Top-K Generation of Integrated Schemas Based on Directed and Weighted Correspondences

Ahmed Radwan
University of Miami
a.radwan@umiami.edu

Lucian Popa
IBM Almaden
lucian@almaden.ibm.com

Ioana R. Stanoi
IBM Almaden
irs@us.ibm.com

Akmal Younis
University of Miami
ayounis@miami.edu

ABSTRACT

Schema integration is the problem of creating a unified target schema based on a set of existing source schemas and based on a set of correspondences that are the result of matching the source schemas. Previous methods for schema integration rely on the exploration, implicit or explicit, of the multiple design choices that are possible for the integrated schema. Such exploration relies heavily on user interaction; thus, it is time consuming and labor intensive. Furthermore, previous methods have ignored the additional information that typically results from the schema matching process, that is, the weights and in some cases the directions that are associated with the correspondences.

In this paper, we propose a more automatic approach to schema integration that is based on the use of directed and weighted correspondences between the concepts that appear in the source schemas. A key component of our approach is a novel top-k ranking algorithm for the automatic generation of the best candidate schemas. The algorithm gives more weight to schemas that combine the concepts with higher similarity or coverage. Thus, the algorithm makes certain decisions that otherwise would likely be taken by a human expert. We show that the algorithm runs in polynomial time and moreover has good performance in practice.

Categories and Subject Descriptors

H.2.1 [**Logical Design**]: Schema and subschema; H.2.5 [**Heterogeneous Databases**]: Data translation

General Terms

Algorithms, Design

Keywords

Schema integration, data merging, data integration, model management, schema mapping, top-k generation, interactive schema generation

1. INTRODUCTION

Schema integration seeks to derive a unified representation of the data, used to simplify the access to heterogeneous data sources. The input to integration is a set of source schemas that relate to each other through correspondences or constraints. The output is a consolidated target schema that constitutes a nonredundant unified representation of all the data.

Schema integration has been an active research field for a long time and continues to be a challenge in practice [1, 3, 12, 23, 19, 4, 20]. This problem lies at the core of many metadata applications, such as view integration, mediated schema creation, and ontology merging. The spectrum of applications extends to web-service integration, mashups and distributed web architectures [21].

Although today the process of integrating schemas is partially automated, it is still labor-intensive. In order to reduce the amount of manual intervention that is required from users, we need to modify or avoid parts of the integration process that unnecessarily increase the load on users. Let us follow the steps that generally need to take place while combining two input schemas.

First, the input schemas are run through one or more schema matching algorithms [22, 10, 2] that return *correspondences* between the elements of the schemas. Such correspondences typically have weights reflecting the confidence of the matchers that the two elements are similar or have overlapping semantics. Moreover, the weights in each correspondence direction can be different; thus, an element A can be similar to (or covered by) an element B with weight w, while the element B is similar to (or covered by) element A with weight w', where w and w' are not necessarily equal. We say in such situation that there are two *directed and weighted correspondences* between elements A and B. In a second step of schema integration, all the directed correspondences[1] between two elements are merged into one undirected correspondence with one aggregated weight. Correspondences for which the aggregated weight is above a threshold are kept, while the rest are discarded. Moreover, after the above pruning step, the weights of the remaining set of correspondences are typically themselves discarded. In the third step, several alternatives for combining the input schemas are available, based on the surviving correspondences, and schema integration tools provide interactive means for the users to select a desired integrated schema.

Consider the simple example in Figure 1. The two input schemas describe the structure of two elements: *householder* and *member*. These schemas are run through one or more matching algorithms. For simplicity, assume in this example that atomic elements that match are assigned correspondences weighted with a similarity of 1, in both directions. Correspondences that have weight 0 are not shown. The schema matching algorithm then calculates the over-

[1]There may be more than two, with multiple schema matchers.

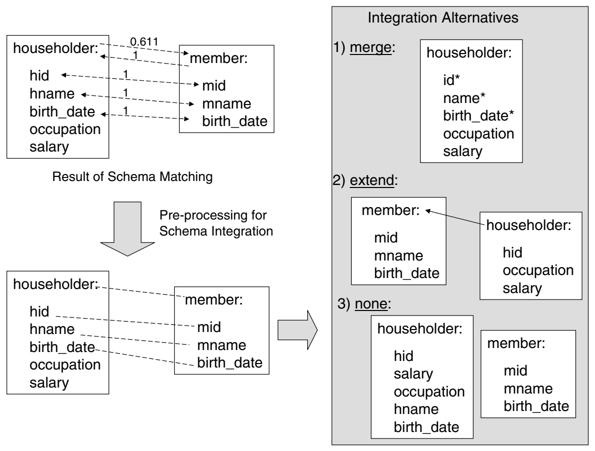


Figure 1: State of the art.

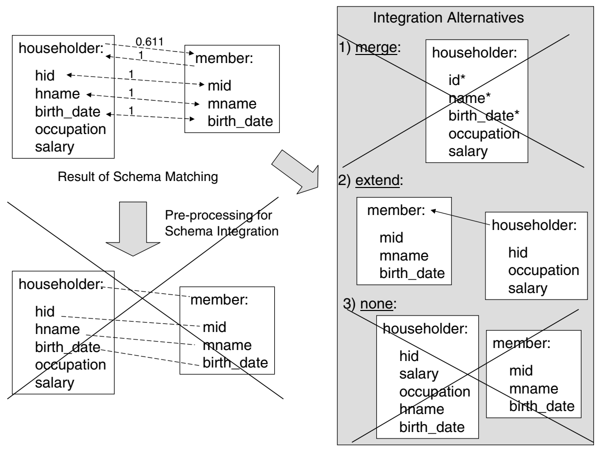


Figure 2: Our solution: keeps more information, outputs the more likely schemas.

all similarities between the non-atomic elements, by aggregating in some way the similarities of the sub-elements. In this example, all the sub-elements of *member* match the sub-elements of *householder*, and therefore *member* is covered entirely by *householder*. Thus, we can conclude that there is a directed correspondence *member* → *householder* with the similarity/weight of 1. On the other hand, the element *householder* contains some sub-elements that are unique, and as a result the similarity of the directed correspondence *householder* → *member* is less than 1.[2] One can see that the weights of the derived correspondences give valuable hints about the coverage of one element by another. Such information is more refined than just saying whether two elements match or not. In particular, it can suggest that one element is likely to be an extension or represents a sub-concept of the other element.

Let us revisit the second step in schema integration, where the results of the matching algorithm are transformed into the undirected correspondences used for the generation of the integrated schema. There are several ways to combine the elements of the two schemas, but in this simplified example there are three natural choices: (1) merge the two root elements *householder* and *member* into one, (2) introduce an extension relationship (or reference) that will say that *householder* is an extension of *member*, or (3) do not combine the two elements at all and just take their union. This is a simple example; in reality schemas are larger and have more complex relationships among their elements. Thus, the space of *possible* candidate schemas that can result from combining the input schemas can be quite large.

The third step in the schema integration process is then concerned with identifying the "best" integrated schema among these alternatives. In most methods, the alternatives are not created explicitly in the system, and the user must "drive" the generation of *one* integrated structure. A good example of such system is described in [19]; in their method, the expert provides a "template" of the integrated schema (also called a mapping model) that effectively specifies the structure of the merged schema and provides the basis for accumulating all the attributes and the relationships from the input schemas. A different kind of method, that is based on the explicit identification of the alternative schema structures, is described in [4]. In their system, a user can systematically explore the alternatives and narrow down, in an interactive way, the desired integrated schema. The advantage of such method is that it is based on a systematic enumeration of the design choices and provides more information to a user. The disadvantage is that the user can be exposed to many "unlikely" schemas and has to explicitly rule out the unlikely choices. Figuring out a way of directing user's attention to the more challenging integration choices was left as an open problem.

Overview of Our Approach In this paper we address the above shortcomings in the schema integration process as follows. (See also Figure 2.) First, we keep and exploit all the information generated by the matching algorithms. In particular, we make use of both the direction and the weights associated with the correspondences between elements. This information enables us to define more refined relationships on the integrated schema. More importantly, this information enables us to *rank* the integrated schemas, by giving higher priority to schemas that combine the concepts with higher similarity or coverage. We are then able to devise a polynomial-time, efficient algorithm that generates the top-k integrated schemas according to the above heuristic.

As a consequence, the resulting system can avoid the generation of many unlikely schemas and can thus minimize the user effort. As our user experiments show, the top-k schemas that we generate will automatically make the correct decisions in the "easy" (and frequent) cases (i.e., concepts that are highly similar will be combined, while concepts that are highly dissimilar will not be combined). Thus, the user can focus her attention on the "difficult" cases while letting the system do the "tedious" part. In fact, we show in section 6.2 that the top-k algorithm can be effectively combined with the interactive system of [4] as follows. The system first generates the (unconstrained) top-k schemas according to the initial input (schemas and correspondences). The user then inspects some of these schema and adds one or more *constraints* in the sense of [4], restricting the space of possible schemas. These constraints enforce decisions to be made on the "difficult" cases. Next, the system reacts by generating the top-k schemas that satisfy the constraints. The process continues, with the user possibly adding more constraints, and so on, until a final schema is obtained.

The top-k generation algorithm itself is non-trivial and we include the proof of its correctness in this paper. Furthermore, we show that the exact complexity of the algorithm is, in the worst case, $O(m_1 m_2 \log(m_1 m_2) + k^2)$, where m_1 and m_2 are the num-

[2] Section 5 will show one method for calculating the similarity measure for complex elements

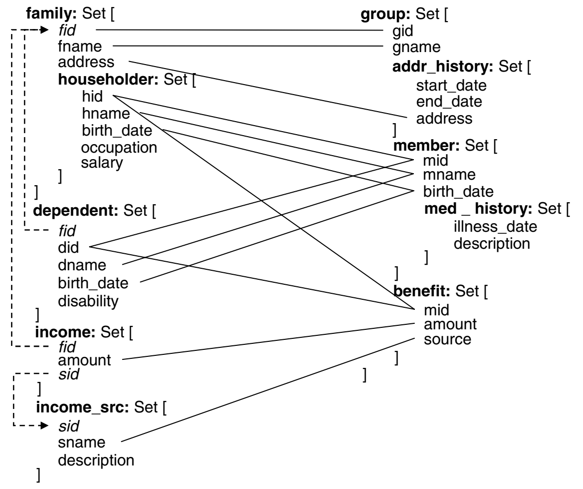


Figure 3: Two input schemas with attribute correspondences.

bers of concepts in the two schemas, respectively. We then show experimentally that the algorithm performs well in practice, on real schemas.

