus "*1**" (which corresponds to values $10 \leq age \leq 19$) maybe less desirable because the range of possible values (i.e, the precision lost by the replacement) in the former case is larger than in the latter.

- If the nodes of the lattice do not represent concrete sets, but concepts in an ontology or a *concept* taxonomy, then the semantic difference between terms can be estimated using *structural* and/or *information-based* approaches:

  In structure-based methods [23, 25], distance between two nodes is measured as the sum of the *distance* weights of edges between them: the weight of an edge between a parent concept and a child is measured based on the structural clues available in the lattice, such as depth (the deeper the edge in the lattice, the less information loss associated to the edge; e.g., in Figure 3, *dogs* are more related to *wolves* than *mammals* are to *fishes*) and local density of the concepts (the denser the lattice, the smaller the semantic distance between a concepts in that neighborhood of the lattice).

  Information-based methods leverage available corpora to measure potential information loss. For example, knowledge about term-occurrence frequencies may help measure information-loss in an information theoretic manner [24] (information content is measured as the negative logarithm of the probability of encountering an instance of the concept in the corpus).

Therefore, when creating an α-summary (where each row corresponds to at least α rows in the original table, as in Table 1(b)) by clustering data values according to the given lattice, we need to account for the loss of precision.

In this paper, we first formulate the problem of α-summarization under weighted value clusterings. Instead of solving a special case of the problem (e.g., assuming a hierarchy instead of a lattice or assuming a specific information loss measure), we introduce a general *utility-weighted value clustering lattice* representation to capture alternatives and choices over these alternatives, introduce the concept of utility of a clustering strategy, and formalize the problem of α-summarization (Section 3). Relying on this, we also introduce three solution preference criteria, `max-min-util`, `pareto-util`, and `max-sum-util` for the problem of α-summarization. We then rephrase the summarization problem into a set cover problem and present a novel *ranked set cover* algorithm that is able to prune the search space significantly to reduce the execution time for α-summarization task (Section 5). In Section 6, we evaluate the proposed approach and show that the proposed ranked set cover algorithm improves both execution time and result quality.

2. RELATED WORK

In [2], Alfred *et al.* present an approach to data summarization by aggregating data over a relational dataset. A good survey of the database summarization can be found in [29], where an online linguistic table summarization system, *SaintEtiQ*, is also presented. *SaintEtiQ* [29, 30] computes and incrementally maintains a hierarchically arranged set of summaries of the input table. In such a hierarchical structure, any non-leaf summary generalizes the content of its children nodes. TabSum [17] summarizes tables through row and column reduction. To reduce the number of rows, it first partitions the original table into groups based on one or more attribute values of the table, and then collapses each group of rows into a single row relying on the available metadata, such as the concept hierarchy. For column reduction, it simplifies the value representation and/or merges multiple columns into one column. [32] discusses a related approach for refinement of table summaries. Neither of these

approaches, however, considers the impact of the imprecision of metadata and the information loss during summarization.

Data compression techniques, like Huffman or Lempel-Ziv, can also be used to reduce the size of the table. A database compression technique is designed in [20] based on vector quantization. Buchsbaum *et al.* [4, 5] address the problem of compressing massive tables through a partition-training paradigm. These methods are not directly applicable, since the compressed tables are not human readable. Histograms can be also exploited to summarize information into a compressed structure. Following this idea, Buccafurri *et al.* [3] introduce a quad-tree based partition schema for summarizing two-dimensional data. Range queries can be *estimated* over the quad-tree since the summarization is a lossy compression. Leveraging the quad-tree structure, [8, 9] proposes approaches to processing OLAP operations over the summary. [9] first generates a 2-dimensional OLAP view from the input multidimensional data and then compresses the 2-dimensional OLAP view by means of an extended quad-tree structure. In fact, any multidimensional clustering algorithm can be used to summarize a table. Such methods, however, cannot take into account specific domain knowledge (e.g. "what are acceptable summarizations, how do they rank?") that hierarchies and lattices would provide.

OLAP operations, *drill-down* and *roll -up*, which help users navigate between more general and more specific views of the data with the help of given value hierarchies, are also related to table summarization. The concept of imprecision in OLAP dimensions is discussed in [22]. In that framework, a fact (e.g., a tuple in the table) with imprecise data is associated with dimension values of coarser granularities, resulting in the dimensional imprecision. In schema design for traditional relational-OLAP systems, issues around heterogeneous dimensions have been discussed in [14, 13]. A dimension is called heterogeneous if two members in a given category are allowed to have ancestors in different categories. Given a heterogeneous dimension, an aggregate view for a category may not be correctly derived from views for its sub-categories (a summarizability [1] problem). In [28], we supported OLAP operations over imperfectly integrated taxonomies. We proposed a re-classification strategy which eliminates conflicts by introducing minimal imprecision. This approach is complementary to the work presented here: the obtained navigable taxonomy can be taken as the input for table summarization.

The table summarization task is also related to k-anonymization problem, introduced as a technique against linkage attacks on private data [31, 16]. The k-anonymization approach eliminates possibility of such attacks by ensuring that, in the disseminated table, each value combination of attributes are matched to k others. To achieve this, k-anonymization techniques rely on a-priori knowledge about acceptable value generalizations. Cell generalization schemes [1] treat each cell in the data table independently. Thus, different cells for the same attribute (even if they have the same values) may be generalized in a different way. This provides significant flexibility in anonymization, while the problem remains extremely hard (NP-hard [18]) and only approximation algorithms are applicable under realistic usage scenarios [1]. Attribute generalization schemes [31, 16] treat all values for a given attribute collectively; i.e., all values are generalized using the same unique domain generalization strategy. While the problem remains NP-hard [18] (in the number of attributes), this approach saves significant amount of time in processing and may eliminate the need for using approximation solutions, since it does not need to consider the individual values. Most of these schemes, such as Samarati's original algorithm [31], however, rely on the fact that, for a given attribute, applicable generalizations are in total order and that each generalization step in this total order have the same cost. [31] leverages this to develop a binary search scheme to achieve savings in time. [16] relies on the same observation to develop an algorithm which achieves attribute-based k-anonymization one attribute at a time, while pruning unproductive attribute generalization strategies.

Unlike these solutions to k-anonymization, the summarization algorithm presented in the paper does not rely on any constraints imposed on the shape and weights of the value clustering lattice and, instead, owes its efficiency to a novel ranked set cover based evaluation mechanism (RSC) that is able to prune the search space effectively.

[1]Summarizability is a notation to denote the conditions under which a value or object can be summarized correctly from a more detailed value or object, based on the given summarization rules (e.g., value mappings, value lattices, etc.). It has been defined in [33] and used in the context of OLAP.

3. PROBLEM FORMULATION

Let us consider a data table, T, and a set, SA, of summarization-attributes. Roughly speaking, our purpose is to find another relation T' which clusters the values in T such that T' satisfies the α-summarization property (where each row in T' clusters at least α rows in T) with respect to the summarization-attributes. In what follows, we first formalize the concept of *weighted clustering lattices.* Based on this, we then specify the problem of α-summarization over weighted clustering lattices.

3.1 Value Clustering Lattices

We model a value clustering lattice as an acyclic graph $M(V, E)$ where V encodes values and clustering-identifiers (e.g., high-level concepts or cluster labels, such as "1*" in Figure 1) and E contains acceptable value clustering relationships.

Definition 3.1 (Value Clustering Lattice). *A value clustering lattice L is a directed acyclic graph $M(V, E)$:*

- *$v = (id : value) \in V$ where $v.id$ is the node id in the lattice and $v.value$ is either a value in the database or a value clustering encoded by the node.*
- *$e = v_i \rightarrow v_j \in E$ is a directed edge denoting that the value encoded by the node v_i can be clustered under the value encoded by the node v_j.*

  Those nodes in V which correspond to the attribute values that are originally in the database do not have any incoming edges in E. There is only one root (i.e., node without any outgoing edge) in the lattice.

The value clustering lattice describes a partial order that reflects acceptable clustering strategies for each value (e.g., Figures 1(a) and(b) and Figure 3).

