Discovering Topical Structures of Databases

Wensheng Wu Berthold Reinwald Yannis Sismanis Rajesh Manjrekar
IBM Almaden Research Center
San Jose, CA 95120
{wensheng,reinwald,syannis,rajeshm}@us.ibm.com

ABSTRACT

The increasing complexity of enterprise databases and the prevalent lack of documentation incur significant cost in both understanding and integrating the databases. Existing solutions addressed mining for keys and foreign keys, but paid little attention to more high-level structures of databases. In this paper, we consider the problem of discovering topical structures of databases to support semantic browsing and large-scale data integration. We describe iDisc, a novel discovery system based on a multi-strategy learning framework. iDisc exploits varied evidence in database schema and instance values to construct multiple kinds of database representations. It employs a set of base clusterers to discover preliminary topical clusters of tables from database representations, and then aggregate them into final clusters via meta-clustering. To further improve the accuracy, we extend iDisc with novel multiple-level aggregation and clusterer boosting techniques. We introduce a new measure on table importance and propose an approach to discovering cluster representatives to facilitate semantic browsing. An important feature of our framework is that it is highly extensible, where additional database representations and base clusterers may be easily incorporated into the framework. We have extensively evaluated iDisc using large real-world databases and results show that it discovers topical structures with a high degree of accuracy.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data Mining

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Database structure, topical structure, discovery, clustering

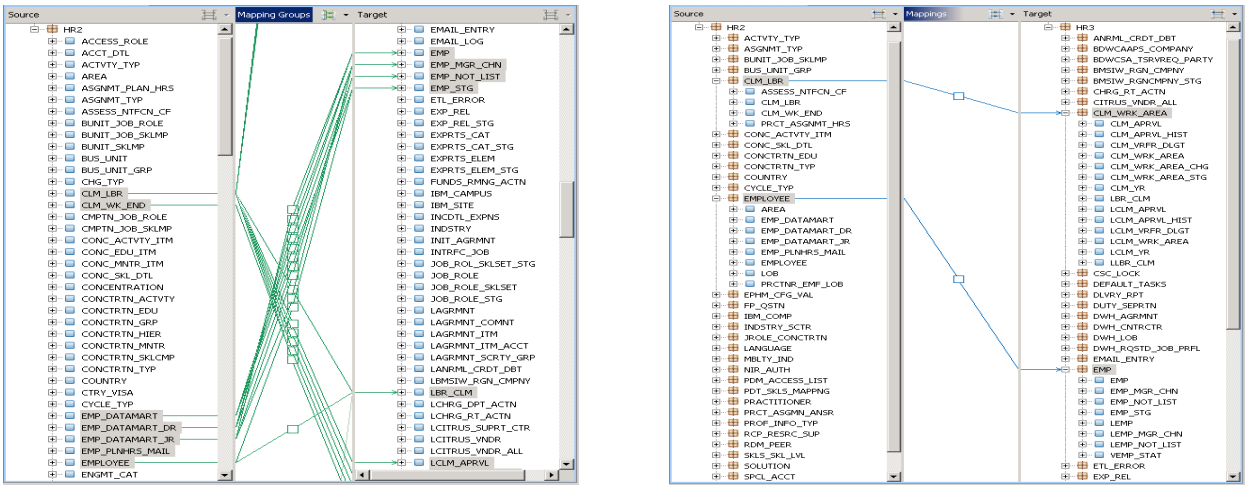
1. INTRODUCTION

A large enterprise typically has a huge number of databases that are increasingly complex [6, 16]. For example, the database for a *single* SAP installation might now contain *hundreds* or even thousands of tables, storing several terabytes of data [30]. To make things worse, the documentation and metadata for these enterprise databases are often scattered throughout the IT departments of an enterprise—they are incomplete, inaccurate, or simply missing [20]. In fact, a recent study [19, 16] indicates that up to 70% of a data architect's time is actually spent on discovering the metadata of databases. Thus, the scale of the databases along with the prevalent lack of documentation, make it a daunting task for data architects and application developers to understand the databases and incur *significant* cost in integrating the databases [26, 16].

To illustrate these challenges, consider a data architect who tries to understand and integrate two large Human-Resource databases HR2 and HR3, shown in the source and target panes of Figure 1.a respectively. These databases are taken from a real-world scenario described in Section 6. Suppose that HR2 has 200 tables, HR3 has 300 tables, and on the average there are 10 attributes per table. Both databases were designed by contractors and have been in service for several years. The designers left the company but did not leave any design documents. Furthermore, the implementation of the databases might not be consistent with the design. For example, the referential relationships of tables often are not enforced in the databases, due to varied reasons including the cost of enforcing the constraints [10, 28]. All these make it extremely difficult for the data architect to understand, reverse-engineer, and integrate the databases.

A *key* step in integrating the databases is to identify the semantic correspondences or *mappings* among the attributes from different databases [25, 12]. The *scale* of the databases again poses serious challenges to this *schema matching* task. Existing matching solutions typically attempt to find mappings between every two attributes [25, 12, 2]. This *all-to-all* approach is based on the assumption that the databases are small and all attributes in one database are potentially relevant to all attributes in another database. This assumption might not hold for large databases [26]. For example, tables in both HR2 and HR3 may be naturally divided into several subject areas or topics such as employee and claim, and the tables from different subject areas are likely not very relevant. As a result, the all-to-all approach is inefficient in that it requires 6M attribute-level comparisons, and inaccurate in that it may match many attributes from irrelevant tables. To illustrate, arrows in Figure 1.a indicate tables whose attributes are matched by this approach. (To avoid the cluttering, not all arrows are shown.) For exam-

SIGMOD'08, June 9–12, 2008, Vancouver, BC, Canada.



(a) **Without** the topical structures (b) **With** the topical structures

Figure 1: Understanding & integrating large databases

ple, a false mapping is discovered between attribute emp_id of table employee in HR2 and labor_claim_id of lbr_clm (labor claim) in HR3, which contain very similar data values.

To address these challenges, we consider the problem of discovering the *topical structure* of a database that reveals how the tables in the database are organized based on their topics or subject areas. Conceptually, each topic or subject area comprises a group of closely related *entities*, where each entity may be represented by multiple tables in the database (e.g., due to normalization).

The topical structure of a database provides an intuitive way of browsing the *semantic* content of the database, and helps users to quickly find relevant information in a large database. For example, Figure 1.b shows the tables in HR2 and HR3 organized by their topics, where each topic is labeled with the name of the most representative table among all the tables on that topic (details are given in Section 5). For example, in HR2, the eight tables on the employee information are grouped under the topic employee.

Knowing topical structures of databases also enables a new approach to matching large database schemas that is both more *scalable* and more *effective* than the previous *all-to-all* approach. Specifically, the matching of two large schemas can now be done in a top-down, *divide-and-conquer* fashion. First, we find similar topics in two databases, where each topic may be simply represented as a text document, e.g., comprising tokens in table names, attribute names, and data values from the tables on that topic. Suppose that HR2 has 20 topics and HR3 has 30 topics. This step involves only 600 comparisons (between text documents), a huge saving from the 6M attribute-level comparisons required by the previous approach. To illustrate, Figure 1.b shows several similar topics discovered between HR2 and HR3, indicated by arrows. For example, topic employee in HR2 is similar to emp in HR3, and clm_lbr in HR2 to clm_wrk_area (claim work area) in HR3. Next, we can focus the further matching effort on the attributes from the tables on similar topics. This would avoid producing many false mappings between attributes from irrelevant tables, e.g., the mapping between employee.emp_id and lbr_clm.labor_claim_id, discovered by the previous approach.

The problem of mining structures of databases has been studied in the context of data cleansing and data integration [10, 3, 20, 28]. However, the focus of previous research was mostly on the discovery of keys [10, 28], foreign keys [10, 21], and functional dependencies [3], while the problem of discovering topical structures has received little attention. A related problem is schema summarization [31] which produces an overview of a complex schema with important elements in the schema. It measures the importance of an element by the number of its relationship & attribute links and the cardinality of its data values. It is not concerned with the topics of the elements. For example, there may be multiple elements in the summary which are all on the same topic, or there may be no elements in the summary to represent the less dominant topics in the schema. In contrast, our goal is to categorize the elements by their topics and exploit the topical structures to not only facilitate semantic browsing but also address the scalability issue in existing schema matching solutions to support a large-scale integration. In addition, we introduce a new measure on table importance based on shortest paths, and propose an approach to discovering representative tables *within each topic*.

In this paper, we describe iDisc, a system that *automatically* discovers the topical structure of a database through clustering. Developing iDisc required several innovations.

Modeling Databases: First, how should we represent the database to the clustering process? There is much *disparate* evidence on the topical relationships among the tables, e.g., table & attribute names, attribute values, and referential relationships. As a result, it may be very difficult to come up with a single best representation. To address this challenge, we propose a novel modeling approach which examines the database from several distinct perspectives and computes *multiple* representations of the database based on its schema information and instance values.