Our method uses an extension on the framework of [4] for schema enumeration. As in [4], we use graphs of *concepts* with has relationships to represent, at a higher-level of abstraction, XML and relational schemas. A concept of a schema is a relation name with an associated set of attributes and, intuitively, represents one category of data (an entity type) that can exist according to that schema (e.g., an "employee", a "department", an "address", etc.). Concepts in a schema may have references to other concepts in the schema and these references are captured by has edges (e.g., "employee" contains a has edge to "department", etc.).

The paper is organized as follows. In section 2 we describe schemas and correspondences, together with their translation in terms of concept graphs. The overall method is overviewed in section 3. We then give our top-k generation algorithm in section 4, where we also present our cost function for differentiating between different ways of combining schemas. Section 5 describes one method for computing the similarities between concepts in two different schemas, given a set of basic correspondences between the atomic elements that appear in such concepts. The experiments are presented in section 6. We review existing related work in section 7 and conclude the paper in section 8.

2. SCHEMAS AND CONCEPT GRAPHS

Consider the two source schemas S_1 and S_2 shown in fig. 3. The schemas are shown using a nested relational model [18] which can be used as a common representation for both relational and XML schemas, in addition to other hierarchical set-oriented data formats. This model is based on sets and records that can be arbitrary nested.

The first schema represents families with their householders, dependents and incomes, as well as the sources for such incomes. The top-level $family$ element represents a *set* of family records, each with three atomic components (also called *attributes*) and one nested set, $householder$, representing the householders for each family. The dotted arrows in schema S_1 represent foreign key constraints: an income has references to both a family and an income source, while a dependent has a reference to a family. The second schema is a variation of the first one, where corresponding to $family$, at the top level, we have a set of $group$ records. Each $group$ record includes nested sets of $addr\_history$, $member$ and $benefit$. As the example illustrates, in general, the source schemas can overlap and also each source schema can have information that is not present in the other (e.g., $salary$, $disability$ in the first schema, $addr\_history$, $med\_history$ in the second schema).

Figure 3 also shows correspondences (lines that represent relationships) between attributes of the two schemas. The correspondences signify "matching" or "similar" attributes (i.e., that carry $similar$ data) in the two schemas. They can be manually specified or discovered through schema matching techniques. Note that each attribute in a schema can have zero, one or multiple correspondences into attributes of the other schema. In general, some correspondences can have "stronger" similarity than others; thus, correspondences can have weights. In our example, for simplicity, we will assume that correspondences between atomic attributes have weights of either 0 or 1. Furthermore, such atomic correspondences typically have no direction, that is, the similarity weight is the same in both directions. We shall distinguish atomic correspondences from the correspondences between complex elements (or concepts, as we shall see shortly), which are calculated based on the former ones, and have values between 0 and 1, even when the atomic correspondences are either 0 or 1. Furthermore, complex correspondences will always have a direction.

Given the source schemas and their correspondences, our goal is to generate an integrated schema that captures the source schemas, by taking the union of all their features while avoiding redundancy. Although in general there are many possible such schemas, our technique enables the system to return a ranked set of k best solutions. As in [4], our integration method uses *concept graphs* as a higher-level, simplified representation of the input and output schemas. The model is described below.

Concept Graphs The objective of concept graphs is to abstract the physical organization of schemas into a more logical view. In the following, let us assume U is a universe of attributes. A *concept* is a relation name C associated with a subset of U (these are the attributes of C). A *concept graph* is a pair (V, has) where V is a set of concepts and has is a set of directed and labeled edges between concepts.

Intuitively, the meaning behind a has edge from A to B is that every instance of concept A has a reference to exactly one instance of concept B. The role of the has edges is to express that certain concepts cannot exist without other concepts (i.e., they *extend* or *refer to* those concepts). In general, there can be multiple edges between the same two concepts, and labels are attached to edges to distinguish between them. As an example, a $student$ concept may have two has edges to a $person$ concept, one reflecting that $student$ "is-a" $person$, and the other reflecting that $student$ "has-a" $person$ as an advisor. Thus, in this model, has edges can encode both *has-a* and *is-a* type of relationships that exist in conceptual models. For simplicity, whenever there is no confusion, we shall drop the labels associated with the has edges. Finally, we note that the has edges in a concept graph can form cycles.

Each schema, with its nesting and constraints, can be replaced by a concept graph. For example, two concept graphs are shown in Figure 4 corresponding to schemas S_1 and S_2. The solid arrows that connect concepts within one schema (e.g., $householder$ to $family$, $income$ to $income\_src$, etc.) represent has edges. For now, let us ignore the dotted lines connecting concepts across schemas (i.e., the correspondences between concepts, which we shall revisit shortly).

To understand how these concept graphs relate to the schemas, consider the concept graph for S_1. There, $family$ and $income\_src$ are top level concepts, with no outgoing has edges; they represent

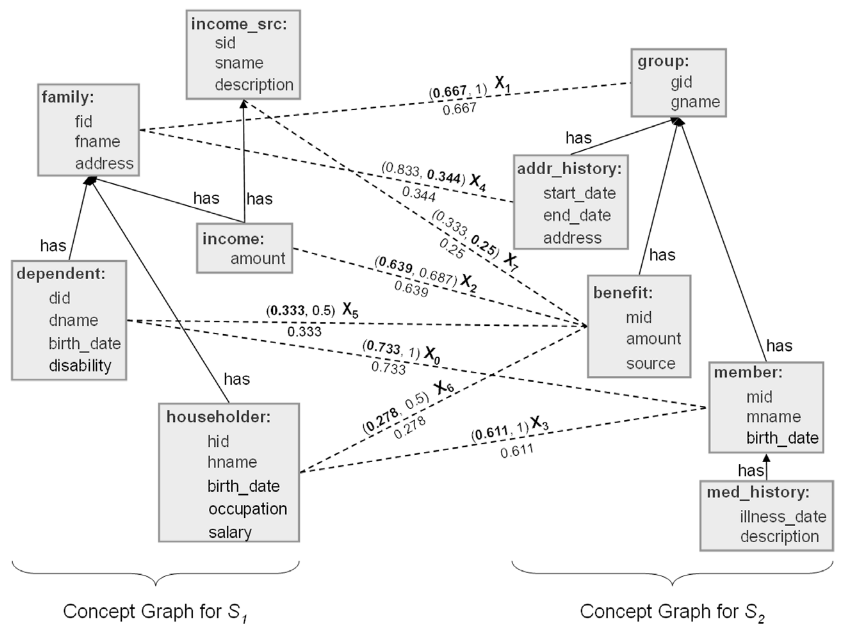

income_src: sid sname description
family: fid fname address
group: gid gname
addr_history: start_date end_date address
income: amount
dependent: did dname birth_date disability
benefit: mid amount source
member: mid mname birth_date
householder: hid hname birth_date occupation salary
med_history: illness_date description
has
(0.667, 1) X1 0.667
(0.833, 0.344) X4 0.344
(0.333, 0.25) X7 0.25
(0.639, 0.687) X2 0.639
(0.333, 0.5) X5 0.333
(0.733, 1) X0 0.733
(0.278, 0.5) X6 0.278
(0.611, 1) X3 0.611
Concept Graph for S1
Concept Graph for S2

Figure 4: Concept graphs with directed similarities.

standalone concepts that do not depend on anything else (according to S_1). In contrast, $householder$ has a reference to $family$, since a householder element cannot exist independently of a family according to the nesting in S_1. Similarly, $dependent$ has a reference to $family$, based on the fact that $dependent$ has a foreign key to $family$ in S_1. Moreover, $income$ has references to both $family$ and $income\_src$, since an income is associated (via foreign keys) with both a family and an income source in the schema S_1.

There is an immediate algorithm that constructs a concept graph from a schema, based on the nesting structure and the integrity constraints [4]. Specifically, a concept can be associated with each set element (i.e., collection) in the schema (e.g., family, householder, etc.). Furthermore, if the set element is nested under another set or has a foreign key into another set, then a corresponding has edge is added. Conversely, there are immediate algorithms for translating concept graphs back into schemas [4]. For the remaining part of the paper, we shall assume that the schemas are given as concept graphs and the schema integration problem is a problem of combining such concept graphs into a unified concept graph.

3. OVERVIEW OF OUR METHOD

The ranking problem we solve is the following: given an input of two concept graphs CS_1 and CS_2 and a table T_{12} encoding the directed and weighted correspondences between concepts, generate the top k schemas that integrate CS_1 and CS_2 while using T_{12} to qualitatively differentiate between possible solutions.

An example of such table T_{12} for the two concept graphs in Figure 4 is Table 1. For each pair (C_1, C_2) of concepts, where C_1 is from schema S_1 and C_2 is from schema S_2, we give two numbers. The first one represents the weight of the directed correspondence $C_1 \rightarrow C_2$, and we shall denote it by $\hat{s}(C_1, C_2)$. The second number represents the weight of the directed correspondence $C_2 \rightarrow C_1$, and we shall denote it by $\hat{s}(C_2, C_1)$. These pairs of numbers (directed similarities) are also shown in Figure 4 on top of the dotted lines that connect the concepts across schemas. For simplicity of terminology, we shall call such a dotted line also a *correspondence*, and use the pair of numbers to differentiate between the two

Table 1: Directed similarities between concepts in S_1 and S_2.

	group	***member***	***benefit***	***med_history***	***addr_history***
family	(0.667,1)	(0.5,0.094)	(0.5,0.094)	(0.3,0.019)	(0.833,0.344)
householder	(0.042,0.75)	**(0.611,1)**	(0.278,0.5)	(0.458,0.062)	(0.111,0.437)
dependent	(0.05,0.75)	(0.733,1)	(0.333,0.5)	(0.55,0.162)	(0.133,0.437)
income	(0.083,0.75)	(0.222,0.25)	(0.639,0.687)	(0.167,0.031)	(0.222,0.438)
income_src	(0,0)	(0,0)	(0.333,0.25)	(0,0)	(0,0)

directions whenever needed. In general, in the graph, we connect concepts by correspondences only if there is one non-zero weight in at least one direction. We call such correspondence a *non-zero* correspondence.