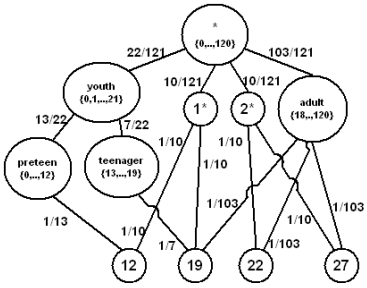


Figure 4: A weighted lattice for the attribute *age*

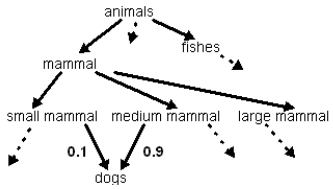


Figure 5: An imprecise taxonomy of animals: there is a disagreement among users of the taxonomy as to whether *dogs* should be classified under *small* or *medium mammals*

3.2 Value Clustering

Given an attribute value in the data table T and a weighted value clustering lattice corresponding to that attribute, we can define alternative clusterings as paths on the corresponding lattice:

DEFINITION 3.2 (VALUE CLUSTERING). *Given a weighted value clustering lattice L, a lattice node v_j is a clustering of a lattice node v_i, denoted by $v_i \preceq v_j$, if $\exists path\ p = v_i \rightsquigarrow v_j$ in L. We also say that v_j covers v_i. Note that p may also be empty.*

Based on this, in the following, we formalize the concept of *utility* of value clusterings.

3.2.1 Utility of a Value Clustering Strategy

As described earlier, the utility value of a clustering step is domain specific and may represent different reasons one may prefer one clustering alternative over the other. A high utility value indicates a clustering step with low information loss and high precision (utility value 1.0 indicates zero information loss and uncertainty due to clustering; thus is excluded from the model). Any utility below 1.0 denotes a certain degree of knowledge relaxation. For example, under one interpretation, the edge in Figure 1(b) from "12" to "preteen" would have utility 1/13 ("12" is one out of 13 values the label "preteen" captures). Similarly, the edge from "12" to "1*" has utility 1/10 (Figure 4).

Given the utilities of clustering edges in the lattice, we can also compute the utility of a value clustering strategy, $util(v_i \preceq v_j)$, between v_i and any of its ancestors v_j, by considering all paths between v_i and v_j. The value clustering utility function used for combining utilities of alternative ways v_i can be re-written into v_j may differ depending on the semantics of the lattice. In the case when precision is defined in terms of set membership, all path utilities are identical, thus combination function would simply return the utility of any path. For example, in Figure 4, all paths from value, 19, to the clustering value, $*$, would return the same path utility, 1/121, since 19 is one out of 121 values covered by $*$.

If different paths between v_i and v_j have different utilities, on the other hand, these may need to be combined using other combination functions. For example, in Figure 5, there are two different paths between "*dogs*" and "*mammals*" representing different views of the scientists using the taxonomy. In this case, replacement of *dogs* with *mammals* needs to consider all interpretations and combine the corresponding utilities. Candidate combination functions include, *minimum*, *maximum*, and *sum*. For example, in Figure 5, *sum* may be suitable: independently of whether "*dogs*" are *small* or *medium mammals*, they are *mammals*. Thus, the imprecision at the lower levels of the lattice should not matter when considering the utility of replacing "*dogs*" with "*mammals*.

3.2.2 Computing the Value Clustering Utilities

Given a value clustering lattice, $\mathcal{L}$, which provides a partial order of value clusterings, there may be exponentially many paths between a given pair of values, v_i and v_j. Thus, computing the value clustering utilities on the given lattice may be costly, if performed naively. We note, however, that in many practical cases, value clustering utilities between all pairs of values in the value hierarchy can be computed in polynomial time. Especially in the special case where any replacement path between a given pair of values return the same precision loss, the cost of the computation is simply quadratic in the number of nodes.

The challenge is dealing with the exponential growth in the number of summarization-tuple clustering strategies when the number of summarization-attributes increases.

3.3 Summarization-Tuple Clustering

So far we have focused on the clustering of the value of single summarization-attribute; i.e., the case where the size of the set, SA, of summarization-attributes is one and there is only one clustering domain. However, in general, the size of the summarization-attribute set of a data table can be greater than one. Thus, we first need to define clusterings involving non-singleton summarization-attributes:

DEFINITION 3.3 (SUMMARIZATION-TUPLE CLUSTERING). *Let t be a tuple on attributes $SA = \{Q_1, \cdots, Q_q\}$. t' is said to be a clustering of the summarization-tuple (SA-tuple), t, (on attributes SA) iff $\forall i \in [1, q]$*

- $t'[Q_i] = t[Q_i]$ *, or*
- $\exists path\ p_i = t[Q_i] \rightsquigarrow t'[Q_i]$ *in the corresponding clustering lattice L_i.*

We use $t \preceq t'$ as shorthand.

3.3.1 Utility of a SA-Tuple Clustering Strategy

Given the above definition, we can define the utility of a summarization-tuple clustering strategy:

DEFINITION 3.4 (UTILITY OF A SA-TUPLE CLUSTERING.). *Let t and t' be two tuples on attributes $SA = \{Q_1, \cdots, Q_q\}$, such that $t \preceq t'$. Then the utility of the corresponding clustering strategy is defined through a monotonic combination function, $\otimes$, of the utilities of the clustering along each*

individual summarization-attribute:

$$util(t \preceq t') = \bigotimes_{1 \le i \le q} util_i,$$

where

- $util_i = 1$, *if* $t'[Q_i] = t[Q_i]$
- $util_i = util(t[Q_i] \preceq t'[Q_i])$, *otherwise.*

When $q=1$ (i.e., there is only one summarization-attribute), $util(t \preceq t')$ is simply the utility of the corresponding single attribute value clustering strategy. In general, $\otimes$ can be any monotonically increasing function; for example, minimum, multiplication, or summation.

3.4 α-Summarization Problem

Having defined *utilities* of tuple clusterings, we can now define the problem of α-summarization with lattices:

DEFINITION 3.5 (α-SUMMARIZED CLUSTERING). *Given two data tables T and T' (with the same schema), and the summarization-attribute set SA, T' is said to be a α-summarized clustering of T ($T \preceq_\alpha T'$ for short) iff*

- $\forall t' \in T'[SA]$, $|\{t'' | t'' \in T'[SA] \wedge t''[SA] = t'\}| \ge \alpha$ *(i.e., t' appears at least α times in T'),*
- *there is a one-to-one and onto*[2] *mapping, μ, from $T'[SA]$ to $T[SA]$, such that*
  - $\forall t' \in T'[SA]$, $\mu(t') \preceq t'$.

Here, $T[SA]$ and $T'[SA]$ are projections of the data tables T and T' on summarization-attributes. $T[SA]$ and $T'[SA]$ are multi-sets; i.e., they may contain multiple instances of the same summarization-tuple.

Given a data table T, there can be multiple such α-summarized clustering alternatives of T. We thus define the utility of a α-summarized clustering as follows:

DEFINITION 3.6 (TABLE CLUSTERING UTILITY). *Let T be a table and T' be its α-summarized clustering (i.e., $T \preceq_\alpha T'$). The utility of T' is defined using a monotonic merge function, $\uplus$, as follows:*

$$util(T') = \biguplus_{t' \in T'[SA]} util(\mu(t') \preceq t').$$

Since there can be more than one alternative clustering, our goal is to find one that maximizes the utilities of the underlying clustering.

PROBLEM 1 (UTILITY MAXIMIZING α-SUMMARIZATION). *Given T, SA, $\mathcal{L}$, α, $\otimes$, and $\uplus$, where*

- *T is a data table with n tuples to be published,*
- *$SA = \{Q_1, \cdots, Q_q\}$ is the set of summarization-attributes,*
- *$\mathcal{L} = \{L_1, \cdots, L_q\}$ is the set of weighted clustering lattices for the attributes in SA,*
- *α is the summarization parameter, and*
- *$\otimes$ and $\uplus$ are the appropriate utility merge functions,*

find another data table T' (and the corresponding mapping μ from $T'[SA]$ to $T[SA]$) which is a α-summarized clustering of T, such that there exists no other α-summarized data table T'' (and the corresponding mapping μ_2), such that T'' is preferable *over T'.*

[2]It is possible to relax the *onto* requirement and allow for some limited number of tuples which are hard to cluster with others, due to outlier values, to remain unsummarized. In this paper, we do not explicitly consider this situation, though it is possible to account for these tuples by representing them through a special form of clustering alternative, with 0 utility value.