We describe methods for constructing three very different kinds of representations: vector-based, graph-based, and similarity-based. In a *vector-based* representation, each table is represented as a text document and the database as a collection of documents; in a *graph-based* representation, the database is represented as a graph, where nodes are tables and edges indicate the linkages (e.g., referential relationships) between the tables; and in a *similarity-based* representation, the database is represented as a similarity matrix with table similarities computed from attribute values.

Combining Evidence: Second, which clustering algorithm(s) should we employ to discover topical clusters from the database representations? Every algorithm typically has its own strength and weakness, and some may be more suitable for certain representations than others. No single algorithm can perform well over all representations. To address this challenge, we propose a novel discovery framework based on the *multi-strategy* learning principle [23]. In this framework, a *multitude* of base clusterers are employed, each takes a representation and invokes a suitable clustering algorithm to discover preliminary topical clusters of tables. The results from the base clusterers are then aggregated into final clusters via *meta-clustering*. We further propose several approaches to constructing generic similarity-based and linkage-based clustering algorithms and describe how to instantiate them into clusterers for different representations.

The proposed framework is unique in that: (1) It is highly extensible, where additional database representations and base clusterers can be easily incorporated to the system, to further improve its performance. (2) It provides an intuitive and principled way of combining evidence through aggregating the *votes* from the clusterers, rather than directly combining the *disparate* evidence from the database via an adhoc function.

Handling Complex Aggregations: A key component of the framework is meta-clusterer. The meta-clusterer must identify and remove errors in the input clusterers and combine the strength of different clusterers, in order to produce better clusters. A similar problem has been studied in the context of clustering aggregation [15]. Unfortunately, existing solutions suffer from several key limitations.

First, *flat aggregation:* all base clusterers are aggregated at once by a single meta-clusterer. Nevertheless, some clusterers are *inherently* more similar than others. For example, clusterers using the same kind of representations may be more similar or correlated since they look at the database from similar perspectives. In other words, they are like experts with similar (but not exactly the same) expertise. Intuitively, it is easier to identify the errors made by an expert if we compare him to others with similar expertise. To address this limitation, we introduce the concept of similarity level for clusterers and propose an approach to organizing the clusterers into an aggregation tree to perform a *multi-level* aggregation. We show that the new aggregation approach significantly improves iDisc's performance.

Second, *equal combination:* all the input clusterers are treated as being equally good by the meta-clusterer. Nevertheless, the performance of the clusterers may often vary a lot, depending on the characteristics of a *particular data set*: the same clusterer may perform well on one data set but very poorly on another. It is thus desirable to be able to *dynamically* adjust the weights of the clusterers *on-the-fly* so that the votes from the better-performing clusterers are weighted more. To address this problem, we propose a novel *clusterer boosting* approach and shows that it can effectively identify and boost "good" clusterers based on their run-time performance.

In summary, this paper makes the following contributions:

- We formally define the problem of discovering topical structures of databases and demonstrate how topical structures can support semantic browsing and large-scale data integration.
- We propose a novel multi-strategy discovery framework and describe the iDisc system which realizes this framework. The system is fully automatic & highly extensible to other representations and clusterers.
- We propose novel clustering aggregation techniques to address limitations of existing solutions.
- We propose a new approach to finding representative tables using a novel measure on table importance.
- We have extensively evaluated iDisc over real-world databases, and results indicate that it discovers topical clusters of tables with a high degree of accuracy.

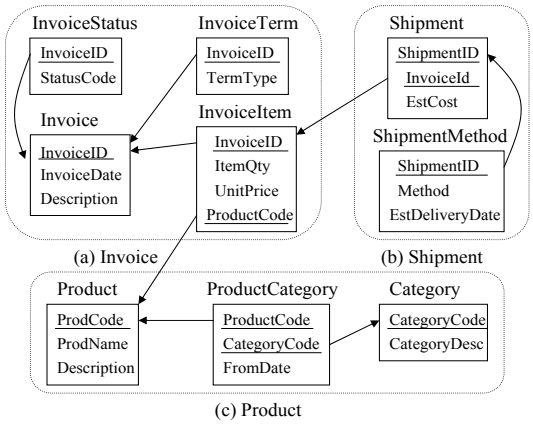


Figure 2: InvDB: An invoice management database

The rest of the paper is organized as follows. Section 2 defines the problem. Sections 3–5 describe the iDisc system. Section 6 presents our results and discusses the current system's limitations. Section 7 reviews related work. Section 8 discusses future work and concludes the paper.

2. PROBLEM DEFINITION

We first formally define the problem. We will use the invoice management database InvDB, shown in Figure 2, as the running example. Note that key attributes are underlined and referential relationships between tables are indicated by directed lines from foreign keys to primary keys. Note also that these keys and foreign keys may not be documented or enforced and may need to be discovered (see Section 3.1.2). We can observe that the tables in InvDB actually fall into three categories or topics: Invoice (the tables in Figure 2.a), Shipment (the tables in Figure 2.b), and Product (the tables in Figure 2.c). The goal of iDisc is then to automatically discover these topics and the tables on each topic. We start by defining topical relationship & structure.

Topical Relationship: Consider a set of topics P. Consider further a database D with a set of tables, where each table T is associated with a topic $p \in P$, denoted as $\text{topic}(T) = p$. Then, we say that there exists a topical relationship between two tables S and T, denoted as $\rho(S,T)$, if $\text{topic}(S) = \text{topic}(T)$. For example, consider the database InvDB in Figure 2. Suppose $P = \{\text{Invoice}, \text{Shipment}, \text{Product}\}$. An example topical relationship is $\rho(\text{InvoiceItem}, \text{InvoiceTerm})$, since $\text{topic}(\text{InvoiceItem}) = \text{topic}(\text{InvoiceTerm}) = \text{Invoice}$.

Note that ρ is transitive, i.e., if $\rho(R,S)$ and $\rho(S,T)$, then $\rho(R,T)$. Clearly, ρ is both reflexive and symmetric. As a result, ρ defines an equivalence class. The topics in P are assumed to be mutually exclusive, so each table in D may be associated with only one topic in P.

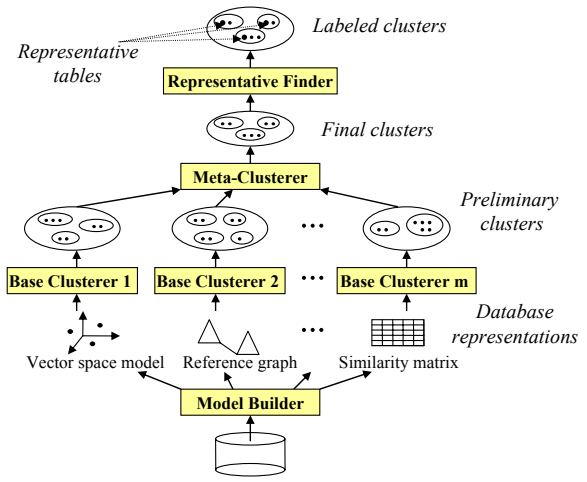


Figure 3: The iDisc architecture

Topical Structure: The topical structure of a database describes how the tables in the database are grouped based on their topical relationship. More precisely, consider a set of topics P, a database D, and a topical relationship ρ between the tables in D with respect to P. The topical structure of D is given by a partition $C = \{C_1, C_2, ..., C_k\}$ formed by ρ over the tables in D, such that the tables in the same group C_i are on the same topic, while the tables from different groups are on different topics. For example, the topical structure of InvDB with respect to the above P is: $\{C_1, C_2, C_3\}$, where C_1 = {InvoiceStatus, InvoiceTerm, Invoice, InvoiceItem}, C_2 = {Shipment, ShipmentMethod}, and C_3 = {Product, ProductCategory, Category}.

Based on the above definitions, we define our problem as follows. We discuss the extensions of the problem to multiple topics per table and hierarchical topical structure in Section 6.3 and Section 8.

Problem Definition: Given a database D with a set of tables, discover: (1) a set of topics P, which the tables in D are about; and (2) the topical structure of D with respect to P, in the form of a partition $C = \{C_1, C_2, ..., C_k\}$ over the tables in D, where $k = |P|$.

3. THE IDISC APPROACH

iDisc takes as the input a database D with a set of tables, and returns the topical structure of D as the output. Figure 3 depicts the architecture of iDisc. It consists of four major modules: model builder, base clusterers, meta-clusterer, and representative finder.

iDisc proceeds as follows. First, the *model builder* examines D from a number of perspectives and obtains a variety of representations for D. Next, each *base clusterer* takes a representation and discovers a preliminary topical clustering (i.e., partition) of the tables in D. These results are then aggregated by the *meta-clusterer* into a final clustering. Finally, the *representative finder* takes the final clustering and discovers representatives in each cluster.

This section describes the model builder, the base clusterers, and the meta-clusterer. Section 4 extends the meta-clusterer to handle complex aggregations. Then Section 5 describes the representative finder.

3.1 The Model Builder

The model builder constructs varied representations for the database from its schema information and instance data. These representations fall into three categories: *vector-based*, *graph-based*, and *similarity-based*.