Conceptually, we distinguish between the problem of computing the entries in the table and the usage of it for schema integration. The first problem falls under the general umbrella of schema matching; in section 5 we shall give one method for calculating such numbers based on a given set of correspondences between atomic attributes.

The Space of Candidate Schemas We now consider all possible integrated schemas that can be generated based on the non-zero correspondences that connect concepts in different schemas. Following the approach in [4], in order to consider all the alternatives for using or not using the correspondences, we use bit assignments as follows. An $assignment$ $\mathbb{A}$ is a fixed-sized, ordered vector of bits, where each bit X represents the state of one correspondence. When a specific bit is set to 1, this means that the associated correspondence must be used to combine the two respective concepts. On the other hand, if it is set to 0, then the correspondence is ignored. Different assignments give rise to different ways of combining the concepts, thus yielding different candidate integrated schemas.

As an example, consider the concept graphs and non-zero correspondences in Figure 4. The $assignment$ can be represented as $[X_7\ X_6\ X_5\ X_4\ X_3\ X_2\ X_1\ X_0]$ (in the figure, the variables are shown above the dotted lines that connect the concepts). There are 2^n possible assignments in general, where n represents the total number of non-zero correspondences between concepts; thus, a

Table 2: $merge$ **and** has **Decisions**

Rule	Condition	Decision
1	$\hat{s}(A,B) \geq \lambda$ AND $\hat{s}(B,A) \geq \lambda$	$merge$
2	$\hat{s}(A,B) \geq \lambda$ AND $\hat{s}(B,A) < \lambda$	$B\ has\ A$
3	$\hat{s}(A,B) < \lambda$ AND $\hat{s}(B,A) \geq \lambda$	$A\ has\ B$

naive enumeration of all assignments is not feasible. The method developed in [4] explores the space of assignments in an interactive way, where user constraints are used to direct the exploration of the space in an efficient way.

Top-k Generation of Assignments Here we adopt a more directed exploration strategy that is based on ranking the assignments and generating the top-k. As a necessary step, we shall introduce a cost model that associates a cost to each possible assignment, by aggregating the costs that are associated with the individual bits in the assignments. The top-k generation algorithm, together with the cost model for ranking assignments, is described in section 4.

Combining the Concepts Once we obtain the top$-k$ assignments, we still have to go through one more step, where for each assignment, we compute a concrete integrated concept graph (and integrated schema). For each assignment, for each bit that is assigned 1, we have to decide how to combine the two concepts that are connected by the respective correspondence. As opposed to [4] where there is no choice but to merge the two concepts, in our approach we can make a more refined decision by taking advantage of the directed similarities between the two concepts.

More precisely, we can take one of the following decisions for combining two concepts. (Note that choosing not to combine the concepts can also be a valid decision. This is covered by the case where the corresponding bit in the assignment is set to 0.)

- $merge$: Two quite similar concepts, for which the two weights are high in each direction, can be merged into one, by taking the union of their attributes and the union of their has edges. At the same time, we avoid redundancy and collapse any corresponding (i.e. matching) attributes that may appear in this union into a *single* attribute in the merged concept.
- Introducing a has edge in either direction: In some cases a concept A is covered relatively well by another concept B, but B is not covered well by A (i.e., the similarity $\hat{s}(A,B)$ is high, but the similarity $\hat{s}(B,A)$ is low). This means that most attributes of A are "covered" by B, but the reverse isn't true. We then make B refer to A, by adding a has edge from B to A and removing all the attributes from B that have corresponding attributes in A. A similar has edge from A to B is added in the reverse situation (i.e., the similarity $\hat{s}(A,B)$ is low, but the similarity $\hat{s}(B,A)$ is high). Note that, in these two cases, the newly added has edge did not exist previously in either of the input concept graphs but is created by the integration process.

These decisions are summarized in Table 2. From a database normalization perspective, this enriched set of combination decisions helps minimize duplication of information and, in doing so, safeguards the database against certain types of logical or structural problems, namely data anomalies [5].

To control the combination process, a threshold λ is used to guide the above decisions. In particular, for a $merge$ decision, we require the similarities in both directions to be greater than λ. For a has decision, we require that only one of the directed similarities is above the threshold λ.

To illustrate the impact that the choice of λ has on the combination procedure, we give an example in Figure 5. At the top

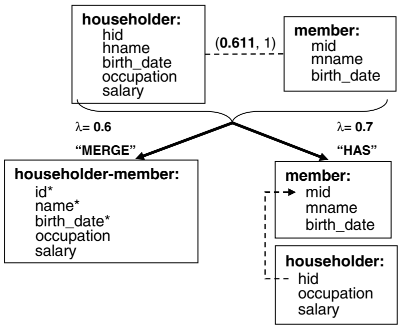


Figure 5: $merge$ **vs.** has **decisions.**

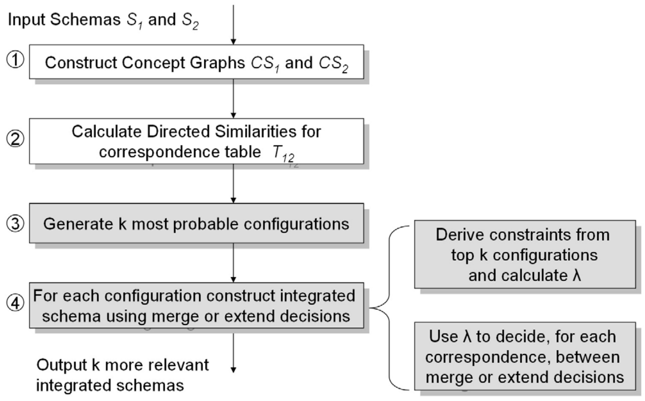

Input Schemas S_1 and S_2
① Construct Concept Graphs CS_1 and CS_2
② Calculate Directed Similarities for correspondence table T_{12}
③ Generate k most probable configurations
④ For each configuration construct integrated schema using merge or extend decisions
Derive constraints from top k configurations and calculate λ
Use λ to decide, for each correspondence, between merge or extend decisions
Output k more relevant integrated schemas

Figure 6: Schema integration process.

we show the concept $householder$ from schema S_1 and the concept $member$ from schema S_2. We also show the correspondence connecting the concepts, together with the directed similarities, $\hat{s}(householder, member) = 0.611$, and $\hat{s}(member, householder) = 1$. If we choose λ to be 0.6, a $merge$ decision will be taken and the resulting schema will consist of a single table for members and householders. If instead we set λ to be 0.7, a has edge will be introduced connecting $householder$ to $member$; this will result in a schema with two tables with a foreign key as shown in the figure, where, attributes in $householder$ having corresponding attributes in $member$ were removed. This schema is arguably better than the one obtained by $merge$, in this example.

Intuitively, the higher λ the better: more refined schemas will be generated in such case. However, we cannot set λ too high either; a high value of λ may cause the combination of two concepts to fail, even though some of the assignments in the top-k require the correspondence to be used! In other words, a high λ may invalidate some of the decisions taken by the top-k algorithm. Choosing a suitable λ is a critical issue and we shall provide such method, by taking the best λ that does not invalidate the decisions made by the top-k generation algorithm.

Summary Figure 6 summarizes the main steps in our system. First, the two input schemas are translated into concept graphs. In the second step, we materialize the table T_{12}, by computing directed similarities between all pairs of concepts across the input graphs. We then invoke the top-k algorithm for the generation of the assignments with the best cost. Based on the top k assignments, we then determine a suitable value for the parameter λ. (The detailed discussion of how λ is determined is deferred to section 4.3.) We then go back to the top k assignments, and based on λ (and the rules in Table 2) we output for each of the top k assignments the corresponding integrated concept graph. The end result is a list of k integrated concept graphs (and integrated schemas) in ranked order.

We note that this is an automatic procedure for generating the top-k integrated schemas. In section 6.2, we show how this procedure can be used in combination with user interaction.

4. TOP-K CANDIDATES

We now address the problem of finding the assignments that will give rise to the most likely integrated schemas out of the entire space of candidate schemas. In order to do this, we introduce a cost function for assignments, which we shall use to rank them and generate the top-k.

4.1 The Cost of an Assignment

In order to compute an aggregated cost for an assignment, we need to consider first the cost associated with each individual decision of whether to use or not a correspondence. Assume a correspondence e connects concepts A and B. Intuitively, the cost must reflect how much the decision to include or exclude the correspondence agrees with the level of similarity between the two concepts connected by the correspondence. At this point, we are only concerned with whether two concepts are to be combined or not; thus, what we need is an *undirected* version of the similarity between the two concepts.

We can use the following simple formula to aggregate the two directed similarities into one number:

$$\hat{S}(A,B) = \min(\hat{s}(A,B), \hat{s}(B,A)).$$

Intuitively, this is a conservative estimate of the similarity between the two concepts. While other formulas can be envisioned (e.g., based on average or weighted average), the formula above follows [6], which analyzed various ways of combining directed *distances* between objects into an undirected distance measure. There, better object matching accuracy was reported when taking the maximum of the directed distances, which in turn corresponds to taking the minimum of the directed similarities.