3.5 α-Summarization Preference Criteria

In this paper, we consider the following pessimistic and holistic preference criteria.

DEFINITION 3.7 (MAX-MIN-UTIL). *Let T be a data table and let T' and T'' be two α-summarized clusterings. T'' is* `max-min-util` *preferable over T' iff*

$$\left(\min_{t' \in T'} util(\mu_1(t') \preceq t')\right) < \left(\min_{t'' \in T''} util(\mu_2(t'') \preceq t'')\right).$$

`Max-min-util` criterion is a *pessimistic* standard that guarantees the worst case summarization is best by maximizing the lowest of the tuple utilities. In other words, $\uplus$ is the *minimum* function.

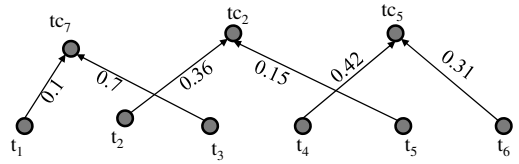
DEFINITION 3.8 (MAX-SUM-UTIL). *Let T be a data table and let T' and T'' be two α-summarized clusterings. T'' is* max-sum-util preferable *over T' iff*

$$\left(\sum_{t' \in T'} util(\mu(t') \preceq t')\right) < \left(\sum_{t'' \in T''} util(\mu_2(t'') \preceq t'')\right).$$

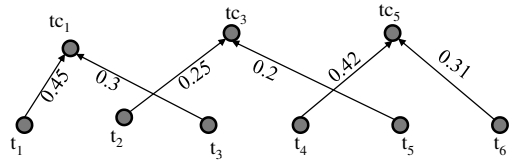
`Max-sum-util` criterion is a *holistic* standard since it considers the summarization as a whole and requires that the total utility in the resulting clustering is maximized. In other words, $\uplus$ is *summation.*

EXAMPLE 3.1. *The following example illustrates how the Max-sum-util and Max-min-util preference criteria work for the same summarization requirement. Suppose we have six tuples (t_1 to t_6) and want to get a 2-summarization for it. I.e., each tuple clustering (tc) in the summarization table would cluster at least two tuples. Besides this summarization requirement, we also know (1) the clusterings that each tuple can be summarized to and (2) the summarization utilities from each tuple to its tuple clusterings. (Section 4 shows how to compute such information from the attribute lattices.) For illustration purpose, we just display the tuple clusterings that can summarize at least two tuples since all the other tuple clustering (covering less than two tuples) cannot be in the solution. E.g., t_2 can be summarized to tc_2 and tc_3 with summarization utilities 0.36 and 0.25 respectively; tc_1 can cluster t_1 and t_3, then it is shown in the tuple clustering list of t_1 and t_3.*

t_1	t_2	t_3	t_4	t_5	t_6
$tc_1(0.45)$	$tc_2(0.36)$	$tc_7(0.7)$	$tc_5(0.42)$	$tc_3(0.2)$	$tc_5(0.31)$
tc_7 (0.1)	$tc_3(0.25)$	$tc_1(0.3)$	$tc_6(0.3)$	$tc_2(0.15)$	$tc_6(0.3)$
$\cdots$	$\cdots$	$tc_4(0.2)$	$\cdots$	$\cdots$	$tc_4(0.2)$
		$\cdots$			$\cdots$



(a) Max-sum-util summarization (sum utility 2.04)



(b) Max-min-util summarization (min utility 0.2)

Figure 6: Summarization example with different summarization criteria (a) Max-sum-util creteria (b) Max-min-util

The summarization satisfying the Max-sum-util criterion is shown in Figure 6(a).

In this 2-summarization, the sum utility to cluster all tuples to their related clusterings is 2.04 (0.1+0.7+0.36 + 0.15 + 0.42 + 0.31). All the other possible summarization strategies have sum utility values less than 2.04. So, this result satisfies the `Max-sum-util` *criterion.*

However, this summarization is not the optimal solution in using the `Max-min-util` *criterion since there exist other summarizations with bigger min utilities than the min utility of this summarization (its min utility 0.1). In fact, one summarization having the* `Max-min-util` *is shown in Figure 6(b). In this summarization, the min utility is 0.2 (the minimum utility among the tuple generalizations). While all the other possible summarization strategies have min utility values less than 0.2. So, this result satisfies the* `Max-min-util` *criterion.*

Theoretically, there are no specific rules to prefer one criterion to the other. It is the user's decision to choose the preference criterion according to their application requirement. If a holistic summarization is wanted, then the max-sum-util should be used. Otherwise, the max-min-util would be applied to get a pessimistic summarization.

4. EXHAUSTIVE FORMULATIONS

We first present exhaustive formulations to the α-summarization problem with weighted lattices. Despite being inefficient, this outlines the overall framework, enables us to study sources for the complexity of the problem, and highlights opportunities for ranked executions. In Section 5, then, we present a novel ranked algorithm that is able to prune the search space significantly to reduce the time for identifying solutions.

4.1 Outline of the Exhaustive Approach

In this subsection, we outline the four steps of an exhaustive approach to the problem. Let us be given a data table, T, with n tuples, summarization-attributes, $SA = \{Q_1, \cdots, Q_q\}$, the set, $\mathcal{L} = \{L_1, \cdots, L_q\}$, of clustering lattices for the attributes in SA, and the summarization parameter, α.

Step 1. Computation of the value clusterings: The exhaustive algorithm first considers each value val_i for every attribute $Q_j \in SA$. Let v_i be the lattice node for val_i in the clustering lattice L_j. The clusterings are sorted according to their utility values. Since this step requires at most $|SA|$ all pairs computations, for most practical situations, it is polynomial in time complexity.

Step 2. Computation of the tuple clusterings: For each tuple t in $T[SA]$, the basic algorithm computes the corresponding tuple clusterings and clustering utilities by combining all value clusterings for values $t[Q_1], \cdots, t[Q_q]$. Since this step computes the tuple clusterings by combining all possible value clusterings, its worst case space and time complexity is $O(\sum_{i=1}^{n} \prod_{j=1}^{|SA|} gen\_num(t_i, Q_j))$, where $gen\_num(t_i, Q_j)$ represents the number of clusterings of value t_i for summarization-attribute Q_j. The complexity term can be approximated as $O(n \cdot vg^{|SA|})$, where vg is the average number of clusterings of the attribute values: the complexity is linear in the number of tuples n, but is exponential in $|SA|$.

Step 3. Preliminary Filtering: In its third step, for each tuple clustering obtained in Step 2, the exhaustive algorithm computes the number of tuples that can be generalized to it. Those tuple clusterings covering less than α tuples are simply removed, as they cannot be involved in any α-summarized result. To identify tuple clusterings *covering* at least α tuples, the basic algorithm maintains a hash table to keep track of the distinct tuple clusterings. Each item in the hash table is a $\langle tc, cnt \rangle$ pair, where cnt is the number of tuples that tc can cover. While scanning the list of possible clusterings for each tuple, if the clustering has already been recorded (for another tuple), the corresponding count in the hash table is incremented. The space complexity of this step is $O(dist\_gen\_count)$ where $dist\_gen\_count$ is the total number of distinct tuple clusterings (which is in the worst case exponential in $|SA|$). The time complexity is $O(n \cdot vg^{|SA|})$; i.e., the time used to scan the tuple clusterings.

Step 4. Once the above three preparatory steps are completed, in its final step the algorithm computes the best α-summarized clustering.

4.2 Set-Covering Formulation

The set-covering problem can be stated as follows: given a finite set S, and a collection, C, of subsets of S, one aims to find a subset, C', of C, where every element of S belongs to at least one subset in C' and where some objective criterion (such as the cardinality of the set cover or the sum of cardinalities of the subsets in the set cover) is optimized.