3.1.1 Vector-Based Representations

Vector-based representations capture the topical structure of the database via the *descriptive* information on the tables. In a vector-based representation, each table is represented as a text document and the database as a collection of documents. Note that in these representations, the structures of individual tables are ignored. There are many possible ways of constructing such documents for the tables in the database, each resulting in a different representation of the database. For example, the document for a table may contain tokens from the name of the table, the names of its attributes, and the content of its attribute values.

To illustrate, consider constructing documents for the tables in InvDB (Figure 2), such that the document for each table contains tokens from both table and attribute names. Then the document d for the table InvoiceStatus will comprise tokens: Invoice, Status, ID, and Code, where both Invoice and Status occur twice in d.

Suppose that the number of unique tokens among the documents for the tables in the database D is n. Then, each document d may be represented as an n-dimensional vector $<w_1, w_2, ..., w_n>$, where the i-th dimension corresponds to the i-th unique token in D, and w_i is the weight of the token for the document d. Many weighting functions may be employed, e.g., the TF $*$ IDF weight [27], and different functions may produce very different representations.

3.1.2 Graph-Based Representations

Graph-based representations capture the topical structure of the database via the *linkage* among the tables. Specifically, the database is represented as a graph, where nodes are tables and edges indicate the linkage between the tables. An important linkage between two tables is their referential (i.e., FK-PK) relationship, where some attributes in one table (i.e., foreign keys) refer to some key attributes in the other table. For example, there is a referential relationship between table InvoiceTerm and table Invoice in InvDB, since the attribute InvoiceID in InvoiceTerm refers to the key attribute InvoiceID in Invoice.

However, the information on keys and foreign keys is often missing in the catalogs, for a variety of reasons including the cost of enforcing the constraints [10, 28]. In fact, in the databases for our experiments (Section 6), keys and referential constraints are neither documented nor enforced by the system. To address this challenge, iDisc implements a method similar to [28] to discover primary keys and then proceeds as follows to discover foreign keys.

Consider a key-attribute A in table T and an attribute B in table T'. B is determined to be a foreign key referring to A in T, if the following conditions are met: (1) $|B \cap A| = |B|$, i.e., B is a subset of A. This is a necessary condition for B to be a foreign key. (2) $|B| > 2$, which is to avoid many false discoveries for boolean attributes, since a boolean attribute may be contained in any other boolean/integer attributes. (3) $|B| > .8|A|$, which is to ensure that the domain of B is sufficiently similar to that of A (not just contained). Note that if $|A| \leq 2$, then Condition 2 might not be satisfied even when Condition 3 is satisfied, e.g., when $|A| = |B| = 2$. (4) NameSim$(A, B) > .5$, where NameSim is a measure (with range [0,1]) on the similarity of attribute names.

3.1.3 Similarity-Based Representations

Similarity-based representations capture the topical structure of the database via the *value-based* similarity between the tables. The idea is that if two tables are about the same topic, they may have several attributes containing similar values [25, 12]. For example, the values for the attribute InvoiceStatus.InvoiceID in InvDB (Figure 2) should be similar to those for InvoiceTerm.InvoiceID. (Note that there are no referential relationships between these two tables.)

In a similarity-based representation, the database D is represented with a $|D| \times |D|$ matrix M, where $|D|$ is the number of tables in D and the entry $M[i, j]$ stores the similarity between the i-th and j-th tables in D. There are many different ways of evaluating the table similarity, each resulting in a different representation for the database. Currently, iDisc employs the following procedure to evaluate the similarity between tables T and T'.

1. Evaluate value similarity between attributes: For every two attributes X and Y, one from each table, compute their similarity as the *Jaccard* similarity between the sets of values in X and Y, i.e., $J(X, Y) = |X \cap Y|/|X \cup Y|$.

2. Discover matching attributes: This step then finds a set Z of matching attributes based on a *greedy-matching* strategy [25]: (1) Let $Z = \emptyset$, $\mathcal{U}$ = all attributes in T, and $\mathcal{V}$ = all attributes in T'. (2) Find $U \in \mathcal{U}$ and $V \in \mathcal{V}$ such that they have a *maximum* (positive) similarity among all pairs of attributes from $\mathcal{U}$ and $\mathcal{V}$. (3) Add attribute pair (U, V) to Z, remove U from $\mathcal{U}$ and V from $\mathcal{V}$. (4) Repeat steps 2 and 3 until no more such pairs can be found.

For example, consider tables T = InvoiceStatus and T' = InvoiceTerm. Suppose that $J(T.\text{InvoiceID}, T'.\text{InvoiceID}) = .75$, $J(T.\text{InvoiceID}, T'.\text{TermType}) = .2$, $J(T.\text{StatusCode}, T'.\text{TermType}) = .15$, and no other attributes have similar values. Then, the first iteration matches attribute T.InvoiceID and attribute T'.InvoiceID, and the second (final) iteration matches T.StatusCode and T'.TermType.

3. Evaluate table similarity: The similarity of T and T', denoted as Sim(T, T'), is then given by the average similarity of their matching attributes: $\frac{\sum_{(X,Y)\in Z}\{J(X,Y)\}}{\max(|T|,|T'|)}$, where $|T|$ is the number of attributes in T. For example, Sim(InvoiceStatus, InvoiceTerm) = (.75 + .15) / 2 = .45.

To summarize this section, we stress that iDisc's goal is not to build best models (which typically do not exist), but to show that it can produce a better solution by building & combining many different (possibly imperfect) models.

3.2 The Base Clusterers

As described, the job of a base clusterer is to take a database representation and discover a preliminary clustering over the tables in the database. Rather than building the individual clusterers separately & repeatedly, iDisc first implements several generic clustering algorithms and then instantiates them into clusterers. In this section, we first describe two generic algorithms employed by iDisc: *similarity-based* and *linkage-based*. We then show how to instantiate the former into clusterers for the vector-based and similarity-based representations, and the latter into clusterers for the graph-based representations.

3.2.1 Generic Similarity-Based Algorithm

Figure 4 shows SimClust, a generic similarity-based clustering algorithm. SimClust takes as the input a set of tables

SimClust($\mathcal{T}$, M, ClsrSim, Q) $\rightarrow C$
Input: $\mathcal{T}$, a set of table $\{T_1, T_2, ..., T_{|\mathcal{T}|}\}$;
M, a similarity matrix for the tables in $\mathcal{T}$;
ClsrSim, a cluster similarity function;
Q, a clustering quality metric
Output: C, a partition of tables in $\mathcal{T}$

1. Set up initial clusters:
   1.1 Let $i = 1$
   1.2 Let $C^1 = \{\{T_1\}, \{T_2\}, ..., \{T_{|\mathcal{T}|}\}\}$
2. Repeat until $|C^i| = 1$
   2.1 Evaluate the quality of C^i via Q
   2.1 Evaluate the similarities of clusters in C^i via *ClsrSim*
   2.2 Find $C_x, C_y \in C^i$ with a maximum similarity
   2.3 Merge clusters C_x and C_y
   2.4 $i \leftarrow i + 1$
3. Return C^i with a maximum Q value

Figure 4: The generic similarity-based algorithm

$\mathcal{T} = \{T_1, T_2, ..., T_{|\mathcal{T}|}\}$, a similarity matrix M whose entry $M[i, j]$ is the similarity between tables T_i and T_j in $\mathcal{T}$, a cluster similarity function *ClsrSim*, and a clustering quality metric Q. It outputs a partition C over the tables in $\mathcal{T}$. Essentially, SimClust can be regarded as a highly customizable hierarchical *agglomerative* clustering algorithm with an automatic stopping rule [17].

SimClust starts by placing each table in $\mathcal{T}$ in a cluster by itself. This generates the first version of the clustering C^1. It then evaluates the quality of C^1, based on the clustering quality metric Q. It also computes the similarities among the clusters in C^1, based on the similarity matrix M and the cluster similarity function *ClsrSim*. Next, it chooses two clusters with a maximum similarity and merges them into a single cluster. This generates the next version of the clustering C^2. It then repeats this process until all the tables are placed in a single cluster. Finally, SimClust returns as the output the clustering with a maximum Q value, among all the $|\mathcal{T}|$ versions of clusterings.

SimClust provides two *customization* points: *ClsrSim* and Q. *ClsrSim* is a cluster similarity function which takes the similarity matrix M and two clusters of tables, C_x and C_y, and computes a similarity value between C_x and C_y. There are many different ways of implementing *ClsrSim*, such as *single-link*, *complete-link*, and *average-link*, where the cluster similarity is respectively taken to be the maximum, minimum, and average similarity between two tables, one from each cluster [17].