We now derive the cost associated with the decision of including or excluding a correspondence within a given assignment. If the correspondence is not included (i.e., the bit is set to 0 in the assignment), then we impose a penalty that is proportional with the similarity between the two corresponding concepts: $cost_e = \theta_1 \times \hat{S}(A,B)$. Intuitively, the higher the similarity between the two concepts, the higher the penalty is for not combining the concepts. On the other hand, if the correspondence is included (i.e., the bit is set to 1 in the assignment), then we impose a penalty that is proportional with the $dissimilarity$ between the two concepts: $cost_e = \theta_2 \times (1 - \hat{S}(A,B))$. We shall often use the notation $\hat{D}(A,B)$ for the quantity $1 - \hat{S}(A,B)$ and call it *undirected distance*. Intuitively, the higher the distance between the two concepts, the higher the penalty is for combining the concepts.

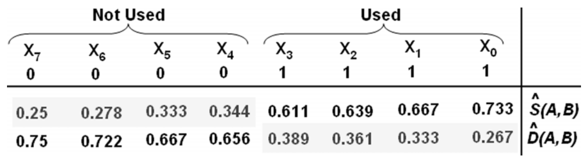
The above proportionality constants θ_1 and θ_2 can be used to give more or less weight for specific correspondences. For the sake of simplicity, we shall assume that there is no such preference and $\theta_1 = \theta_2 = 1$.

Now we can define the aggregated cost for an assignment. Our objective function for selecting the best assignments will be to minimize the overall penalty for making the wrong decisions. Given an assignment $\mathbb{A}$ having a set K_1 of used correspondences (i.e., bits set to 1) and a set K_2 of unused edges (i.e., bits set to 0), we define the cost of $\mathbb{A}$ to be:

$$cost(\mathbb{A}) = \frac{1}{n}\left(\sum_{e \in K_1} \hat{D}(A,B) + \sum_{e \in K_2} \hat{S}(A,B)\right) \qquad (1)$$

where n is the total number of non-zero correspondences (i.e., bits in $\mathbb{A}$). Note that $cost(\mathbb{A})$ is always a number between 0 and 1.

Figure 7 shows an example assignment $\mathbb{A} = [00001111]$ for the concept graphs of Figure 4, where for clarity we show both the similarity $\hat{S}(A,B)$ and the distance $\hat{D}(A,B)$. Applying equ. 1, we obtain $cost(\mathbb{A}) = \frac{1}{8}$ [$(0.267 + 0.333 + 0.361 + 0.389) + (0.344 + 0.333 + 0.278 + 0.25)$] $= 0.319$.



Not Used				Used				
X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0	
0	0	0	0	1	1	1	1	
0.25	0.278	0.333	0.344	0.611	0.639	0.667	0.733	$\hat{S}(A,B)$
0.75	0.722	0.667	0.656	0.389	0.361	0.333	0.267	$\hat{D}(A,B)$

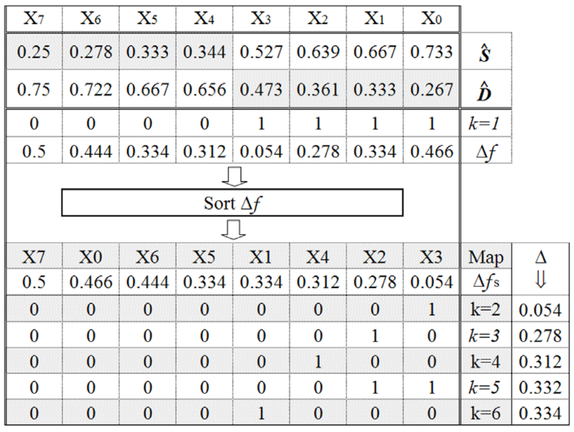
Figure 7: Assignment cost calculation

The cost model we adopt here is very basic, and it can be refined to account for additional factors. For example, it can be modified to differentiate between the actual ways of using a correspondence, that is, to differentiate between the cost of merging two concepts and the cost of adding a has relationship between the concepts. The cost model can also be refined to account for how the schemas themselves will be used, by taking user constraints or query workloads into account. Nevertheless, our overall framework is independent on the concrete choice of a cost function, as various cost functions can be plugged into our top-k algorithm for the generation of the best assignments. Once we obtain the best assignments (with the top-k algorithm), we shall come back to the directed similarities, when deciding on the actual structure of the integrated concept graph that results from each assignment.

4.2 Top-K Algorithm

In general, assignment problems can be solved using, for example, the Munkres algorithm [14] or other related top-k algorithms [8, 16]. These papers show that assignment problems can be generally solved in polynomial time, in cases that satisfy a $(1:1)$ mapping cardinality constraint, which means in our terms, that one concept in one schema can only be combined with at most one concept in the other schema. This is a serious limitation since, in many practical cases, a concept in a schema can have correspondences (and can be combined) with multiple concepts in another schema. The study in [7] proposed formalizing schema matching problems as assignment problems. It gives an algorithm that, given two input schemas and a set of similarities between their elements, generates the top-k schema matchings, in cases where valid solutions satisfy a $(1:n)$ mapping cardinality constraint. This constraint means that a single element in the first schema can be mapped to multiple elements in the second schema, but a single element in the second schema can only be mapped to a single element in the first schema. This assumption is also a strong limitation and prevents us from using such algorithm.

We now start describing our algorithm for top-k enumeration. The initial step is to calculate the first, optimal assignment $\mathbb{A}_1$, which minimizes the cost. Let e_i be the correspondence represented by bit i in the assignment. Let $\hat{S}_i$ and $\hat{D}_i = 1 - \hat{S}_i$ be the (undirected) similarity and distance associated with e_i. Based on the cost formulas in the previous subsection, if $\hat{S}_i \geq \hat{D}_i$, the

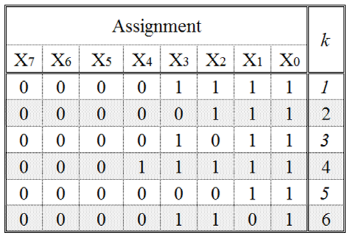


X7	X6	X5	X4	X3	X2	X1	X0	
0.25	0.278	0.333	0.344	0.527	0.639	0.667	0.733	$\hat{S}$
0.75	0.722	0.667	0.656	0.473	0.361	0.333	0.267	$\hat{D}$
0	0	0	0	1	1	1	1	*k=1*
0.5	0.444	0.334	0.312	0.054	0.278	0.334	0.466	Δf

Sort Δf

X7	X0	X6	X5	X1	X4	X2	X3	Map	Δ
0.5	0.466	0.444	0.334	0.334	0.312	0.278	0.054	Δf_s	⇓
0	0	0	0	0	0	0	1	k=2	0.054
0	0	0	0	0	0	1	0	*k=3*	0.278
0	0	0	0	0	1	0	0	k=4	0.312
0	0	0	0	0	0	1	1	*k=5*	0.332
0	0	0	0	1	0	0	0	k=6	0.334

(a)



Assignment								*k*
X7	X6	X5	X4	X3	X2	X1	X0	
0	0	0	0	1	1	1	1	*1*
0	0	0	0	0	1	1	1	2
0	0	0	0	1	0	1	1	*3*
0	0	0	1	1	1	1	1	4
0	0	0	0	0	0	1	1	*5*
0	0	0	0	1	1	0	1	6

(b)

Figure 8: (a) Initial steps in the top-k enumeration process, (b) top 6 assignments

cost of including e_i in an assignment is smaller than the cost of excluding it. Thus, the optimal assignment (of minimum cost) must necessarily have the bit i set to 1. Conversely, if $\hat{S}_i < \hat{D}_i$, the cost of including e_i in an assignment is larger than the cost of excluding it. Thus, the optimal assignment must necessarily have the bit i set to 0. As it can be seen, it is immediate to derive the optimal assignment $\mathbb{A}_1$. For our example, the assignment shown in Figure 7 is the optimal assignment.

The challenge is to efficiently derive the next $k-1$ best assignments. That is, given the jth assignment, we need to decide which bits to "flip" in the assignment in order to obtain the $(j+1)$th best assignment. Let us define $\Delta f_i = |\hat{S}_i - \hat{D}_i|$, to quantify the cost impact of flipping (i.e., complementing) the bit i from its current value in the optimal assignment $\mathbb{A}_1$. Let us define the vector $\Delta f = [\Delta f_{n-1}, ..., \Delta f_0]$. For our example, the top section of Figure 8(a) shows the optimal assignment (in the line with $k = 1$) and the vector Δ_f (in the next line). For each i, Δf_i represents the *increase in cost* with respect to $cost(\mathbb{A}_1)$ if the bit i (i.e., variable X_i) in $\mathbb{A}_1$ were to be flipped. We next sort the Δf vector in increasing order. Let us denote the sorted vector as Δf_s. Moreover, a vector Map keeps the association (after sorting) between elements of Δf_s and the variables X_i. See Figure 8(a) for an illustration of Map and Δf_s.

In order to generate the next best assignments, we explore, *incrementally*, modifications of the original assignment $\mathbb{A}_1$. The incremental modifications are based on the sorted vector Δf_s of cost increases. The goal, at each iteration, is to discover the next assignment that minimizes the increase in cost (as compared to the current assignment). To give an intuition, let us look at Figure 8(a) again. It can be seen that the 2nd best assignment (shown in the line for $k = 2$) can be obtained by just flipping the bit X_3, since this will give the least cost increase (according to Δf_s). Next, to compute the 3rd best assignment, we need to change the variable with the next cost increase (i.e., X_2) and leave X_3 unflipped (relative to $\mathbb{A}_1$). To obtain the 4th best assignment, we have two choices. We could either flip variable X_4 (which is the variable with the next cost increase) and leave the rest as in $\mathbb{A}_1$, or flip both X_2 and X_3. In order to decide which choice gives the smaller cost increase, we need to actually calculate the costs for the two choices (i.e., 0.312 vs. $0.278 + 0.054$) and take the minimum.