We note that, the requirements of the Step 4 of the exhaustive algorithm can be represented as a set covering problem: by treating (a) each tuple clustering tc_i as a set, (b) each tuple t_i as an element, and (c) every clustering relation $t_i \preceq tc_j$ as the set relationship $t_i \in tc_j$, we can formulate the α-summarization problem with weighted clustering hierarchies as a k-set-cover problem (for $k = \alpha$).

While the k-set-cover problem is known to be NP-complete, there are various approximation algorithms [7, 21]. Thus the problem under `max-sum-util` preference criterion can be solved through a k-set-cover approximation algorithm. In fact, in the next section, we will show that it is possible to leverage set-covering based formulation of the problem within an incremental framework which produces results more efficiently *and* effectively than a *naive* set-covering formulation.

Input : Sorted tuple clustering streams ($TCStr_i$s) for each tuple in T, and summarization parameter α.
Any tuple clustering not covering at least α tuples are dropped from the stream.
Output : TC' such that clustering rules are satisfied

1. $TC_{combined} = \emptyset$;
2. $PQ = \emptyset$; /*priority queue of tuple clusterings at the boundary*/
3. Get the first tuple clusterings from each $TCStr_i$;
4. Insert into $TC_{combined}$;
5. initPQ(PQ);
6. repeat
   (a) if $TC_{combined}$ covers all tuples in T
      i. TC'=construct_and_solve($TC_{combined}, \alpha$);
      ii. if $TC' \neq \emptyset$,
         return (TC', extract_mapping(TC')) and **stop**;
   (b) $tc_{nextbest}$ = getFromPQ() /*get from the priority queue the tuple clustering with the next best utility value*/
   (c) $TC_{combined} = TC_{combined} \cup \{tc_{nextbest}\}$

   until $tc_{nextbest} = null$

initPQ(PQ)

1. Get the next tuple clusterings from each $TCStr_i$;
2. Insert them into PQ;

getFromPQ(PQ)

1. tc_h =dequeue(PQ);
2. Let $TCStr_i$ be the stream from which tc_h was originally pulled from
3. Get the next tuple clustering, tc_j from $TCStr_i$
4. enqueue(PQ, tc_j)
5. **return** (tc_h)

Figure 7: Pseudo-code of the ranked algorithm

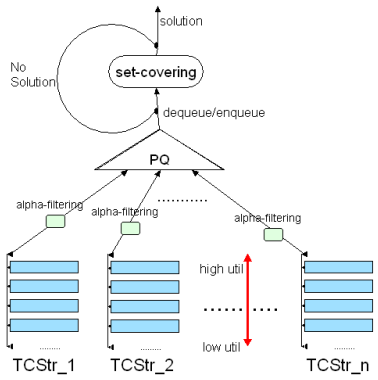


Figure 8: Outline of the ranked set cover algorithm

5. RANKED PROCESSING

In the previous section, we have seen that exhaustive solutions to the proposed problem has three sources of complexity: (a) the number of summarization-attributes increases the number, m, of tuple clusterings exponentially, (b) the set-covering formulation of the problem has an exponentially increasing cost as a function of m. Also, (c) the `max-min-util` preference criterion leads to a non-linear (i.e., more expensive) problem formulation. In this section, we first present a generic ranked algorithm which significantly reduces the number, m, of tuple clusterings which need to be considered. We then develop a novel ranked set-covering algorithm that performs efficiently and more effectively than non-ranked set-covering solutions.

5.1 Ranked Algorithm

The algorithm, outlined in Figures 7 and 8, takes as input (a) a stream of clusterings for each tuple in $T[SA]$ and (b) the summarization parameter α. Tuple clustering streams are taken as input streams in decreasing order according to their *util* values. The algorithm merges the n sorted tuple clustering streams by considering first those tuple clusterings with the highest *util* values (Step 3). Any tuple clustering that is not covering at least α input tuples are eliminated from consideration. To identify tuple clusterings *covering* at least α tuples, we maintain a hash table of clusterings, which keeps the count of the input streams in which a given clustering is seen. Once we obtain a proper set of clusterings, $TC_{combined}$, which covers all the tuples in the data table, we convert this set into set-covering problem (Step 6(a)i). If this step returns a solution, then the ranked algorithm stops (Step 6(a)ii). Otherwise, it continues incrementally considering other tuple clusterings in their utility order. The algorithm uses a priority queue to keep track of the next best clusterings of the tuples in the database (Steps 5 and 6b). This process continues until either all tuple clusterings have been checked or a solution is found.

EXAMPLE 5.1. *In this example, let $\alpha = 2$, we illustrate the ranked algorithm using the tuples and tuple clusterings in Example 3.1.*

- *Initially (after Step 3 and 4), $TC_{combined} = \{tc_1, tc_2, tc_3, tc_5, tc_7\}$.*
- *The program gets the tuple clustering with the next best utility value (0.3) from the priority queue (Step 6a), we get tc_1 for t_3 (with utility 0.3), tc_6 for t_4 (with utility 0.3), and tc_6 for t_6 (with utility 0.3). $TC_{combined} = \{tc_1, tc_2, tc_3, tc_5, tc_6, tc_7\}$. So far, the tuple clusterings that can summarize more than 2 tuples are tc_1 (for t_1 and t_3), tc_5 (for t_4 and t_6), and tc_6 (for t_4 and t_6). Obviously, tuples t_2 and t_5 cannot be covered yet.*
- *We continue to fetch the next best utility value (0.25), and get tc_3 for t_2. Now, we have one more tuple clustering tc_3 which can summarize more than two tuples. And, every tuple can be covered by some tuple clusterings which can summarize at least two tuples. Then, we can run the set cover algorithm on these tuples and the tuple clusterings tc_1, tc_3, tc_5, and tc_6. Here, neither tc_2 nor tc_7 is fed into the set cover algorithm since none of them covers enough number (2) of tuples.*
- *The resultant summarization is the same to that in Figure 6(b).*

DEFINITION 5.1 (PARETO-UTIL). *Let T be a data table and let T' and T'' be two α-summarized clusterings. T'' is* `pareto-util` *preferable over T' iff*

$$\forall_{\mu(t')\equiv\mu_2(t'')} \quad \big(util(\mu(t') \preceq t') \leq util(\mu_2(t'') \preceq t'')\big) \wedge$$

$$\exists_{\mu(t')\equiv\mu_2(t'')} \quad \big(util(\mu(t') \preceq t') < util(\mu_2(t'') \preceq t'')\big) .$$

`Pareto-util` criterion prefers solutions where (for any monotonic merge function, $\uplus$, e.g., minimum or summation) it is not possible to obtain any further improvements without having to make any single tuple worse in utility. The ranked set cover algorithm finds summarizations with Pareto-util.

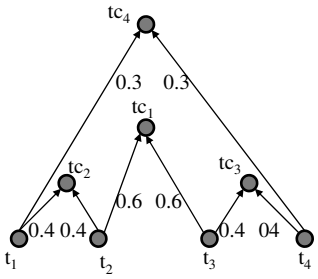


Figure 9: An example tuple clustering lattice for which the ranked approach is sub-optimal under `max_sum_util` criterion

The correctness of the above ranked algorithm depends on the objective function used. Thus, we need to consider each preference criterion independently. Before we do so, however, we state a fundamental lemma:

LEMMA 5.1 (MONOTONICITY). *If the set of tuple clusterings, $TC_{combined}$, contains a (*not necessarily optimal*) α-summarized table clustering (i.e., satisfying the α constraint), then any superset of $TC_{combined}$ also contains the same α-summarized table clustering.*

Max_min_util Criterion.

THEOREM 5.1. *The ranked algorithm is correct w.r.t. the preference criterion,* `max_min_util`.

Proof sketch A direct corollary of Lemma 5.1 and of the fact that tuple clusterings are considered in decreasing order of utility is that the first solution satisfying the α constraint is also the one which has the largest minimum utility. Any further solution will need to include at least one tuple clustering with a lower utility[3], thus cannot improve on this objective function. □

In fact, since the input streams are sorted in decreasing order of utilities of tuple clusterings, the first α-summarized solution enumerated using ranked processing will also be the one with the maximum minimum utility; i.e., it will implicitly satisfy the `max_min_util` criterion. Thus the integer program can be executed without any objective function.