Q is a metric for evaluating the quality of clusterings. Determining the number of clusters in a data set is a well-known difficult problem [17]. Many methods have been proposed, such as *elbow criterion*, *gap statistics*, and *cross-validation*, but there is no best solution. Intuitively, a good clustering should be one such that objects within the same cluster are similar while objects from different clusters are dissimilar. Based on this intuition, iDisc implements a default Q as follows:

$$Q(C) = \sum_{C_i \in C} |C_i|/N * (\text{IntraSim}(C_i) - \text{InterSim}(C_i)), \quad (1)$$

where N is the total number of tables in the database, and $|C_i|$ is the number of tables in cluster $C_i \in C$. IntraSim(C_i) is the *average* similarity of tables within the cluster C_i, while InterSim(C_i) is the *maximum* similarity of C_i with any other cluster in C, where the cluster similarity is the *average* similarity of tables between clusters. This default Q is intuitive, easy to implement, and has performed quite well over several

LinkClust($\mathcal{T}$, G, EdgeDel, Q') $\rightarrow C$

Input: $\mathcal{T}$, a set of tables;
G, a linkage graph for the tables in $\mathcal{T}$;
EdgeDel, a function that suggests edges to be removed;
Q', a clustering quality metric

Output: C, a partition of tables in $\mathcal{T}$

1. Let $i = 1$
2. Repeat until G has no edges
   2.1 Let C^i = connected components in G
   2.2 Evaluate the quality of C^i via Q'
   2.3 Let E_c = EdgeDel(G)
   2.4 Remove edges in E_c from G
   2.5 $i \leftarrow i + 1$
3. Return C^i with a maximum Q' value

Figure 5: The generic linkage-based algorithm

databases in our experiments. But note that Q is customizable to other possible implementations (see Section 3.2.3).

3.2.2 Generic Linkage-Based Algorithm

Figure 5 shows LinkClust, a generic linkage-based algorithm. Unlike SimClust, LinkClust discovers groups of related tables based on their linkage information. For example, Figure 6 shows the tables in InvDB (Figure 2) and their linkage information. (The details on how to obtain this graph will be given in Section 3.2.3).

The main idea of LinkClust is to formulate the problem as one of discovering community structure in an interconnected network, e.g., a social network or the Internet. A key observation is that often the links within a community are dense, while the links between communities are relatively sparse. By finding and removing those inter-community links, we may reveal the communities in the network. For example, dotted edges in Figure 6 are the inter-community links in the graph.

LinkClust takes as the input: (1) a set of tables $\mathcal{T}$; (2) a undirected graph G whose vertices are tables in $\mathcal{T}$ and edges indicate the linkage between the tables; (3) a function *EdgeDel* that suggests edges to be removed from G; and (4) a metric Q' on the clustering quality. It returns a partition of tables in $\mathcal{T}$ as the output. LinkClust is a *divisive* algorithm. It starts by finding connected components in G, which forms the first version of the clustering. It then removes edges suggested by the *EdgeDel* from G to produce the next version of the clustering. The process is repeated until no edges remain in G. Finally, the version of the clustering with the highest Q' value is returned as the output.

LinkClust also has two customization points: *EdgeDel* and Q'. We first describe two possible implementations of *EdgeDel*: one based on shortest-path betweenness [5], and the other based on spectral graph theory [8].

(a) Shortest-path betweenness (SP): The key idea is to first find the shortest paths between the vertices, and then measure the *betweenness* of an edge (i.e., the possibility of the edge lying *between* two clusters) by the fraction of the shortest paths that contain the edge. For example, in the linkage graph shown in Figure 6, the number of shortest paths that contain the edge (InvoiceItem, Product) (an inter-community link) is 18 (shown on the edge), the maximum among all the edges, while the number for the edge (Invoice, InvoiceTerm) (a within-community link) is only 8.

More precisely, the betweenness of an edge $e \in E$, denoted as $\beta(e)$, is given by: $\sum_{s,t \in V, s \neq t} \sigma_{st}(e)/\sigma_{st}$, where σ_{st} is the number of distinct shortest paths between vertices s and t, and $\sigma_{st}(e)$ is the number of distinct shortest paths between s and t that contain the edge e. EdgeDel(G) then returns an edge with a maximum β value.

(b) Spectral graph partitioning (SPC): In this case, EdgeDel returns an edge-cut of G, which comprises a set of edges which are likely lying between two clusters. Spectral graph theory provides an elegant way of finding a good edge-cut. Specifically, consider G's Laplacian matrix $\mathcal{L}_G = \mathcal{D}_G - \mathcal{A}_G$, where $\mathcal{D}_G$ is a diagonal matrix whose entry $D[i,i]$ is the degree of the i-th vertex in G, and $\mathcal{A}_G$ is G's adjacency matrix. Then it can be shown that finding a minimum edge-cut of G corresponds to finding the smallest positive eigenvalue λ_2 of $\mathcal{L}_G$ [8]. Further, the eigenvector for λ_2 (known as Fiedler's vector) suggests a possible bi-partitioning of the vertices in G, where the vertices with positive values are placed in one cluster and the vertices with negative values in the other cluster. For example, Figure 6 shows these values next to the vertices. Accordingly, the three tables about product will be placed in one cluster, and the rest of the tables in another cluster. Note that if G contains several connected components, EdgeDel finds edge-cuts for larger components (with more vertices) first.

Metric Q': Similar to Q in SimClust, Q' measures the quality of clusterings in LinkClust. Q' captures the intuition that a good partition of the network should be one such that nodes within the same community are well-connected, while there are only few edges connecting different communities. Based on this intuition, iDisc implements a default Q' as follows [24]:

$$Q'(C) = \sum_{C_i \in C} (|E_{ii}|/|E| - (|E_i|/|E|)^2), \qquad (2)$$

where $|E|$ is the total number of edges in the graph, $|E_{ii}|$ is the number of edges connecting two vertices both in the cluster C_i, and $|E_i|$ is the number of edges that are incident to at least one vertex in C_i. Note that $|E_{ii}|/|E|$ is the observed probability that an edge falls into the cluster C_i, while $(|E_i|/|E|)^2$ is the expected probability under the assumption that the connections between vertices are random, i.e., without regard to the community structure. Finally, we note that Q' is also customizable to other implementations.

3.2.3 Generating Base Clusterers

We now describe how iDisc generates base clusterers by instantiating SimClust or LinkClust. We consider in turn the representations described in Section 3.1.

Vector-based representations: For vector-based representations, iDisc generates base clusterers by instantiating SimClust. Specifically, consider a database D with a set of tables $\mathcal{T} = \{T_1, T_2, ..., T_{|\mathcal{T}|}\}$, and denote the token vector for table T_i as $\hat{T}_i$. First, for every two tables $T_i, T_j \in \mathcal{T}$, we evaluate their similarity based on their token vectors. The similarity between two vectors may be evaluated in a variety of methods, e.g., via the Cosine function commonly employed in Information Retrieval [27], where $\text{Cos}(\hat{T}_i, \hat{T}_j) = \hat{T}_i \cdot \hat{T}_j / (\|\hat{T}_i\| \|\hat{T}_j\|)$. Note that multiple base clusterers may be generated from the same representation by employing different methods for evaluating the vector similarities.

Next, a similarity matrix M is constructed such that its entry $M[i,j]$ holds the similarity between tables T_i and T_j. Finally, a base clusterer is created by instantiating SimClust with $\mathcal{T}$, M, and particular implementations of *ClsrSim* and Q. For example, if *ClsrSim* = *single-link* and Q =

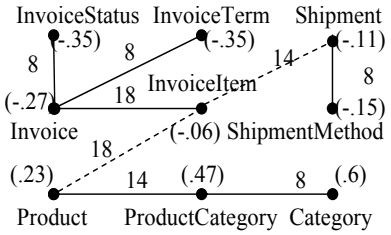


Figure 6: A linkage graph

Table	B1	B2	B3	Meta
InvoiceStatus	1	1	1	1
InvoiceTerm	1	1	1	1
Invoice	1	1	1	1
InvoiceItem	1	1	2	1
Shipment	2	2	2	2
ShipmentMethod	2	2	2	2
Product	3	3	3	3
ProductCategory	3	3	3	3
Category	4	3	3	3

Table 1: A meta-clusterer

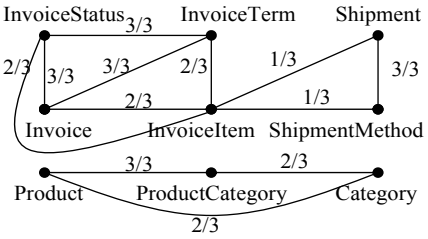


Figure 7: Vote-based similarities

the default Q (Formula 1), then the generated base clusterer can be denoted as SimClust($\mathcal{T}$, M, *single-link*, default_Q).

Graph-based representations: For graph-based representations, iDisc generates base clusterers by instantiating LinkClust. Since LinkClust expects a undirected graph as the input, if the representation is a directed graph, e.g., a reference graph described in Section 3.1, it firsts needs to be transformed into a undirected graph. This is done by ignoring the directions of the edges in the original graph. For example, Figure 6 shows the linkage graph transformed from the original reference graph for InvDB in Figure 2. A base clusterer is then created by instantiating LinkClust with particular $\mathcal{T}$, G, *EdgeDel*, and Q'. For example, LinkClust($\mathcal{T}$, G, *SP*, default_Q') denotes a base clusterer where *EdgeDel* is implemented using the SP method, and the default implementation of Q' (Formula 2) is used.

Similarity-based representations: Similar to vector-based representations, for similarity-based representations, iDisc also generates base clusterers by instantiating SimClust. The difference is that here the similarity matrix in the representation is directly used for the instantiation.