As the example suggests, the key ingredients behind our algorithm are the following:

- To find the top k assignments, it is enough to consider combinations of flipping or not flipping the lower $k-1$ variables in the Map vector.
- Furthermore, at any point in the algorithm, there are at most *two* new alternatives to be considered. To discover the next solution, it is enough to compare the new alternative(s) with the best candidate in a candidate set. The assignment with minimum cost increase becomes the next solution, while the rest will be re-evaluated in the next iteration.

In the algorithm, we make use of bit vectors of the form $F = [f_{n-1}, ..., f_0]$ to keep track of the bit flips as compared with the best assignment $\mathbb{A}_1$. Each vector F encodes an assignment as follows. If $f_i = 1$, then the variable in the ith position in the Map vector is flipped relative to its value in $\mathbb{A}_1$. Alternatively, if $f_i = 0$ then the same variable is left unchanged. The algorithm iteratively outputs the top-k assignments in increasing cost in the form of bit vectors $F = [f_{n-1}, ..., f_0]$. The actual assignments can be found immediately based on F, Map and $\mathbb{A}_1$.

Additional Vector Notation When describing a bit vector F, it is convenient to drop the most significant bits that are set to 0. We refer to such a representation as the *condensed* representation. We denote by $len(F)$ the size (or the length) of the condensed representation of F. As an example, the condensed representation of F = [00000010] is [10]; moreover, $len(F)$ = 2. (By convention, we take the condensed representation of a bit vector with all zeros to be [0].) We also make use of a "reverse" operation that expands a bit vector. If F is a bit vector in condensed representation, and m is such that $len(F) \leq m$, we denote by $e_m(F)$ the *expansion* of F on m bits. Such expansion is obtained by adding 0's in front of the leading 1, so that the size of the resulting vector is m. Finally, we use $||$ to denote the concatenation of a bit vector in front of another. For example, $[1] \,||\, [01] = [101]$.

Algorithm 1 gives the exact steps for enumerating the top-k assignments. The two data structures used are: a list "*cand*" of candidates and a list "*soln*" of solutions generated so far. The list *cand* is implemented as a priority queue, so that its operations (insertion, deletion, and taking the minimum) have all logarithmic complexity.

In lines 1 and 2, $soln$ is initialized with the top 2 assignments, and $cand$ is initialized with the 3^{rd} best assignment. (Recall that we manipulate F-vectors.) The **for** loop (lines 3-18) iteratively outputs the i^{th} best assignment starting from $i = 3$. In lines 4-7, the candidate c of minimum cost increase in $cand$ is output as the i^{th} solution, and then moved from $cand$ to the end of $soln$.

Algorithm 1 Enumerating the top k assignments

```
1: $soln \Leftarrow \{[0], [1]\}$
2: $cand \Leftarrow \{[10]\}$
3: for $3 \leq i \leq k$ do
4:     $c = \min(cand)$
5:     Output $c$ as the $i^{th}$ best solution
6:     Remove $c$ from $cand$
7:     Add $c$ at the end of $soln$
8:     Write $c$, in condensed representation, as $[1] \parallel e_m(s)$. (Here, $len(c)$ is $m+1$, and $s$ is the condensed representation of the "lower" $m$ bits of $c$).
9:     Let $next_m(s)$ be the first element $s'$ in $soln$ that is after $s$ and satisfies $len(s') \leq m$.
10:    if $next_m(s)$ exists then
11:        $candidate = [1] \parallel e_m(next_m(s))$
12:        Insert $candidate$ in $cand$
13:    end if
14:    if $s = [0]$ then
15:        $candidate = [10] \parallel e_m(s)$
16:        Insert $candidate$ in $cand$
17:    end if
18: end for
```

In lines 8 to 17, two new possible candidates are calculated based on the current best solution c and the solutions found so far. The bit vector c, assumed to have $m + 1$ bits, is first decomposed into two parts: the highest significant 1, and the remaining lower m bits whose condensed representation is denoted by s. Next, we look for $next_m(s)$, which is the first vector s' in $soln$ that succeeds s (i.e., has higher cost) and satisfies $len(s') \leq m$. In particular, s' is the first higher cost solution (after s) that can still be written on m bits. Note that $next_m(s)$ may not necessarily be the immediate successor of s. Also, note that $next_m(s)$ may not exist.

In lines 10-13, if $next_m(s)$ is found, a new candidate is generated and added to $cand$. This new candidate is obtained by replacing s with $next_m(s)$ within the lower m bits of c. Additionally, in lines 14-17, if all the lower m bits of c are 0, a new candidate is generated by advancing the leading 1 by one position. Intuitively, these are the two possible ways of generating new candidate assignments, while minimally increasing the cost.

Example Let us follow the first three iterations of the algorithm for our running example (see Figure 8 again). The solution set $soln$ is initialized with $\{[0], [1]\}$, and the candidate set $cand$ is initialized with $\{[10]\}$.

- candidate $c = [10]$ is the only entry in $cand$ and, hence, it is the best. Thus, c becomes a solution (the 3rd), and it is moved from $cand$ to the end of $soln$. Two new candidates are placed into $cand$: [11], obtained by replacing [0] in c with $next_1([0]) = [1]$, and [100], obtained by advancing the leading 1 in c. The candidate set is now $cand = \{[11]_{0.332}, [100]_{0.312}\}$, where we write as a subscript the associated cost increase for each candidate (based on Δf_s).
- the best candidate c is now [100], since $0.312 < 0.332$. Thus, c becomes the 4th solution, and it is moved from $cand$ to $soln$. As before, two new candidates are placed in $cand$: [101], obtained by replacing [00] in c with the expansion on 2 bits of $next_2([0]) = [1]$, and [1000], obtained by advancing the leading 1 in c. The set $cand$ is now $\{[101]_{0.366}, [1000]_{0.334}, [11]_{0.332}\}$.
- the best candidate c is now [11]. This entry is removed from $cand$ and becomes the 5th solution. Note that c, in this case, will not generate any new candidates. First, there is no $next_1([1])$ in $soln$, since all solutions following [1] have len greater than 1. Moreover, c does not pass the test in line 14. The candidate set becomes $cand = \{[101]_{0.366}, [1000]_{0.334}\}$ (and [1000] will become the $6th$ solution in the next iteration).

4.2.1 Algorithm Analysis

Algorithm Correctness We can prove the correctness of the top-k algorithm by induction. For simplicity of the presentation, we shall assume that there are no ties among the assignments (i.e., no two assignments have the same cost). If ties occur, the same proof of correctness applies but with slight extensions (essentially, we must define a certain order among the assignments with the same cost, and then observe that the algorithm outputs ties in that order).

For $k = 1$, 2, and 3, the algorithm is correct by initialization. For $k > 3$, assume that the algorithm has generated the first $k - 1$ solutions $F_1, \cdots F_{k-1}$. We show that the solution with the k-th best cost, F_k, must be in the candidate set $cand$ at the beginning of the k-th iteration of the algorithm. Therefore, F_k will be generated as the k-th solution by the algorithm. Let us write F_k, in condensed representation, as $[1] \parallel e_m(s)$. We assume here that $len(F_k)$ is $m + 1$, and s is the condensed representation of the lower m bits of F_k. There are two cases to consider.

Case 1: $len(F_k) > max_{1 \leq i \leq k-1}(len(F_i))$. Since F_k is the first (i.e., lowest-cost) solution of its length, it must be that $s = [0]$. (We make use here of an obvious monotonicity property on assignments, and of the assumption of no ties.) Consider now the assignment $c = [1] \parallel e_{m-1}([0])$, which is the same as F_k except that the leading 1 occurs in a position that is lower by one. Since $cost(c) < cost(F_k)$, it must be the case that c is one of the previous $k - 1$ solutions. Hence, in an earlier iteration, the algorithm must have added F_k to $cand$ (lines 14-17).

Case 2: $len(F_k) \leq max_{1 \leq i \leq k-1}(len(F_i))$. Thus, there is an earlier solution F_l such that $len(F_k) \leq len(F_l)$. In turn, this implies that there is a solution F_i (with $i \leq l$) such that $len(F_i)$ is exactly the same as $len(F_k)$. This can be shown by observing that solutions do not "skip" any length number, that is, if there is a solution of length p then there is another (lower-cost) solution of length $p - 1$. Essentially, the length can be reduced by 1 as follows: a vector of the form $[10\ldots]$ can be replaced by $[01\ldots]$, while a vector of the form $[11\ldots]$ can be replaced by $[01\ldots]$. In each case, the resulting vector has a lower cost.

Since $len(F_i) = m + 1$, it must be that F_i is of the form $[1] \parallel e_m(s_i)$ where s_i is some bit vector in condensed representation such that $len(s_i) \leq m$. Note that both s and s_i must be among the top $k - 1$ solutions, since their respective costs are smaller than F_k and F_i. Furthermore, since $cost(F_k) > cost(F_i)$, and F_k and F_i are identical on the leading bit, it must be that $cost(s) > cost(s_i)$. This implies that s_i appears in $soln$ before s. Let s' be the nearest solution in $soln$ that precedes s and has length smaller than m. We know that such s' exists, because s_i itself precedes s and has length smaller than m. Note that $next_m(s') = s$.

Let us now form the vector $F' = [1] \parallel e_m(s')$. Since $cost(s') < cost(s)$, we have that $cost(F') < cost(F_k)$. Thus, F' is one of the top $k - 1$ solutions. This fact, combined with the fact that $next_m(s')$ exists, implies that in an earlier iteration, the algorithm must have added $[1] \parallel e_m(next_m(s'))$ to $cand$ (lines 10-13). But this last vector is precisely F_k. This concludes the proof.