Max_sum_util Criterion. Secondly, let us consider the `max_sum_util` preference criterion. Figure 9 provides a counter-example illustrating that the above ranked algorithm **does not** provide an optimal solution for this objective function. In this example, the weights on the edges are the corresponding tuple clustering utilities, thus the order in which clustering will be considered is tc_1, tc_2, tc_3 and tc_4. The ranked algorithm will provide a 2-summarized solution (using tc_2 and tc_3) with total utility 1.6, after tc_1, tc_2, and tc_3 are seen. However, there is a better solution (using tc_1 and tc_4), with overall utility 1.8.

Pareto_optimal Criterion. The ranked algorithm returns a *pareto-optimal* solution (i.e., a solution where it is not possible to obtain any improvements without rendering any single contributor worse off).

THEOREM 5.2. *The ranked algorithm is correct w.r.t. the preference criterion* `pareto_util`.

[3]For simplicity, we assume utilities are distinct. It is trivial to extend the argument to the cases where there can be multiple clusterings with identical utilities.

Proof sketch The theorem follows from the fact that, once a solution is found, any future results that will be discovered will use at least one tuple clustering with a lower utility. □

5.2 Ranked Set Cover

In [21], Park and Shim proposed a set cover based approximation algorithm, *APPROX-FQ*, for cell-based k-anonymity through value suppressions[4]. In particular, [21] shows that it is possible to obtain a $2(1 + ln2k)$-approximation on the amount of cell-suppressions by first generating a $(k, 2k-1)$-cover (where the sizes of the sets are between k and $2k-1$) whose suppression length sum is at most $(1 + ln2k)$ times that of the optimal solution and then converting the resulting $(k, 2k-1)$-cover into a $(k, 2k-1)$-partition where all the sets are disjoint. While generating the $(k, 2k-1)$-cover for a given table, [21] selects the sets to be processed in the increasing order of *suppression-length per uncovered records.* To further speed up the cover generation, *APPROX-FQ* restricts processing to only those itemsets with at least k support.

The approach underlying the k-anonymization algorithm in [21] is not directly applicable to our problem: in particular, [21] does not consider a lattice (values are either taken as given or simply suppressed) or take into account utilities. In order to be applicable for α-summarization with lattices,

- instead of simply suppressing (values in the cells), the $(k, 2k-1)$-cover should generalize (tuples) based on the available lattice information, and
- instead of picking sets based on their suppression length sum, we need to pick sets (tuple clusterings) based on their utilities per uncovered tuple.

The first item above cannot be directly achieved in the suppression based *APPROX-FQ* framework: to merge two sets, *APPROX-FQ* simply unions the corresponding suppression lists. Thus, while *APPROX-FQ* can always return a k-anonymous solution simply increasing the amount of suppression, when generalizations are fed in an incremental manner, the set cover approximation may simply fail. Thus, the set cover may need to be called multiple times, once for each execution of Step 6a). If done in a naive way, this may cause significant overhead in terms of subproblems that are repeatedly re-evaluated as the set grows incrementally.

On the other hand, we see that it is possible to leverage the algorithmic structure of the set cover solutions to focus the ranked processing to only those parts of the problem which require further evaluation and avoid repeated calls to set cover algorithm. Figures 10 through 13 provide the pseudocodes for the ranked set cover (RSC) algorithm to replace the optimization steps (Step 6a) of the ranked algorithm in Figure 7. Figure 10 provides the main body of the set cover algorithm. As in [21], the algorithm first creates a $(\alpha, 2\alpha-1)$-cover (but using tuple clusterings instead of cell-suppressions), then converts this cover into a partition by eliminating overlaps to create a α-summarized table.

Figure 11 describes how a $(\alpha, 2\alpha-1)$-cover of the given set of tuples is created by iteratively picking those tuple clusterings with minimum or maximum utilities per uncovered tuples for merging. While this step reflects the outline of the

[4]Note that a value suppression can be seen as an extreme degree of generalization, where the value is replaced with the root of the corresponding hierarchy.

Input:
- A set $TC = \{tc_1, \cdots, tc_m\}$ of tuple clusterings for all tuples $T = t_1, \cdots, t_n$,
- Set of tuples $tuples(tc_i)$ for $\forall tc_i \in TC$,
- Summarization parameter α,
- $TCStr_i$-s, tuple clustering streams ($TCStr_i$s) sorted in the descending order of utilities for each tuple in T (see Figure 7).

Output: A cover set $CSet$ such that
- $\forall t \in T$, $t \in C_i.tuples$ where $C_i \in CSet$. (i.e., every tuple is covered).
- $\forall C_i \in CSet$, $\alpha \leq |C_i.tuples| \leq 2\alpha - 1$. (i.e., satisfy a loose version of α-summarization).
- $\forall (C_i, C_j) | C_i \in CSet, C_j \in CSet$, $C_i.tuples \cap C_j.tuples = \emptyset$. (i.e., no tuple is covered by two sets).

ranked-set-cover(TC, $tuples(tc_i)$−s, α, $TCStr_i$-s)
1. $CSet'$ = split-cover(TC, $tuples(tc_i)$-s, α, T);
   /*Get tuple covers such that for $\forall C \in CSet'$, $\alpha \leq |C.tuples| \leq 2\alpha - 1$, but each pair (C_i, C_j) in $CSet'$, $C_i.tuples \cap C_j.tuples$ may not be empty.*/
2. $CSet$ = cover-to-Partition ($CSet'$, α, T, $TCstr_i$);
   /*Get tuple covers such that for $\forall C \in CSet$, $\alpha \leq |C.tuples| \leq 2\alpha - 1$, and each pair (C_i, C_j) in $CSet$, $C_i.tuples \cap C_j.tuples = \emptyset$*/
3. /*From the set $CSet$, compute the final combined utilities*/
   For (each $C \in Cset$)
   (a) Generalize every $t \in C.tuples$ to $C.tc$ with related utility $C.util_t$;

/* C represents a set cover. $C.tc$ is the corresponding tuple clustering, and $C.tuples$ is a set of tuples in T covered by $C.tc$, $C.util$ is the combined utility for generalizating tuple $t \in C.tuples$ to $C.tc$.*/

Figure 10: The main body of *ranked-set-cover*, called by Step 6a in Figure 7

split-cover(TC, $tuples(tc_i)$-s, α, T);
1. $CSet' = \emptyset$; $T_{covered} = \emptyset$;
2. while($T_{covered} \neq T$)
   (a) Let $tuples(tc_i)_{uncovered} = tuples(tc_i) \cap (T - T_{covered})$; (i.e., $tuples(tc_i)_{uncovered}$ contains tuples that are not covered by any cover in $CSet'$)
   (b) Choose tc_i such that $\frac{util(tc_i)}{min\{|tuples(tc_i)_{uncovered}|, 2\alpha-1\}}$ is minimized (or maximized);
   /*minimized/maximized means choosing the minimum/maximum ratio; $util(tc_i)$ is the combined utility that tuples in $tuples(tc_i)$ are generalized to tc_i. */
   (c) if $|tuples(tc_i)| \leq 2\alpha - 1$ (i.e., tc_i covers less than $2\alpha - 1$ tuples)
      i. Let $\mathcal{T} = tuples(tc_i)$;
   (d) else if($tuples(tc_i)_{uncovered} > 2\alpha - 1$)
      i. Choose a subset $\mathcal{T}$ of $tuples(tc_i)_{uncovered}$ s.t. $|\mathcal{T}| = 2\alpha - 1$
   (e) else
      i. Choose a subset $\mathcal{T}$ of $tuples(tc_i)$ s.t. $|\mathcal{T}| = max\{\alpha, tuples(tc_i)_{uncovered}\}$ and $tuples(tc_i)_{uncovered} \subseteq \mathcal{T}$;
   (f) Let $\mathcal{U}$ be the related utilities that tuples in $\mathcal{T}$ are generalized to tc_i;
   (g) $T_{covered} = T_{covered} \cup \mathcal{T}$;
   (h) $C_{new} = \langle tc_i, \mathcal{T}, \mathcal{U} \rangle$;
   (i) $CSet' = CSet' \cup C_{new}$;

Figure 11: The *split-cover* procedure

$(k, 2k-1)$-cover algorithm in [21], there are two fundamental differences: (a) First of all instead of amount of suppression being used as a measure to select the sets to be merged, the algorithm in Figure 11 selects the tuple clusterings based on their degrees of utilities (Step 2b). (b) Secondly, as we will show in Section 6, instead of always picking sets in a way to *minimize* the suppression ratio per uncovered tuples (as in [21]), under different conditions, in Step 2b it may be preferable to *maximize* or *minimize* the utility ratio.