3.3 Aggregating Results via Meta-Clusterer

Given a set of m preliminary clusterings $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_m\}$ from the base clusterers, the goal of the meta-clusterer is to find a clustering C, such that C agrees with the clusterings in $\mathcal{C}$ as much as possible. More precisely, we say that C and $\mathcal{C}_i \in \mathcal{C}$ disagree on the placement of two tables $T, T' \in \mathcal{T}$, if one places them in the same cluster while the other places them in different clusters. Denote the number of disagreements among C and $\mathcal{C}_i$ as $d(C, \mathcal{C}_i)$. Then, the job of the meta-clusterer is to find C that minimizes $\sum_{i=1}^{m} d(C, \mathcal{C}_i)$. A similar problem has been studied in the context of clustering aggregation and ensemble clustering (e.g., see [15]). But these research efforts focused mostly on combining different clustering algorithms (e.g., single-link vs. complete-link), and did not consider how to effectively combine different representation models (see Section 7 for a detailed discussion).

For example, columns 2–4 of Table 1 show the preliminary clusterings given by three base clusterers: Base 1 (B1), Base 2 (B2), and Base 3 (B3), on InvDB. The value j in the column for Base i indicates that Base i places the corresponding table in the j-th cluster of its clustering. For example, the second cluster in the clustering given by Base 1 contains tables Shipment and ShipmentMethod. Both Base 1 & 2 take a vector-based representation (with tokens from table names) as the input and employ SimClust with the default Q. Base 1 uses *complete-link* for *ClsrSim*, while Base 2 uses *single-link*. Base 3 takes a linkage-based representation (the reference graph in Figure 2) as the input and employs LinkClust with the default β and Q'.

It is interesting to note that Base 1 finds four clusters while Base 2 finds three: the cluster on Product in Base 2 is split into two clusters {Product, ProductCategory}, {Category} in Base 1. (We will further discuss this in Section 6.2.) Furthermore, InvoiceItem is placed in the Shipment cluster by Base 3. This is due to the fact that the betweenness score (18) for the edge (InvoiceItem, Invoice) is larger than the score (14) for the edge (InvoiceItem, Shipment). Overall, we can observe that the performance of base clusterers may vary depending on particular database representations and clustering algorithms employed.

The last column of Table 1 shows the clustering C obtained by the meta-clusterer *Meta* (M). Note that there are two disagreements between Meta and Base 1: on Category and Product, and Category and ProductCategory. It can be shown that the total number of disagreements among Meta and the three base clusterers is seven, the minimum among all possible C's.

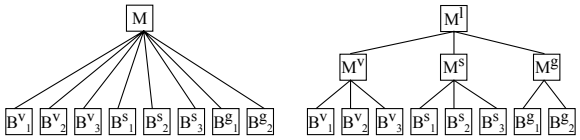
Unfortunately, the problem of finding the best aggregated clustering can be shown to be NP-complete [15]. Several approximation algorithms have been developed [15], and most of them are based on a majority-voting scheme. The meta-clustering algorithm in iDisc is also based on a voting scheme, but has a **key difference**. Unlike other solutions, e.g., the Agglomerative algorithm in [15], it does not assume an explicit clustering threshold (e.g., 1/2 of the votes). Instead, the algorithm *automatically* determines an appropriate number of clusters in the aggregated clustering, based on the *particular* votes from the input clusterers.

Meta-Clustering: The algorithm involves two phases.
(a) *Vote-based similarity evaluation*: Consider two tables $T, T' \in \mathcal{T}$ and a clustering $\mathcal{C}_i \in \mathcal{C}$. A vote from $\mathcal{C}_i$ on the topical relationship between T and T' is given by a 0/1 function $\mathrm{V}_{T,T'}(\mathcal{C}_i)$, which takes on the value one if T and T' are placed in the same cluster in the clustering $\mathcal{C}_i$; and zero otherwise. Based on the votes from the base clusterers, the similarity between two tables $T, T' \in \mathcal{T}$ is computed as: $\frac{1}{m}\sum_{i=1}^{m} \mathrm{V}_{T,T'}(\mathcal{C}_i)$, where m is the number of base clusterers. For example, Figure 7 shows the similarities between the tables in InvDB, based on the votes from B1, B2, and B3.

(b) *Re-clustering*: Next, a similarity matrix M_v is constructed from the above similarities. iDisc then generates the meta-clusterer as SimClust($\mathcal{T}$, M_v, *single-link*, default_Q). But note that other options for *ClsrSim* and Q may also be used.

4. HANDLING COMPLEX AGGREGATIONS

In this section, we extend iDisc to handle complex aggregations. We first describe how to exploit the *prior knowledge* on the inherent property of the clusterers to organize them into an aggregation tree, to perform a multi-level aggregation. We then describe how to adjust the weights of certain clusterers in the *run-time* based on their actual performance, to achieve a more effective aggregation.



(a) Single-level aggregation (b) Multi-level aggregation

Figure 8: Examples of aggregation trees

4.1 Multi-Level Aggregation

A key difference between multi-level and flat aggregations is that in a flat or single-level aggregation, the clusterings from all base clusterers are aggregated at once by a *single* meta-clusterer, while in a multi-level aggregation, aggregation is performed by *multiple* meta-clusterers, with some meta-clusterers taking as the input the aggregated clusterings from the previous meta-clusterers.

Aggregation tree: In general, we may represent the aggregation structure with an *aggregation tree* H. The leaf nodes of H correspond to base clusterers, while the internal nodes correspond to meta-clusterers, each aggregating the clusterings from its child clusterers. The *level* of the aggregation is the depth of the deepest internal node in H. For example, Figure 8.a shows a single-level aggregation tree with eight base clusterers ($\mathcal{B}$'s); Figure 8.b shows a two-level aggregation tree with four meta-clusterers ($\mathcal{M}$'s) on the same base clusterers.

Given a set of base clusterers, the key problem is then how to form an *effective* aggregation tree. A key observation is that the clusterers were not created equally and some are inherently more similar than others. For example, consider again the base clusterers in Figure 8. Suppose that $\mathcal{B}_i^v$'s, $\mathcal{B}_j^s$'s and $\mathcal{B}_k^g$'s are respectively based on vector, similarity, and graph representations. Then, $\mathcal{B}_1^v$ is more similar to $\mathcal{B}_2^v$ than to $\mathcal{B}_1^g$, since $\mathcal{B}_1^g$ looks at the database from a very different angle and may find very different clusters. In other words, $\mathcal{B}_1^v$ and $\mathcal{B}_2^v$ are experts with similar expertise, while the expertise of $\mathcal{B}_1^v$ and $\mathcal{B}_1^g$ may be quite different. Intuitively, if we are to correct the errors by $\mathcal{B}_1^v$, then it may be more effective to compare it against $\mathcal{B}_2^v$ or $\mathcal{B}_3^v$, which has similar expertise, than $\mathcal{B}_1^g$, which has different expertise.

Tree construction: Motivated by the above observation, we define the *similarity level* of clusterers as follows. (a) Level 1 (the most similar): clusterers which take the *same* representation (e.g., a vector-based representation), but employ different clustering algorithms (e.g., single-link vs. complete-link versions of the similarity-based algorithm); (b) Level 2: clusterers which take the *same kind* of representations (e.g., a vector-based representation constructed from table names vs. a vector-based representation constructed from both table & attribute names); and (c) Level 3: clusterers which take *different* kinds of representations (e.g., a vector-based vs. a graph-based representation). Furthermore, if one of the clusterers is a meta-clusterer, their similarity level is given by the *least* similarity level among all the base clusterers.

Based on the above definition, the aggregation tree is then constructed from a set of base clusterers in a bottom-up, clustering-like fashion. It involves the following steps: (1) Initialize a set $\mathcal{W}$ of current clusterers with all the base clusterers. (2) Determine the maximum similarity level l among all the clusterers in $\mathcal{W}$. (3) Find a set S of all clusterers with the similarity level l. (4) Aggregate the clusterers in S using a meta-clusterer $\mathcal{M}$ and remove them from $\mathcal{W}$. Add $\mathcal{M}$ into $\mathcal{W}$. (5) Repeat steps 2–4 until there is only one clusterer left in $\mathcal{W}$, which is the root meta-clusterer.

For example, given the eight base clusterers shown in Figure 8.a, the algorithm produces the aggregation tree shown in Figure 8.b, where $\mathcal{B}_1^v$, $\mathcal{B}_2^v$, and $\mathcal{B}_3^v$ are first aggregated by $\mathcal{M}^v$, which is further aggregated with $\mathcal{M}^s$ and $\mathcal{M}^g$ by $\mathcal{M}^l$.