Complexity Analysis First of all, the size of $cand$ is $O(k)$. This is because at each iteration, we take out one element and put at most two new candidates. So, the list increases by at most 1 at each iteration. Then, at each iteration, the complexity of finding the candidate with minimum cost (line 4) is $O(\log k)$ (given a priority queue implementation for $cand$). Determining $next_m(s)$ requires

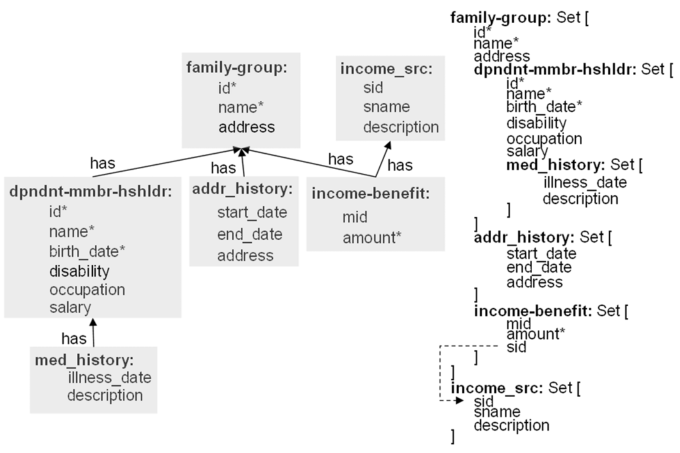

family-group: id* name* address
income_src: sid sname description
has
has
has
has
dpndnt-mmbr-hshldr: id* name* birth_date* disability occupation salary
addr_history: start_date end_date address
income-benefit: mid amount*
has
med_history: illness_date description
family-group: Set [
id*
name*
address
dpndnt-mmbr-hshldr: Set [
id*
name*
birth_date*
disability
occupation
salary
med_history: Set [
illness_date
description
]
]
addr_history: Set [
start_date
end_date
address
]
income-benefit: Set [
mid
amount*
sid
]
]
income_src: Set [
sid
sname
description
]

Figure 9: Combined concept graph and schema ($\lambda = 0.53$).

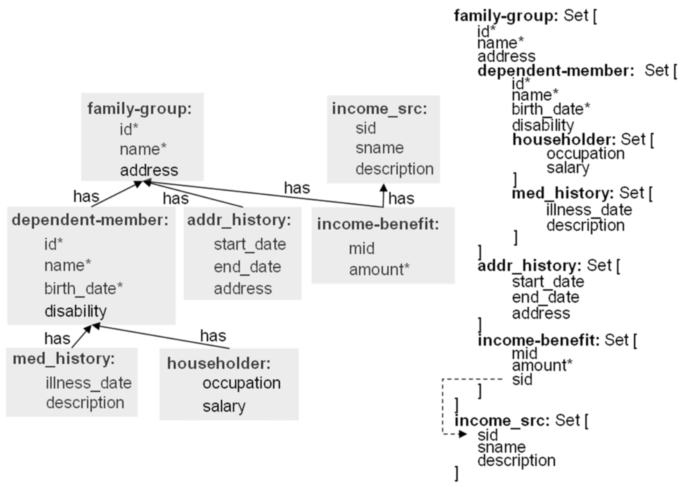


Figure 10: Combined concept graph and schema ($\lambda = 0.63$).

$O(k)$ in the worst case, since it requires traversing the $soln$ list, and the size of $soln$ is at most k.

So, each iteration takes $O(k)$ and there are $O(k)$ iterations. Hence, the overall complexity is $O(k^2)$. Since the initial time to sort the Δf_s vector is $O(n \log n)$, we obtain an overall complexity of $O(n \log n + k^2)$.

Note that the cost of obtaining the next solution is only $O(k)$ at each step. This is important, since it is often the case that we do not need to generate all the top k solutions at once, but rather generate them lazily, one by one, as the user interaction demands it. In such case, the time to obtain the next solution is the important one.

4.3 Lambda Tuning

We now consider the problem of determining the parameter λ that controls how to combine concepts (via $merge$ or via has edges) once the top-k assignments have been generated. Note that assignments already specify which correspondences to be used; what is missing is how to use them.

As seen from the rules for combining concepts (see table 2), the choice of the parameter λ is critical, as it directly affects the generated schema. Figure 9 shows the combined concept graph when $\lambda = 0.53$. Note that $family$ and $group$ are merged into one concept, as well as $income$ and $benefit$; moreover $dependent$, $member$ and $householder$ are also merged. Figure 10 shows the combined concept graph when $\lambda = 0.63$. It illustrates the effect of introducing a has edge from $householder$ to $member$ rather than merging $householder$ with $member$, due to the increase in λ. (Note that $dependent$ is merged with $member$, so the has edge from $householder$ goes into the merge of $member$ and $dependent$). Figures 9 and 10 also show the corresponding integrated schemas.

Note that as λ decreases, the number of correspondences that are used for $merge$ will increase, thus resulting in a more *"compact"* integrated schema. In the extreme case of $\lambda = 0$, all the selected correspondences will be used for merge. On the other hand, increasing λ will limit the number of correspondences that are used for $merge$ to those connecting tightly related concepts, thus, resulting in a more "normalized" integrated schema.

From the database normalization perspective, it is appealing to increase the value of λ. However, a value for λ that is too high runs the risk of violating some of the assignments generated by the top-k algorithm. In particular, if λ is too high, there may be an assignment (in the top-k) requiring that a certain correspondence be used. However, because λ is too high, none of the rules in Table 2 apply; hence the concepts cannot be combined, even though the correspondence has been selected by the assignment.

An immediate approach to address this problem is to set λ to the highest possible value such that the decisions encoded by the top-k assignments can all be satisfied. Concretely, let $\mathbb{E}$ be the set of correspondences that are selected by at least one of the top-k assignments. Denote the two directed similarities associated with a correspondence in $\mathbb{E}$ as $\hat{s}_1$ and $\hat{s}_2$. The steps for setting λ are:

1. iteratively scan all the correspondences in $\mathbb{E}$,
2. for each such correspondence, record $\max(\hat{s}_1, \hat{s}_2)$ and add this value to a list L, and finally
3. set λ to be the minimum of the values in L.

The above procedure guarantees that all the correspondences used in any of the top-k assignments will be taken into consideration, when combining the concepts with the chosen λ. Note that this is the maximum possible value for λ that can be obtained without ignoring any of those correspondences.

Stability Analysis Note that the number of the top candidates k may affect λ tuning. Specifically, increasing k may decrease λ if any of the additional candidates uses a new correspondence having $\max(\hat{s}_1, \hat{s}_2) < \lambda$. So, the problem of tuning λ can be replaced by a new problem of tuning k. An alternative is to generate a larger number of solutions, k', and perform stability analysis to identify the correspondences that are the most stable throughout the solution set. The parameter λ can then be set to satisfy the constraints of the most stable correspondences. Then λ will not be as much dependent on the value of k, but on the correspondences that have the highest probability to be included in the final solution. For this purpose we employ a stability analysis method similar to the one proposed in [7] and test it on real and synthetic data. Assume $\mathbb{E}$ represents the set of correspondences used in any of the top-k' assignments. Our stability analysis counts how many times a correspondence in $\mathbb{E}$ appears in the top-k' assignments. A correspondence that appears a sufficient number of times will be part of the set of most stable edges. Otherwise, it will be ignored.

5. COMPUTING CONCEPT SIMILARITIES

In this section we introduce *concept similarity*, a directed similarity measure for concepts and concept graphs. In essence, this is a variation of the well-known Hausdorff distance [15]. The concept similarity measure quantifies the distance between concepts by incorporating both the attributes within concepts and the references to other concepts. Other techniques for calculating similarities/distances between schema elements could also be envisioned [22, 10, 2]. The similarity measure we define and implement here has two appealing characteristics: its simplicity and the fact that it is a straightforward extension of a widely studied distance measure (the Hausdorff distance).

Hausdorff Distance We start by reviewing the basic notions related to the Hausdorff distance. We shall use $\hat{d}(a,b)$ to denote the distance between two objects a and b; correspondingly, $\hat{s}(a,b) = 1 - \hat{d}(a,b)$ represents the similarity between a and b. Both measures are asymmetric (e.g., $\hat{d}(a,b)$ is not necessarily equal to $\hat{d}(b,a)$) and vary in the range [0,1]. The distance between an object a and a *set* of objects $B = \{b_1, ..., b_{N_B}\}$ is defined as:

$$\hat{d}(a,B) = \min_{b \in B} \hat{d}(a,b) \tag{2}$$

There are several ways to define the directed distance between two object sets $A = \{a_1, ..., a_{N_A}\}$ and $B = \{b_1, ..., b_{N_B}\}$ as shown in [9]. The study in [6] proposed the following variation of the Hausdorff distance:

$$\hat{d}(A,B) = \frac{1}{N_A} \sum_{a \in A} \hat{d}(a,B) \tag{3}$$

The complement of the above formula (i.e., the similarity $\hat{s}(A,B) = 1 - \hat{d}(A,B)$) represents a measure that quantifies the degree to which the set A is covered by the set B. This measure indicates the degree to which A can be considered a "subset" of B. This similarity measure varies in the range $[0,1]$, where 1 denotes complete "inclusion".

5.1 Concept Similarity

We now adapt the notion of Hausdorff distance between sets of objects, to define distances and similarities between concepts. Let $\alpha(X)$ denote the collection of $attributes$ in a concept X, and let $\beta(X)$ denote the collection of concepts to which X has direct references (i.e., has edges). N_α and N_β are the numbers of $attributes$ and, respectively, the number of outgoing has edges of concept A, and where $d(a,b)$ is 0 if there is a correspondence between the attributes a and b, and 1 otherwise. For the moment we assume that has edges do not form cycles, a case that is discussed separately later in this Section. We can now define the concept distance that corresponds to equ. 2 by the following:

$$\hat{d}(a,B) = \min\left(min_{b \in \alpha(B)} d(a,b), min_{B' \in \beta(B)} \hat{d}(a,B')\right) \tag{4}$$

and the concept version of equ. 3 as:

$$\hat{d}(A,B) = \frac{1}{N_\alpha + N_\beta}\left(\sum_{a \in \alpha(A)} \hat{d}(a,B) + \sum_{A' \in \beta(A)} \hat{d}(A',B)\right) \tag{5}$$

Note that both of these definitions are recursive.