Figure 12 shows the pseudo-code for converting a given $(\alpha, 2\alpha-1)$-cover into a $(\alpha, 2\alpha-1)$-partition. The algorithm

cover-to-partition($CSet$, α, T, $TCstr_i$)
1. for($t \in T$)
   (a) Let P contain all the C_i-s s.t. $t \in C_i.tuples$ and $C_i \in CSet$;
   (b) while $|P| \geq 2$
      i. Randomly pick two covers C_i and C_j from P;
      ii. if $|C_i.tuples| > \alpha$ or $|C_j.tuples| > \alpha$
         A. Remove t from the larger $C.tuples$
         B. update $C.util$;
         C. Remove C from P;
      iii. else
         A. C_{new} = get-best-common-tc(C_i, C_j, $TCstr_i$s);
         B. Remove C_i and C_j from P;
         C. Insert C_{new} to P;
2. return $CSet$;

Figure 12: The *cover-to-partition* procedure

get-best-common-tc(C_i, C_j, $TCstr_i$s)
1. $C_{new}.tuples = C_i.tuples \cup C_j.tuples$;
2. for(each $t \in C_{new}.tuples$)
   (a) Let $TCStr_t$ be the tuple clustering stream for t;
   (b) $TCstrPointer(t)$ = min{position of $C_i.tc$ in $TCstr(t)$, position of $C_j.tc$ in $TCstr(t)$};
   /*the position of a tc in $TCStr(t)$ is the end of $TCStr(t)$ if tc is not in $TCStr(t)$*/
   (c) Let $TCStr'_t$ = tuple clusterings appear after $TCstrPointer(t)$ in $TCStr_t$;
3. Identify tuple clustering tc_i whose combined utility computed from $utils(tc_i)$ is maximal;
4. $C_{new}.tc = tc$;
5. $C_{new}.utils = utils(tc_i)$;
6. return C_{new};

Figure 13: The *get-best-common-tc* procedure

eliminates overlaps between sets included in the $(\alpha, 2\alpha-1)$-cover by either removing the tuples in the intersection (if there are sufficiently many other tuples in the sets to ensure α-summarization or (if sets are small) merging them under a new tuple clustering. Note that unlike [21] which simply unions the suppression lists to merge the given two tuple sets (each defined by a list of suppressions), in Step 1(b)iiiA, our cover-to-partition algorithm leverages the knowledge about available tuple clustering options to pick a high utility clustering that covers both clusterings.

The pseudo-code for the procedure which implements this process is presented in Figure 13. Step 3 of the pseudo-code identifies a tuple clustering that covers the two sets of tuples using the tuple clustering streams that were provided as input to the ranked algorithm. Note that relying on the monotonicity of the utility merge functions, this step can be implemented using a ranked top-1 join algorithm [10, 11] on the corresponding (utility-sorted) tuple clustering streams.

6. PERFORMANCE EVALUATION

In this section, we evaluate the proposed ranked scheme for α-summarization. In the results presented in this section, we used real data set ("*Adults*" [15, 19] and [12]). The data set contains around ∼30K tuples. The two *Adults* hierarchies from [15] and [12] differ from each other. Thus, using these two sources as ground truth, we have created a combined *lattice*, where for any node in the lattice with two parents, the utilities of the clustering edge to the two parents are set to 0.25 as opposed to 0.5 for all other clustering edges. The utility of a path is computed by multiplying the utilities of the edges on the path. Experiments with other weighting strategies produced similar utility optimality and execution time behaviors. We compare the behavior of two different

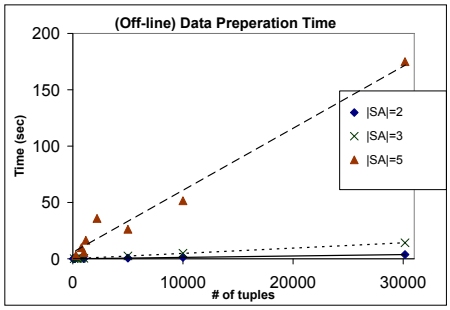


Figure 14: Time for tuple clustering utilities

algorithms: *naive set cover (modified [21])*, and *ranked set cover* proposed in this paper. Experiments ran on a 2.9GHz Pentium (Windows XP) with 2GB main memory.

6.1 Off-line cost

Figure 14 shows the cost of tuple utility evaluation process. As it can be seen here, the time to compute utility evaluations is largely linear in the number of tuples for this data set. As expected, the computation time increases quickly with $|SA|$.

On the other hand, the tuple utility computations can be performed either off-line (and later re-used for different α-summarizations of the same table) or can be computed in a non-blocking, streaming manner and can be pipelined into the ranked algorithm in run time, without having to wait until all tuple utilities are available.

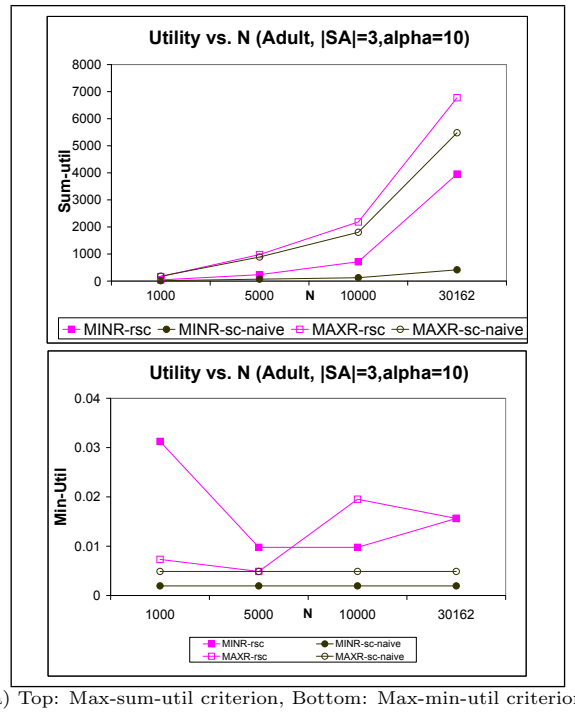
6.2 Summary utilities

Native set cover and ranked set cover:

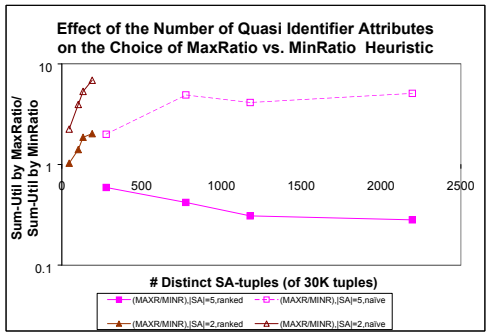
Figure 15(a) compares the ranked set cover approach against the naive-set cover, where all tuple clusterings covering at least α tuples are passed to a version of *APPROX-FQ* [21] modified to take into account clustering lattices and utilities.