Table Pairs	B1	B2	B3	MV
(InvoiceStatus, InvoiceTerm)	1	1	1	1
(InvoiceStatus, Invoice)	1	1	1	1
(InvoiceStatus, InvoiceItem)	1	1	0	1
(InvoiceTerm, Invoice)	1	1	1	1
(InvoiceTerm, InvoiceItem)	1	1	0	1
(InvoiceItem, Invoice)	1	1	0	1
(InvoiceItem, Shipment)	0	0	1	0
(InvoiceItem, ShipmentMethod)	0	0	1	0
(Shipment, ShipmentMethod)	1	1	1	1
(Product, ProductCategory)	1	1	1	1
(Product, Category)	0	1	1	1
(ProductCategory, Category)	0	1	1	1

Table 2: An example on clusterer boosting

4.2 Clusterer Boosting

Unlike the multi-level aggregation which utilizes the *static* property of the clusterers, *boosting* exploits their *dynamic* behavior. It first estimates the performance of a clusterer by comparing it to other clusterers and then assigns more weights to the clusterers which are likely to be more accurate. The results from the clusterers are then re-aggregated based on the new weights. Specifically, consider a meta-clusterer $\mathcal{M}$ aggregating clusterings from a set of clusterers $\mathcal{C} = \{\mathcal{C}_1, ..., \mathcal{C}_n\}$. Boosting involves the following steps:

1. Determining a pseudo-solution: The pseudo-solution $\mathcal{S}$ consists of a set of table pairs (T, T') which the majority of the input clusterers place them in the same cluster. Following the notation in Section 3.3, we have $\mathcal{S} = \{(T, T') | \sum_{i=1}^{n} V_{T,T'}(\mathcal{C}_i) > n/2\}$. For example, consider the meta-clusterer shown in Table 1. Table 2 lists the table pairs (out of 36 for InvDB) which are placed in the same cluster by at least one base clusterer. For each pair, it shows which base clusterers (columns 2–4) and whether the majority of the base clusterers (column MV) place them in the same cluster (indicated by 1/0). Then S comprises the 10 table pairs which have value 1 in the column MV.

2. Ranking input clusterers: $\mathcal{S}$ is then utilized to evaluate the input clusterers $\mathcal{C}_i$'s. For this, iDisc employs a measure Ψ which is taken to be the percentage of table pairs in $\mathcal{S}$ that are found by $\mathcal{C}_i$ (i.e., $V_{T,T'}(\mathcal{C}_i) = 1$). Ψ is similar to the *recall* metric in Section 6. For example, Ψ(B1) = .8, Ψ(B2) = 1, and Ψ(B3) = .7. $\mathcal{C}_i$'s are then ranked by their Ψ scores. For example, we have B2 > B1 > B3.

3. Adjusting weights: First, set the initial weights for all clusterers in $\mathcal{C}$ to 1. Consider top k clusterers in $\mathcal{C}$, for a desired number k. Select the clusterer with the best score, increase its weight to 2. Repeatedly find a clusterer C_i with next best score. If C_i is not highly correlated with any previously selected clusterers, set its weight to 2; otherwise, move to the next best clusterer (intuitively, this is because a clusterer with similar expertise has already been boosted).

The correlation between two clusterers $\mathcal{C}_i$ and $\mathcal{C}_j$ is given by their correlation coefficient: $\rho_{X_{\mathcal{C}_i}, X_{\mathcal{C}_j}} = \text{cov}(X_{\mathcal{C}_i}, X_{\mathcal{C}_j}) / \sigma_{X_{\mathcal{C}_i}} \sigma_{X_{\mathcal{C}_j}}$, where $X_{\mathcal{C}_i} = V_{T,T'}(\mathcal{C}_i)$ is a random variable corresponding to the clusterer $\mathcal{C}_i$, and the sample space is taken to be the set of all table pairs (T, T'), where $T \neq T'$. Based on a well-known rule-of-thumb from Statistics, two cluster-

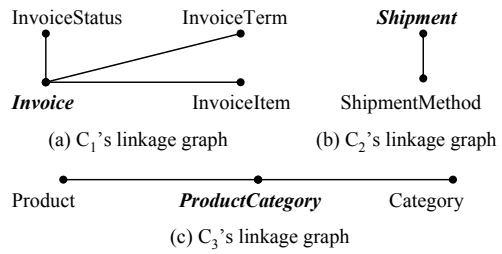


Figure 9: An example on finding representatives

ers may be regarded to be highly correlated if their $|\rho| \geq .3$. For example, the sample space for InvDB contains 36 table pairs (12 of them are shown in Table 2). It can be shown that ρ(B1, B2) = .86, ρ(B1, B3) = .46, and ρ(B2, B3) = .64. So B2 will first be boosted; since B2 is highly correlated to both B1 and B3, no other clusterers will be boosted.

5. FINDING CLUSTER REPRESENTATIVES

In a complex database, there may be a large number of tables on the same topic. As a result, it is often desirable to discover important tables within each cluster. These tables are *cluster representatives*. They serve as the entry points to the cluster and give users a general idea of what the cluster is about. In addition, the names of these representative tables may be used to label the cluster as Figure 1.b illustrates.

In this section, we describe iDisc's *representative finder* component. The key issue in discovering representative tables is a measure on the *importance* of tables. A key observation is that if a table is important, then it should be at a focal point in the linkage graph for the cluster. Motivated by this observation, iDisc measures the importance of a table based on its *centrality* score on the linkage graph. Specifically, given a linkage graph $G(V, E)$, the *centrality* of a vertex $v \in V$, denoted as $\xi(v)$, is computed as follows:

$$\xi(v) = \sum_{s,t \in V, s \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}, \quad (3)$$

where σ_{st} is the number of distinct shortest paths between vertices s and t, while $\sigma_{st}(v)$ is the number of distinct shortest paths between s and t that *pass through* the vertex v. Based on this definition, we now describe the representative discovery algorithm RepDisc in detail.

Representative Discovery: RepDisc takes as the input a clustering $C = \{C_1, C_2, ..., C_k\}$ over the tables in the database D, a linkage graph G of D, and a desired number r. It returns as the output up to r representative tables for each cluster in C.

Consider a cluster $C_i \in C$, RepDisc proceeds as follows to find representatives for C_i. (1) Obtain the linkage graph G_{C_i} for the tables in the cluster C_i. G_{C_i} is a subgraph of G induced by a set of tables in C_i. For example, consider the clustering C from the meta-clusterer Meta shown in Table 1. C contains three clusters $\{C_1, C_2, C_3\}$, e.g., C_1 = {InvoiceStatus, Invoice, InvoiceItem, InvoiceItem}. Figure 9 shows the linkage graphs for these clusters, induced from the complete linkage graph in Figure 6. (2) Evaluate centrality scores for the tables in C_i using Formula 3. (3) Rank the tables by the descending order of their centrality scores, and return top r tables in the ranked list. For example, suppose $r = 1$, the discovered representative tables for the clusters in Figure 9 are highlighted with their names bolded.

Complexity of RepDisc*:* For each of the k clusters in C, three steps are executed. Consider cluster $C_i \in C$ and denote the induced graph for C_i as $G(V_r, E_r)$, where V_r is a set of tables in C_i and E_r is a set of linkage edges between the tables in C_i. In step 1, for every two tables in V_r, we need to determine if there is an edge between them. Suppose G is implemented with an adjacency matrix. This can be done in $O(|V_r|^2)$. Further, the time to create the graph is $O(|V_r| + |E_r|)$. Thus, the overall complexity of step 1 is $O(|V_r|^2)$ (since $|E_r|$ is $O(|V_r|^2)$). Step 2 can be implemented based on Brandes [5], where the complexity can be shown to be $O(|V_r| * |E_r|)$. The complexity of step 3 is $O(|V_r|)$. So the overall complexity for steps 1–3 is $O(|V_r| * |E_r|)$, with the dominant factor being the time for step 2. Assume that each cluster contains about the same number of tables with roughly the same amount of linkage between the tables, the complexity of RepDisc is $O(k * |V_r| * |E_r|) = O(k * |V|/k * |E|/k) = O(|V| * |E|/k)$. In other words, it is about $1/k$ of the time for computing centrality scores for the entire graph G.

6. EMPIRICAL EVALUATION

We have evaluated iDisc on several real-world databases. Our goals were to evaluate the effectiveness of iDisc and the contribution of different system components.

6.1 Experiment Setup

Data Set: We report the evaluation of iDisc on three large Human-Resource (HR) databases: HR1, HR2 and HR3, whose characteristics are shown in Table 3. These databases were obtained from the service department of a large enterprise and cover varied aspects of resource management: HR1 on engagement management, HR2 on skill development, and HR3 on invoice tracking. They are among a large number of HR databases in the service department which has an ongoing effort of understanding & integrating these databases. For each database, we asked a data architect who has been using the database to manually examine the database, and determine (1) a set of topics in the database, and (2) which topic each table in the database is about. These were then used as the "gold standard" for our experiments.

Performance Metrics: We measured the performance of iDisc using three metrics: precision (P), recall (R), and F-measure (F1) [27]. *Precision* is the percentage of table pairs determined by iDisc to be on the same topic that are indeed on the same topic according to the gold standard. *Recall* is the percentage of table pairs determined by the domain expert to be on the same topic that are discovered by iDisc. *F-measure* incorporates both precision and recall. We use the F-measure where precision P and recall R are equally weighted, i.e., $F1 = 2PR/(R + P)$.