Since our framework uses similarities between concepts rather than distances, we need to transform these calculations slightly. The corresponding similarity measures can be easily computed by using:

$$\hat{s}(a,b) = 1 - \hat{d}(a,b) \quad \hat{s}(a,B) = 1 - \hat{d}(a,B)$$
$$\hat{s}(A,B) = 1 - \hat{d}(A,B)$$

Intuitively, a concept similarity should be lower if the attributes of the corresponding concepts are covered via other referenced concepts and not directly. To achieve this intended behavior, we modify the above computation as follows. Assume that we need to compute the similarity between concepts A and B. Then, we keep track of the number δ_A of has edges that we follow from A (with each recursive call to equ. 5). Similarly, we keep track of the number δ_B of has edges that we follow from B (with each recursive call to equ. 4). Whenever we reach the fixed point in the recursion where we need the similarity $\hat{s}(a,b)$ between attributes a and b (see equ. 4) we use a scaling factor κ to alter this similarity, based on the *difference* between δ_A and δ_B. More precisely, we take $\kappa = \frac{\alpha}{\alpha+\delta}$, where $\delta = |\delta_A - \delta_B|$ and where α is an application dependent parameter. We then alter the similarity between a and b by using a weighted similarity formula $\hat{s}(a,b) = \kappa \times s(a,b)$.

The intuition behind κ is that we want to penalize the similarity between attributes when the attributes reside in concepts that have *different* depths relative to the original concepts A and B that are being compared. This guarantees in particular that the similarity between two identical concept graphs will always be 1. The behavior of κ against δ with different α values is depicted in figure 11. It can be seen that as α increases the penalty with depth distances decreases, and vice versa.

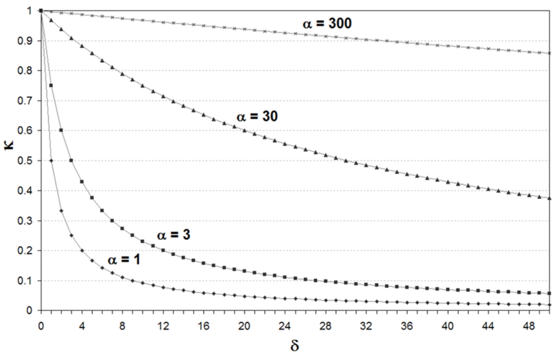


Figure 11: The behavior of κ as α varies.

In our experiments in section 6, we shall use $\alpha = 1$, due to the fact that the hierarchy depth of the concepts in the tested schemas is not too large (5 or 6, at most). Thus, we increase the penalty quicker, even for low values of δ.

To calculate the complete set of similarities, we iteratively calculate the similarity between each pair of concepts across input graphs. For our ongoing example, Table 1 shows the similarities associated with the set of concept correspondences between each concept of S_1 and each concept of S_2. These similarities are also shown in Figure 4 on top of the dotted lines connecting concepts across schemas. Additionally, the minimum of the directed similarities is shown below the correspondences (this is the undirected version $\hat{S}(A,B)$ as described earlier).

For our example, the similarity numbers show that $householder$ matches better (with higher similarity) with $member$ than with $benefit$. This is the result we would intuitively expect (since there are more attributes in common between $householder$ and $member$ than between $householder$ and $benefit$). As a result, in the integration process, we will assign higher weight to the schemas that combine $householder$ with $member$ than to the ones that combine $householder$ with $benefit$.

It can be also seen that the directed similarity from $member$ to $householder$ is 1 based on the fact that $member$ as well as its "superconcepts" (i.e., $group$ in this case) are covered by $householder$ and its "superconcepts" (i.e., $family$).

Note that concepts that are "below" a given concept (i.e., "subconcepts" that have paths of has edges leading *to* the given concept) do not affect the similarity calculation. For example, the subconcept $med\text{-}history$ is not taken into account by the similarity calculation for $member$. One reason for this design choice is that, intuitively, a has edge from A to B represents a *strong* relationship from A to B but not necessarily from B to A. For example, an instance of $member$ cannot exist without an instance of $group$; however, it can exist without an instance of $med\text{-}history$. To capture this semantics, only direct or indirect superconcepts are involved in similarity calculations in our method.

Finally, we note that the method for computing directed similarities between concepts that we presented in this section is completely orthogonal to the overall method for schema integration. In particular, other methods can be easily plugged in; for example, one could develop methods that involve both superconcepts and subconcepts, or could adapt existing algorithms (e.g. [10]) to to calculate directed similarities between concepts.

Handling Cycles Recall that our concept similarity measure is calculated recursively. Thus, cycles in the input concept graphs can cause problems for the computation of concept distances. Cycles arise naturally in practice (e.g., an employee has a department, and a department has an employee which is the manager). To handle cycles in our implementation, we incorporate a cycle detection technique, which keeps track of the visited concepts during recursion. When a cycle is detected, we interrupt the repetitions of the cycle when the difference between two successive calculations ($\Delta\hat{s}$) becomes smaller than a considerably very low value ϵ. From the recursive nature of the formulas there is a guarantee that $\Delta\hat{s}$ converges (i.e., $\forall iter(\Delta_{iter}\hat{s} > \Delta_{iter+1}\hat{s})$), but since there is no guarantee on the rate of the convergence, for performance issues we interrupt the cycle after a predefined number $iter_{max}$ of iterations.

6. EXPERIMENTAL EVALUATION

We evaluate our integration approach using a number of real world and synthetic scenarios. In section 4 we showed that our top-k generation algorithm has low complexity, which makes it amenable for an efficient implementation. In this section we verify this on our implementation; we analyze the performance of generating the top-k schemas for a number of scenarios, and also evaluate the usability and effectiveness of the integration process based on a user study. Furthermore, we evaluate the tuning method for the integration parameter λ, by varying the number of top configurations and analyzing the effects on the final results. Our system is implemented using Java and the experiments were carried on a PC compatible machine, with Intel Core Duo processor (1.8GHz), running Windows XP and JRE 5.0.

6.1 Real Integration Scenarios

We test the performance and effectiveness of our method in a number of integration scenarios [4]. The schemas used in the experiment are: a relational and an XML schema, each representing gene expression experimental results (GENEX); two XML schemas representing enterprise business objects (such as orders, and customers related to orders), one from SAP and the other one from the IBM WebSphere Business Integration (WBI) suite; a relational and an XML version of the Mondial database [13]; two relational schemas from the Amalgam integration benchmark [11] for bibliographic data; the first schema in the Amalgam benchmark and a DBLP schema (obtained from the DBLP website); and two XML schemas, each with a different nesting structure, representing information about departments, projects and employees.

Figure 12 shows the characteristics of these scenarios, including the number of concepts, the number of correspondences between attributes, as well as the number of non-zero correspondences between concepts (i.e., correspondences between concepts that have non-zero similarity in at least one direction). In the fifth column of the table, we give the total number of integrated schemas (i.e., the size of the space of candidate schemas) in each scenario. The last portion of the table indicates the average time per generated schema when using the top-k schema integration method. This time is obtained by running the top-k algorithm for several values of k and then reporting the average time between two consecutive solutions (for each fixed k).

Although the numbers of concepts and correspondences in these scenarios are not large, the schemas are not necessarily trivial. For example, the SAP schema is split over 15 files with concepts having tens (20-50) of attributes and multiple levels of nesting. Moreover, many of the concepts are referred to from other concepts. One reason for why there is only a small number of concept correspondences (seven in the WBI-SAP integration scenario) is that not every concept in one schema has a match in the other schema. Nevertheless, the size of the space of candidate schemas in these scenarios is sufficiently large to illustrate the need for an effective tool to explore all the integration choices. For larger schemas, the effectiveness of our top-k enumeration method would become even more apparent.

As it can be seen, the average time to generate the next schema is consistently low, even when k is large (e.g., 128). Note that in practice, in an interactive system, we will seldom need to generate such a large number of schemas (see also the next subsection).

We also observed from the experiment that the top schemas generated by the algorithm make the "correct" decisions for a large fraction of the bits (or, variables) in the assignment vector. In particular, the highly similar concepts (with many attributes in common) are combined, while the highly dissimilar concepts are not combined. In a sense, these are the "easy" decisions that an expert may often agree with. This leaves out the "difficult" cases, involving groups of "ambiguous" concepts for which any combination may be possible.

To illustrate, consider the Mondial scenario. There are 18 correspondences between concepts. Most of them (14, precisely) involve concepts that are semantically the same (e.g., *river*, *sea*, *lake*, *city*, which occur in both input schemas). However the concept $country$ in one of the schemas can be combined with any subset of COUNTRY, ECONOMY, and POPULATION in the other schema. This is reflected in the respective similarity numbers which are very close to 0.5. The reason for this "ambiguity" is that $country$ in the first schema is an unnormalized relation that includes attributes specific to economy and population, which are stored separately, in different relations, in the second schema. The schemas at the top of our ranking will enumerate all possible choices of combining these concepts (from the most normalized to the most unnormalized), while making the obvious choice for the rest of the concepts. In contrast, the schemas that are at the bottom of the ranking will not combine some of the concepts that are obviously equivalent.

In the next subsection, we shall make a more precise qualitative argument, via a user study.

6.2 User Experiment

In general, the best assignments generated by the top-k algorithm may not always yield the schema that is "best desired" by a particular expert, although they may yield a good approximation. Thus, a human expert is still required for the schema integration process. Our goal is to minimize the human effort rather than eliminate it.

Integration Scenario	Attribute Corresp.	Source Concepts	Concept Corresp.	Total Integr. Schemas	Time/schema (ms) (top-64)	(top-128)	(top-256)
Genex	31	13	6	64	0.50	n/a	n/a
WBI-SAP	46	22	7	128	1.70	2.44	n/a
Mondial	74	49	18	>3000	1.43	1.72	2.13
Amalgam	16	24	12	>3000	0.95	1.12	1.43
Amalgam-DBLP	26	29	11	2048	0.85	0.98	1.10
Dept-Proj-Emp	16	10	8	192	0.22	0.25	n/a

Figure 12: Evaluation on real schema integration scenarios.