The ranked set-cover algorithm performs consistently better than the naive set-cover algorithm, both for max-sum and max-min utility criteria. The *maximum ratio (MAXR)* heuristic (which prefers merging high-utility sets (i.e., tuple clusterings) earlier) performs better than the *minimum ratio (MINR)* heuristic which would merge low-utility sets earlier. While this particular observation is in line with the premise of [21], which picks clusterings with less suppressions early on, other experiments showed that the parallels with [21] do not always hold in the case of ranked set-cover evaluation. Consider Figure 15(b) which plots the relative overall sum-utility (*maximum ratio/ minimum ratio*) as a function of $|SA|$ and the number of distinct SA-tuples[5] in the *Adult* database. As can be seen here, the naive set-cover algorithm indeed performs better with *maximum ratio* heuristic. On the other hand, especially for large $|SA|$ the ranked algorithms performs better when sets to be merged are selected based on *minimum ratio* heuristic. This is because the ranked algorithm has not one, but two sources of clusterings: the first one is during *split-cover* (governed by utility-based set selection heuristic) and the second during *cover-to-partition*, where small-sized overlapping covers are merged into partitions through clusterings. Since MINR

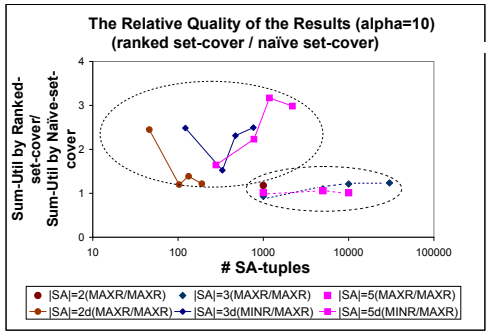
[5]To eliminate any chance of ambiguity due to redundant tuples, we focus only on the distinct SA-tuples



(a) Top: Max-sum-util criterion, Bottom: Max-min-util criterion



(b) Results with solid lines correspond to ranked set cover. Results with dashed lines correspond to naive set cover



(c) Results with solid lines correspond to cases with distinct SA-tuples. To be fair, for each case, the combination that provides the best results (according to (b)) is chosen

Figure 15: Utilities: Naive vs. ranked set cover

selects the tuple clusterings with small utilities first, clusterings passed to *cover-to-partition* likely have low utilities, but are more general. Thus, at the second step, there is less need for significant clusterings. MAXR, on the other hand, optimistically selects high utility covers, but these covers re-

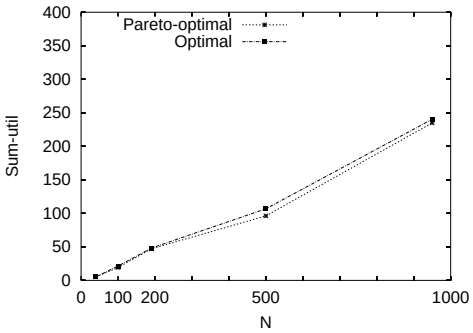


Figure 16: Results for comparing optimal sum-utils and pareto-optimal sum-utils.

quire significant adjustments during the *cover-to-partition* transition. This is more obvious when $|SA|$ is large (i.e., clusterings in second step may involve more loss).

Finally, Figure 15(c) compares the result qualities of the ranked set cover and naive set cover algorithms. As can be seen here, the ranked set-cover algorithm provides significantly better results, especially when the input summarization-tuples are distinct. When the tuples have large degrees of duplicates, the relative qualities returned by the two algorithms become comparable. However, as we will see next, even in this case, the ranked set cover will pay off in terms of gains in execution time and result utilities.

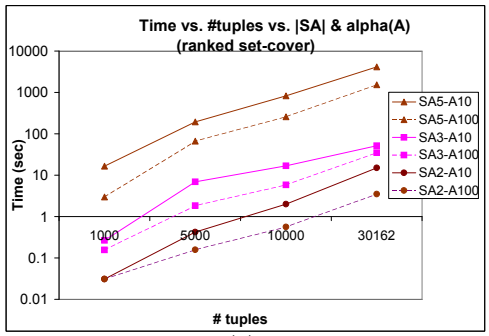
Optimal and pareto-optimal summary utilities

The above set of experiments show the effectiveness of both naive set cover and ranked set cover which solve the set cover formulation by applying the approximate set-cover algorithms proposed in this paper. When we enforce exact algorithm to the set cover formulation, we can get optimal solution for the set cover problem. As discussed in Section 5.1, for min utility, both the naive and ranked approaches get optimal results for the set cover formulation. However, for sum utility, the naive approach can get optimal solution while the ranked approach gets pareto-optimal solution. In this experiment, we compare the optimal solution and the pareto-optimal solution in using sum utility.

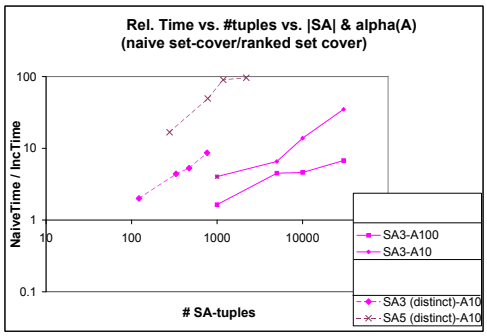
We use Lingo optimizer (LINGO 9.0 solver) to compute the optimal solution for a set cover problem by formulating the problem to Lingo scripts. (The appendix shows how to generate Lingo scripts for the set cover problem). The LINGO non-linear global solver is set to return results within 5% of the optimal. Figure 16 shows the sum utility values for the naive and ranked scenarios with $QI = 2$ and random weights on the lattice edges. It is apparent that the pareto-optimal utility (Pareto-Util) is pretty close to the optimal solution (Max-Sum-util), thus shows that the pareto-optimal solution is a good estimation of the optimal solution.

6.3 Run-time cost

Above, we have seen that the result qualities of the proposed ranked set-cover method can surpass those of naive set-cover algorithms when applied to α-summarization with weighted clustering lattices. Figure 17 shows that this benefit does not come with any execution time overhead. In fact, the execution time of the ranked set cover algorithm is largely linear in the number of $|SA|$-tuples (Figure 17(a)),



(a)



(b)

(in naive set cover, SA5-A10, SA5-A100 failed to complete in 3hours)

Figure 17: Time: Naive vs. ranked set cover

while (especially when α is low and the input consists of distinct summarization-tuples) the naive set cover algorithm quickly becomes multiple orders slower (upto 100X in these experiments).

7. CONCLUSIONS

In this paper, we first formulated the problem of value clustering-based α-summarization of data in the presence of weighted value clustering hierarchies. We provided a mechanism to express alternative clustering strategies and to account for various utility measures in summarization alternatives. We presented a novel ranked set cover based algorithm which reduces the complexity of the problem. Experiment results showed significant gains in execution time when the proposed ranked approach is used.

8. REFERENCES

[1] G. Aggarwal, K. K. Tomas Feder, R. Motwani, R. Panigrahy, D. Thomas, and A. Zhu. Approximation algorithms for k-anonymity. *Journal of Privacy Technology*, 2005.

[2] R. Alfred and D. Kazakov. Data summarization approach to relational domain learning based on frequent pattern to support the development of decision making. *ADMA*, 2006.

[3] F. Buccafurri, F. Furfaro, D. Sacca, and C. Sirangelo. A quad-tree based multiresolution approach for two-dimensional summary data. In *SSDBM'2003:*

[4] A. L. Buchsbaum, D. F. Caldwell, K. W. Church, G. S. Fowler, and S. Muthukrishnan. Engineering the compression of massive tables: an experimental approach. In *SODA*, 2000.

[5] A. L. Buchsbaum, G. S. Fowler, and R. Giancarlo. Improving table compression with combinatorial optimization. *J. ACM*, 50(6):825–851, 2003.

[6] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *SIGMOD Record*, 26(1):65–74, 1997.

[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 2001.