Experiments: For each database we conducted three sets of experiments. First, we measured the utility of various database representations (Section 3.1) and the accuracy of individual base clusterers (Section 3.2). Second, we measured the aggregation accuracy of the baseline meta-clustering algorithm (Section 3.3). Third, we measured the impact of the proposed complex aggregation techniques (Section 4). For all the base clusterers and meta-clusterers, the default Q and Q' (Sections 3.2.1 & 3.2.2) were employed. Vector-based representations were constructed from table & attribute names and the Cosine function was employed for

Database	# Tables	# Columns				# Rows
		Total	Numerical	Textual	Date/Time	
HR1	97	1,347	365	771	221	1M
HR2	141	1,549	325	1,003	221	140M
HR3	282	4,367	821	2,721	825	29M

Table 3: The data set for our experiments

computing vector similarities. Since the databases contain a huge number of rows, we first created a sample of 4k size for each attribute, and then use them for discovering foreign keys (Section 3.1.2) and attribute matches (Section 3.1.3).

6.2 Results & Observations

Database Representations & Base Clusterers: Figures 10.a, 10.b & 10.c show the performance of base clusterers on HR1, HR2 & HR3, respectively. For each database, eight base clusterers were employed. These base clusterers are indicated by the types of database representations and clustering algorithms they use. For example, Vec-SL represents a base clusterer which uses vector representation and implements *single-link* as its *ClsrSim* function (Section 3.2.1); Graph-SPC represents a base clusterer which uses graph representation and implements the *EdgeDel* function using the spectral method (Section 3.2.2). For each base clusterer, three bars are shown, representing its performance in precision, recall, and F1 (from left to right). From these results, we can make several observations.

First, base clusterers employing a complete-link algorithm (CL) tend to have higher precision and lower recall than ones based on a single-link algorithm (SL). This is not surprising, given that CL-based clusterers typically produce a large number of small clusters, while SL-based ones produce a small number of large clusters. Second, the precision of base clusterers using graph-based representations is relatively low in HR1 & HR3. Detailed analysis reveals that keys and foreign keys in many tables of these two databases are named using patterns such as "xxx_ID" and "xxx_CD". This confused the foreign-key discovery algorithm and as a result, many false foreign keys were discovered. A possible solution to this is to first determine which tables are similar, e.g., by using vector-based representations, then only discover foreign-keys among the similar tables. Third, the base clusterers utilizing vector-based representations perform consistently well over all three databases. This is due to the fact that similar tables in these database tend to have many common words, e.g., Emp, Emp_Resume, and Emp_Photo. Fourth, the base clusterers utilizing similarity-based representations had poor performance on HR2. The main reason is that a large number of tables in HR2 have several similar timestamp-like columns, e.g., create_dt and del_dt for table creation and deletion datetimes. As a result, many tables are falsely determined to be similar to each other, reducing the effectiveness of similarity-based representations. In summary, the performance of different base clusterers may vary a lot, depending on the characteristics of particular databases, database representations, and clustering algorithms.

Meta-clusterers: For each database, four meta-clusterers were constructed. Meta-Vec aggregates base clusterers Vec-SL, Vec-CL, and Vec-AL; Meta-Sim aggregates Sim-SL, Sim-CL, and Sim-AL; and Meta-G aggregates Graph-SP and Graph-SPC. Note that all these three meta-clusterers aggregate base clusterers which use the same kind of database representations. The last meta-clusterer Meta-All aggregates

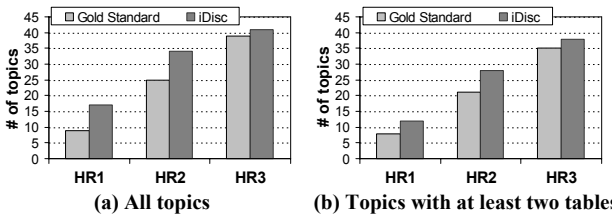


Figure 12: # of topics: Gold standard vs. iDisc

all eight base clusterers. The results are shown in the first four groups of bars in Figures 11.a–11.c. We observe that with the aggregation, the effects of "bad" base clusterers can be cancelled out. For example, in HR1, the precision of Meta-Vec (87.6%) is much higher than that of Vec-SL (44.8%); furthermore, Meta-All is far more accurate than Vec-SL, Graph-SP, and Graph-SPC, and its F1 is higher than that of all base clusterers. All these strongly indicate the effectiveness of meta-clustering.

Multi-Level Aggregation: We then formed a two-level aggregation tree with a top-level meta-clusterer, Meta-LVL, combining three lower-level meta-clusterers: Meta-Vec, Meta-Sim, and Meta-G. The structure of the tree is as shown in Figure 8. The performance of Meta-LVL is shown in the second to last group of bars in Figures 11.a–11.c. By contrasting Meta-LVL with Meta-All (i.e., one-level aggregation), we can observe that F1 values significantly improve over *all* three databases, ranging from 2.6 percentage points in HR3 to as high as 12.7 percentage points in HR1. Furthermore, the recall increases consistently, with very significant improvement in both HR1 (by 19 percentage points) and HR2 (by 15.7 percentage points). All these indicate the effectiveness of multi-level aggregation, where clusterers using the same kind of representations are first aggregated to remove errors and increase the precision, and then a second level meta-clusterer is employed to combine clusterers with different kinds of representations (and thus with quite different "expertise") to increase the recall.

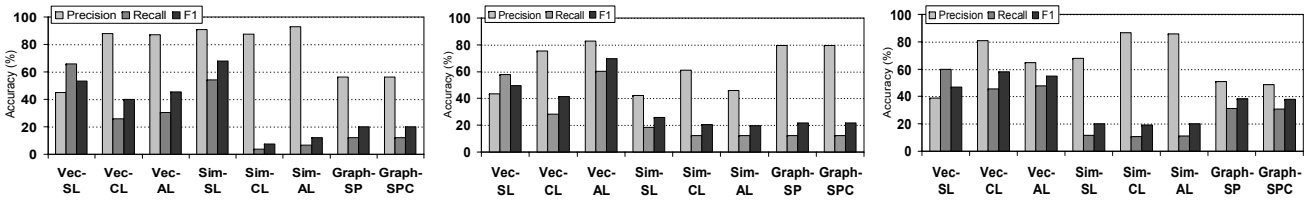
Clusterer Boosting: Next, we applied the boosting technique (Section 4.2) on Meta-LVL, where k was set to $\lfloor n/2 \rfloor + 1$ (i.e., up to two input clusterers will be boosted). The boosted version of Meta-LVL is called Meta-WGT, whose performance is shown in the last group of bars of Figures 11.a–11.c. By contrasting Meta-WGT with Meta-LVL, we can observe increase in F1 consistently over all three databases, with the largest increase (26.8) in HR3. These indicate the effectiveness of the proposed boosting technique.

Meta-WGT is also the best performer among all the clusterers on the data set. Its precision ranges from 67.2 to as high as 89.5, recall from 57.1 to as high as 84.3, and F1 from 61.7 to as high as 86.8. These indicate iDisc's effectiveness.

Number of Topics: Finally, we compared the number of clusters (i.e., topics) discovered by Meta-WGT with that in the gold standard. Figure 12.a plots the total numbers of topics versus the databases. Figure 12.b plots the total numbers of topics with at least two tables versus the databases. We observe that the numbers of topics discovered by iDisc are very close to the numbers given by the gold standard. These further indicate the effectiveness of iDisc.

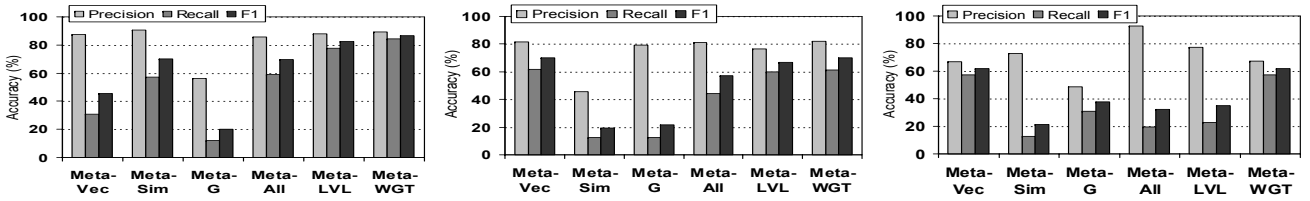
6.3 Discussion

There are several reasons that prevent iDisc from achieving perfect accuracy. First, iDisc may disagree with the domain expert on the *granularity* of partitioning and the num-



(a) On HR1 (b) On HR2 (c) On HR3

Figure 10: The utility of different kinds of database representations & the accuracy of various base clusterers



(a) On HR1 (b) On HR2 (c) On HR3

Figure 11: The performance of the baseline meta-clustering algorithm & the complex aggregation techniques

ber of subject areas in a database. For example, tables on the employee information and tables on the department information may be placed in the same cluster by the domain expert, e.g., a department cluster based on the view that employees may be organized around the departments they work for, while iDisc may produce more refined clusters, e.g., one focusing on the employee information and the other on the department information.