[8] A. Cuzzocrea, F. Furfaro, and D. Saccà. Hand-olap: A system for delivering olap services on handheld devices. In *ISADS '03*
[9] A. Cuzzocrea, D. Saccà, and P. Serafino. A hierarchy-driven compression technique for advanced olap visualization of multidimensional data cubes. In *DaWaK*, pages 106–119, 2006.
[10] R. Fagin. Combining fuzzy information from multiple systems. In *PODS*, 1996.
[11] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 2003.
[12] B. C. M. Fung, K. Wang, and P. S. Yu. Top-down specialization for information and privacy preservation. In *ICDE*, 2005.
[13] C. A. Hurtado, C. Gutierrez, and A.O.Mendelzon. Capturing summarizability with integrity constraints in olap. *ACM Trans. Database Syst.*, 30(3):854–886, 2005.
[14] C.Hurtado and A.Mendelzon. Reasoning about summarizability in heterogeneous multidimensional schemas. ICDT 2001.
[15] V. S. Iyengar. Transforming data to satisfy privacy constraints. In *Proc. of ACM SIG-KDD*, pages 279–288, 2002.
[16] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Incognito: Efficient full-domain k-anonymity. SIGMOD 2005.
[17] M.-L. Lo, K.-L. Wu, and P. S. Yu. Tabsum: A flexible and dynamic table summarization approach. *ICDCS*, 00:628, 2000.
[18] A. Meyerson and R. Williams. On the complexity of optimal k-anonymity. PODS 2004.
[19] M. E. Nergiz and C. Clifton. Thoughts on k-anonymization. *Data Knowl. Eng.*, 63(3):622–645, 2007.
[20] W. K. Ng and C. V. Ravishankar. Relational database compression using augmented vector quantization. ICDE 1995.
[21] H. Park and K. Shim. Approximate algorithms for k-anonymity. In *SIGMOD '07*, pages 67–78, 2007.
[22] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. Supporting imprecision in multidimensional databases using granularities. In *SSDBM*, pages 90–101, 1999.
[23] R. Rada, H. Mili, E.Bicknell, and M. Blettner. Development and application of a metric on semantic nets. IEEE Transactions on Systems, Man and Cybernetics, 19(1), 1989.
[24] P. Resnik. Semantic Similarity in a Taxonomy: An Information-Based Measure and its Application to Problems of Ambiguity in Natural Language. JAIR, Vol.11, 1999.
[25] R. Richardson and A.F. Smeaton. Using WordNet in an Knowledge-Based Approach to Information Retrieval. Working paper CA-1294, Dublin City Univ., Dublin, 1994.
[26] Y. Qi, K.S. Candan, and M.L. Sapino. Feedback-based InConsistency Resolution and Query Processing on Misaligned Data Sources, SIGMOD 2007
[27] Y. Qi, K.S. Candan, M.L. Sapino, and K.W. Kintigh. Integrating and Querying Taxonomies with QUEST in the Presence of Conflicts. SIGMOD 2007.
[28] Y. Qi, K. S. Candan, J. Tatemura, S. Chen, and F. Liao. Supporting olap operations over imperfectly integrated taxonomies. In *Proc. of ACM SIGMOD*, 2008.
[29] R. Saint-Paul, G. Raschia, and N. Mouaddib. General purpose database summarization. VLDB 2005.
[30] R. Saint-Paul, G. Raschia, and N. Mouaddib. Database summarization: The saintetiq system. ICDE 2007.
[31] P. Samarati. Protecting respondents' identities in microdata release. TKDE, 13(6):1010–1027, 2001.
[32] K.-L. Wu, S.-K. Chen, and P. S. Yu. Dynamic refinement of table summarization for m-commerce. WECWIS 2002.
[33] M. Rafanelli, A. Shoshani. STORM: A Statistical object representation model. SSDBM90.

9. APPENDIX: 0-1 INTEGER LINEAR PROGRAM (ILP) FORMULATION OF SET COVER PROBLEM

To simplify discussion, let us suppose that the relation T is only on attributes QI. The set cover formulation is converted into a 0-1 integer linear program to get anα-summarization solution.

Given a tuple $t_h \in T$ and a tuple clustering $tc_l \in TC$, let use $g_{h,l} \in \{0,1\}$ to represent the fact that t_h is summarized to tc_l in the final result.

- Input: A set of tuple clusterings $TC = \{tc_1, \cdots, tc_m\}$ for all the tuples $T = \{t_1, \cdots, t_n\}$, where
  Property 1. $\forall t_i \in T, \ \exists tc_j \in TC, \ t_i \preceq tc_j$ (i.e., every tuple has at least one tuple clustering),
  Property 2. $\forall tc_i \in TC, \ |\{t | t \preceq tc_i\}| \geq \alpha$ (i.e., every tc_i summarizes at least α tuples in T).
- Output: A mapping $g_{h,l} \in \{0,1\}$, for all $t_h \in T$ and $tc_l \in TC$, such that
  Cond 1. $\forall t_i \in T, \ \forall tc_j \in TC \ (g_{i,j} = 1) \rightarrow (t_i \preceq tc_j)$ (i.e., tuples are summarized to compatible tuple clusterings),
  Cond 2. $\forall t_i \in T, \ \exists tc_j \in TC \ s.t. \ (g_{i,j} = 1)$ (i.e., every tuple must be summarized to at least one tuple clustering),
  Cond 3. $\forall t_i \in T, \ \not\exists (tc_j, tc_l) \in TC \ s.t. \ (g_{i,j} = 1) \wedge (g_{i,l} = 1)$ (i.e., no tuple can be clustered by more than one tuple clustering),
  Cond 4. $\forall tc_j \in TC, \ |\{t_i | g_{i,j} = 1\}| \geq \alpha$ (i.e., every tc_i summarizes at least α tuples).
  Cond 5. The mapping satisfies the preference criterion selected by the user.
  The output is then a subset, TC', of the input tuple clusterings, where $TC' = \{tc_j \in TC | \exists t_i \in T \ s.t. \ g_{i,j} = 1\}$.

In the integer linear program, $g_{h,l}$ is introduced as an *integer valued* variable:

CONSTRAINT 1. $\forall_{h \leq n, l \leq m} \ 0 \leq g_{h,l} \leq 1$

Since it is an integer valued variable $g_{h,l}$ can only be 0 or 1. When $g_{h,l}$ is zero, it means that either t_h cannot be clustered to tc_l or t_h does not select tc_l as its summarization (even it could). Thus, we express **Condition 1** in a constraint form as follows

CONSTRAINT 2. $\forall_{h \leq n, l \leq m}$, *if* $t_i \not\preceq tc_l$, *then* $g_{h,l} = 0$ *is introduced.*

Note that if $t_i \preceq tc_l$, then no constraint is introduced.

To describe **Conditions 2 and 3** (i.e., for each tuple t_h, there exists one and only one tuple clustering) we use only one constraint which leverages the integer (in fact 0-1) nature of the variable $g_{h,l}$:

CONSTRAINT 3. $\forall_{h \leq n}, \ \sum_{l \leq m} g_{h,l} = 1$.

Next, we describe **Condition 4**, which states that each picked tuple clustering must cover at least α tuples:

$$\forall_{h \leq n, l \leq m} \ (g_{h,l} = 1) \rightarrow \sum_{i \leq n} g_{i,l} \geq \alpha. \quad (1)$$

Once again, leveraging the 0-1 nature of the variable $g_{h,l}$, we express this in constraint form as follows:

CONSTRAINT 4. $\forall_{h \leq n, l \leq m} \ \sum_{i \leq n} g_{i,l} \geq g_{h,l} \cdot \alpha$.

Note that, when $g_{h,l} = 0$ (i.e., the tuple clustering is not picked), the above constraint trivially holds. When $g_{h,l} = 1$, then the summation over all input tuples guarantees the α-summarization condition of Equation 1.

Once fed into a constraint solver, the above constraints will lead to an α-summarization solution. Among all such summarizations, we are looking for one which also maximizes the objective function (**Condition 5**):

$$\texttt{max} \quad \sum_{h \leq n, l \leq m} g_{h,l} \cdot util(t_i \preceq tc_l)$$

for `max-sum-util` criterion or

$$\texttt{max} \quad \min_{h \leq n, l \leq m} g_{h,l} \cdot util(t_i \preceq tc_l)$$

for `max-min-util` preference criterion [6].

Once a solution satisfying the objective function is found, $\forall_{h \leq n, l \leq m}\ g_{h,l} = 1$ is interpreted as $\mu(tc_l) = t_h$. In other words, an α-summarization table T' can be created using those tuple clusterings, where $\forall_{h \leq n, l \leq m}\ g_{h,l} = 1$.

[6]The pareto-util preference criterion cannot be directly expressed in ILP.