Second, iDisc and the domain expert may also disagree on the assignment of the tables to the clusters, particularly for those "boundary" tables that connect several related entities. For example, a works-for table (which employees work for which departments) may be placed in the employee cluster by one and in the department cluster by the other. A possible solution to the above two cases is to construct the gold standard by employing multiple domain experts and accept a discovered table pair as a correct answer as long as it agrees with one of the domain experts. But there may be a need to first reconcile any conflicts among the domain experts. Another possible solution is to first determine the table pairs which the majority of the domain experts agree on and then compare iDisc's results only to these pairs.

Third, some databases in our experiments contain *reference* tables such as country (with attributes like name, region and ISO code) and language (with attributes like name and code). These tables are often referred to from multiple subject areas. The domain expert may decide to form a subject area (e.g., reference entities) to include all such reference tables or may place them in some referring subject areas. Since iDisc may make very different decisions, this situation largely affects its performance. A possible solution is to extend iDisc so that it produces soft clusters, where each table may be assigned to multiple clusters, i.e., with "soft" membership, and regard these assignments as correct answers if one of them (or one of the top k most confident assignments) agrees with that given by the domain expert.

7. RELATED WORK

We discuss related work from several perspectives.

Mining Database Structures: There has been a lot of research on mining database structure [10, 3, 20, 28, 21]. Bellman [10] is a well-known system that discovers *join relationships* among the tables in a database. There exists a join relationship between two tables if they have joinable attributes, i.e., attributes which are semantically similar. Similar attributes are identified using several set *resemblance* functions, similar to the Jaccard function we utilized to compute attribute similarities (Section 3.1.3). The structure of the database is then a set of tables connected via join relationships or join paths. As we have seen, two tables connected via a join relationship (in particular, a referential relationship) may not be on the same topic. So the goal of our work is to identify such inter-topic links and partition the tables accordingly. In fact, the problem of finding clusters of inter-related tables was also identified in Bellman as an important direction for further research [20].

There is also previous work on abstracting and classifying conceptual schemas such as ER models [13, 29, 7]. As discussed earlier, conceptual schemas are rarely available as part of the database design documentation. Furthermore, this research largely relied on manually specified semantic links among the elements, such as *is-a* and *aggregation* relationships. In contrast, our solution does not require this information.

Data modeling products, such as ERwin [1] and RDA [2], allow users to organize entities in a large logical model by subject areas during a top-down modeling process, to cope with the complexity and facilitate a modular development. Our solution complements these functions by enabling users to *reverse-engineer* subject areas from a large-scale physical database during a bottom-up modeling process.

Information Integration & Complexity Issues: Information integration is a key problem in enterprise data management [6] and has been extensively studied [12]. Recently, the complexity issue in data integration has received active attention [26, 4], largely due to the increasing complexity of data sources. [26] proposes a fragment-oriented approach to matching large schemas in order to reduce the matching complexity. It first decomposes large schemas into several sub-schemas or *fragments* and then performs fragment-wise matching. A fragment considered in [26] is either an XML schema segment, a relational table, or manually specified. Our work is complementary to this work by providing an automatic approach to partitioning a large schema into semantically meaningful fragments. [4] proposes an incremental approach to matching large schemas.

The complexity issue has been also studied in view in-

tegration [18], where user views are mapped to a common semantic data model to reduce the complexity of integration.

Multi-Strategy Learning & Clustering Aggregation: The multi-strategy learning paradigm [23] has been employed in schema matching [11] and information extraction [14] tasks with great success. But we are not aware of any previous attempts to apply this paradigm for mining database structures. [11] describes LSD, a system that matches source schemas against a mediated schema for data integration. LSD employs a set of base learners such as name matcher, naive Bayes learner, and country-name recognizer. The predictions from the base learners are then combined via a *meta-learner*. LSD needs training data to train both its base learners and meta-learner. In contrast, the base clusterers and meta-clusterers in iDisc do not require training. In other words, the learning in iDisc is unsupervised.

DELTA [9] computes the textual similarity between attributes based on their definitions in data dictionary (when available) to discover attribute correspondence. SemInt [22] automatically learns a neural-network classifier based on schema information & data statistics and employs it to match attributes. [22] notes the complementary nature of these two tools and suggests to combine them to improve the matching accuracy. [14] employs a multi-strategy learning approach to extracting information from text documents, where the extraction segments are represented in various models such as term-space and relational models.

A formal treatment to clustering aggregation can be found in [15]. There is a *key* difference between multi-strategy learning [23] and clustering aggregation [15]. A multi-strategy learning approach [23] typically starts with raw evidence and addresses problems such as how to construct multiple effective representations from the evidence and how to design suitable base clusterers for each representation. In contrast, clustering aggregation [15] does not concern with these problems. It starts directly from the results from base clusterers and seeks an effective way of combining them.

8. CONCLUSIONS & FUTURE WORK

We introduced the problem of discovering topical structures of databases and described an automatic discovery system iDisc. iDisc is unique in that (1) it examines the database from varied perspectives to construct multiple representations; (2) it employs a multi-strategy framework to effectively combine evidence through meta-clustering; (3) it employs novel multi-level aggregation and clusterer boosting techniques to handle complex aggregations; and (4) it employs novel measure on table importance to effectively discover cluster representatives. Experiments over several large real-world databases indicate that iDisc is highly effective, with an accuracy rate (F1) of up to 87%.

Besides further evaluation on additional data set, we are investigating two directions to extend iDisc. First, we plan to develop soft clustering & meta-clustering techniques as discussed in Section 6.3 and incorporate them into iDisc to examine their impact on its performance. We intend to draw upon and extend recent research in faceted browsing and search. Second, we plan to extend iDisc to produce hierarchical topical structure, where each topic may be further divided into sub-topics. This would not only enable directory-style semantic browsing but also further support the divide-and-conquer approach to schema matching and reduce the complexity of large-scale integration.

9. REFERENCES

[1] CA ERwin Data Modeler (www.ca.com).
[2] IBM Rational Data Modeler (www.ibm.com).
[3] P. Andritsos, R. Miller, and P. Tsaparas. Information-theoretic tools for mining database structure from large data sets. In *SIGMOD-04*, 2004.
[4] P. Bernstein, S. Melnik, and J. Churchill. Incremental schema matching. In *VLDB-06*, 2006.
[5] U. Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.
[6] P. Brown et al. Toward automated large-scale information integration and discovery. In *Data Management in a Connected World*, 2005.
[7] S. Castano, V. Antonellis, M. Fugini, and B. Pernici. Conceptual schema analysis: Techniques and applications. *TODS*, 23(3):286–333, 1998.
[8] F. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.
[9] C. Clifton, E. Housman, and A. Rosenthal. Experience with a combined approach to attribute-matching across heterogeneous databases. In *IFIP*, 1997.
[10] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *SIGMOD-02*.
[11] A. Doan et al. Reconciling schemas of disparate data sources: A machine-learning approach. In *SIGMOD-01*.
[12] A. Doan and A. Halevy. Semantic integration research in the database community: A brief survey. *AI Magazine, Special Issue on Semantic Integration*, 26(1):83–94, 2005.
[13] P. Feldman and D. Miller. Entity model clustering: Structuring a data model by abstraction. *The Computer Journal*, 29(4):348–360, 1986.
[14] D. Freitag. Multistrategy learning for information extraction. In *ICML-98*.
[15] A. Gionis, H. Mannila, and P. Tsaparas. Clustering aggregation. In *ICDE-05*.
[16] L. Haas. Beauty and the beast: The theory and practice of information integration. In *ICDT*, 2007.
[17] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2006.
[18] S. Hayne and S. Ram. Multi-user view integration system: An expert system for view integration. In *ICDE-90*.
[19] IBM Corporation. Attributes & Capabilities Study, 2006.
[20] T. Johnson, A. Marathe, and T. Dasu. Database exploration and Bellman. *Data Eng. Bull.*, 26(3), 2003.
[21] A. Koeller and E. Rundensteiner. Heuristic strategies for the discovery of inclusion dependencies and other patterns. *Journal on Data Semantics*, V, 2006.
[22] W. Li and C. Clifton. SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data & Knowledge Eng.*, 33(1), 2000.
[23] R. Michalski and G. Tecuci, editors. *Machine Learning: A Multistrategy Approach*. Morgan Kaufmann, 1994.
[24] M. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 2004.
[25] E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4), 2001.
[26] E. Rahm, H. Do, and S. Massmann. Matching large XML schemas. *SIGMOD Record*, 33(4):26–31, 2004.
[27] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McCraw-Hill, New York, 1983.
[28] Y. Sismanis et al. GORDIAN: Efficient and scalable discovery of composite keys. In *VLDB-06*.
[29] T. Teorey, G. Wei, D. Bolton, and J. Koenig. ER model clustering as an aid for user communication and documentation in database design. *CACM*, 32(8), 1989.
[30] Winter Corporation. SAP NetWeaver: A complete platform for large-scale business intelligence, 2005.
[31] C. Yu and H. Jagadish. Schema summarization. In *VLDB-06*.