In this subsection, we shall evaluate the effectiveness of our top-k algorithm as part of an *interactive tool* that helps a human expert reach the "best desired" schema.

The tool works as follows. It first generates the (unconstrained) top-k schemas according to the initial input (schemas and correspondences). The user then inspects a few of the top schemas (the first, the second, etc.) and adds, if needed, one or more *constraints* in the sense of [4]. These constraints are a mechanism through which a human expert can "fix", or override, the undesired choices made by the top schemas. Next, the system reacts by generating the (new) top-k schemas that satisfy the constraints. The process continues, with the user possibly adding more constraints, and so on, until the "best desired" schema is obtained.

There are two types of user constraints that we allow: "Always use correspondence X", and "Never use correspondence X".[3] The first constraint requires that the pair of concepts connected by correspondence X must always be combined. It is equivalent to fixing the bit for X (in the assignment vector) to be always 1. Conversely, the second constraint requires the correspondence X to be ignored. Thus, it is equivalent to fixing the bit for X (in the assignment vector) to be always 0. Note that, with each such constraint, the space of the candidate schemas is reduced by half.

The regeneration of the top-k schemas is done by re-invoking the top-k algorithm, after taking out the fixed variables from the assignment vector. We note that, in the tool, we generate the top-k schemas only one at a time, as demanded by the user. (Thus, the time per schema is more important than the time to generate all top-k schemas.)

To evaluate the effectiveness of the resulting tool, we conducted a *user study* to evaluate the number of user interaction steps needed to reach the "best desired" schema. We had four users, which are researchers in our department and are database experts. Nevertheless, they required a few hours to go through the three scenarios we selected for this experiment, since they needed to understand the semantics of the schemas (and the domain). The results of our user study are summarized in Figure 13.

	Genex (constr.)	(schemas)	Mondial (constr.)	(schemas)	Dept-Proj-Emp (constr.)	(schemas)
User A	1	2	1	2	2	3
User B	1	2	1	2	1	2
User C	1	2	3	4	2	3
User D	0	3	3	4	0	5

Figure 13: User study in three of the scenarios.

We measure two types of interactive steps in the experiment. First, we count the total number c of constraints that a user has to add, to instruct the tool. Second, we count the total number s of *different* schemas that the user explores, before deciding the final integrated schema. These numbers are not an exact measure of the human effort involved, since they do not include the time to understand the source schemas, or the time to evaluate the integrated schemas. Still, these parameters give an indication of the complexity of the interactive process. As it can be seen, the number of interaction steps is consistently low in all scenarios for all users.

For Genex, the users were consistent and picked the same "best desired" schema. A total of 4 out the 6 correspondences connected semantically equivalent concepts (with many common attributes). The top schemas generated by the tool chose, correctly, to merge these concepts. The users were left to decide how to combine the concepts of $array$ and $measurement$ in one schema, with the concept of $array\_measurement$ (which included features of both array and measurement) from the other schema. All users decided that a good design is to leave $array$ as a separate concept, and to merge $array\_measurement$ with $measurement$. However, they accomplished this in different ways. User D let the tool enumerate the first few schemas and stopped when she realized that the 3rd schema was the desired one. Users A, B and C looked at the first schema, which merged the above three concepts into one, and concluded right there that they did not want $array$ to be merged with $array\_measurement$. Hence, they added a constraint to prevent such merging. The top schema (after regeneration) was then the desired schema.

The Mondial scenario had similar characteristics with Genex. In particular, the "hard" choices involved the $country$ concept (as discussed earlier). Here, the final schema was not the same for all the users. Two of them (C and D) chose to merge all four concepts that had country features (i.e., $country$, COUNTRY, ECONOMY, POPULATION), by explicitly adding three constraints. The other two (A and B) achieved a more normalized schema, by merging $country$ with COUNTRY, but leaving ECONOMY and POPULATION as separate concepts (with has edges to the integrated concept for country). Only one constraint was needed for this.

Finally, the scenario with more "ambiguous" choices was Dept-Proj-Emp, where most of the concepts in each schema could match several of the concepts in the other schema. For example, $employee$ in the first schema could match both $manager$ and emp in the second schema, while $fund$ in the first schema could match both $grant$ and $project$ in the second schema. Users A and D reached the same final schema, although in different ways. Users B and C decided on different schemas. Thus, different users can often have different integrated schemas (and different criteria of goodness) in mind. This confirms, one more time, that an automatic system for schema integration is not realistic. Even so, the tool was helpful in identifying the choices that all the users agreed with (e.g., merge $employee$ with emp, and merge $fund$ with $grant$).

Notably, the schema chosen by User C in this final scenario showed a case where the unconstrained top-k algorithm could not have generated the "best desired" schema (even for a large k). The

[3]These constraints are called, respectively, $Apply(X)$ and $\neg Apply(X)$, in [4].

reason for that is that User C decided that $fund$ should be merged with $grant$ and, additionally, with $project$. The reasoning was that usually there is a one-to-one relationship between a project and a grant; hence, the two concepts could be combined into one. On the other hand, the unconstrained top-k algorithm gives little weight to the choice of merging $fund$ with $project$ since they only have one attribute in common (project name).

Overall, the user study validates the idea that the top-k algorithm identifies many of the correct choices in practice. Thus, the user can focus her attention on the "difficult" cases. This is reflected in the number of user constraints needed to guide the system. This number is relatively small in our experiment.

6.3 Synthetic Scenarios for Stability Analysis

In order to analyze the effect of the number k of solutions used on the value of the integration parameter λ, we used a set of synthetic integration scenarios. More specifically, we studied the frequency of used edges in the top-k assignments, and inspected the change in this frequency as an effect of varying k. Intuitively, edges frequently used in the top-k assignments are more likely to be part of the best assignment. By contrast, rarely used (or unused) edges are less likely to be part of the best assignment.

Intuitively, setting λ based on stability analysis makes sense if edge frequencies quickly stabilize. In other words, the variation of k beyond some value should not significantly affect the calculated frequencies. If such k is small, we can rely just on a few number of top candidates to evaluate the frequencies.

To validate this experimentally, we generated 100 synthetic integration scenarios as follows: we randomly varied the number of input schemas between 2 and 10, then, for each schema, we randomly generated a number of concepts between 3 and 20. Finally we randomly generated the similarities associated with all correspondences. We used a uniform probability distribution function in all random number generations. For each scenario, we calculated the top-k assignments and varied k as $k = 1, 10, 20 \ldots 100$. For each top-k, we calculated the frequency of each used correspondence and then calculated the difference between such frequency and the frequency of the same edge in the preceding top-k group. Figure 14 displays the average for all used correspondences. The value corresponding to $k = 10$ is the average difference in frequencies between the top-10 candidates and the top-1 candidate, while the value corresponding to $k = 20$ is the average difference in frequencies between the top-20 candidates and the top-10 candidates, and so on. It can be seen that, as k increases, the differences in frequencies become less significant.

In conclusion, this experiment validates the idea that λ can be set by varying k and employing stability analysis, rather than based on a single value of k that is given by the user. Nevertheless, once λ is determined, via stability analysis, we still go back and generate the top-k best schemas, where k is the actual limit imposed by the user.

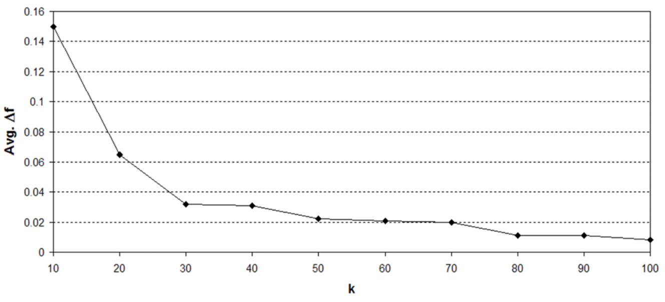


Figure 14: The average change in the frequency of the used edges in the top k schemas, as k varies.

7. RELATED WORK

There are a number of features that distinguish our approach from existing work on schema integration, model merging and ontology merging. In the context of ontology merging, most literature has been primarily focused on the problem of ontology alignment, which is deriving relationships across concepts in different ontologies (see ILIADS [25] as an example). In contrast, the focus of our method is on the ranked exploration of the alternatives for the structural, non-redundant unification of the concepts.

We briefly visited the method of Pottinger and Bernstein [19] in section 1; this method subsumes much of the earlier work on schema integration [1, 3, 23] and also includes merging-specific features that are present in PROMPT [17] and other ontology merging systems, such as FCA-Merge [24]. In the method described in [19], the user must provide in advance a "template" of the integrated schema. In contrast, the input to our method is just a set of atomic correspondences which can be discovered by an automatic schema matching tool. From such input, our system is then able to generate the most "likely" candidate integrated schemas. As in [19] and following [4], our method operates at a logical level, where the schemas are described in terms of concepts and their relationships.

The work of [19] is extended in [20] by considering the schema integration problem in the context of source schemas related by GLAV mappings, as opposed to just correspondences between schema attributes. One possible direction is to investigate whether our ranking and enumeration mechanism can extend to such context.

Our paper builds upon the framework introduced in [4], which reduces the schema integration problem to an enumeration of all possible ways of merging two graphs of concepts. This enumeration defines in a very precise, mathematical sense, the space of candidate schemas. While the system in [4] relies exclusively on user interaction to navigate and explore the space of candidate schemas, the method we propose in this paper is more automatic and relies on top-k enumeration to give higher priority to the more "likely" schemas. In particular, we keep and exploit all the weights generated by the matching algorithms, which enables us to cost the integrated schemas. Furthermore, the use of weights also enable us to define more refined relationships on the integrated schema (i.e., incorporate both *merge* and *has* decisions).

Our formalization of the schema integration problem as a top-k enumeration is different from the general assignment problem [14], and existing techniques (see [14, 8, 16, 7]) do not immediately extend to our context. In particular, as discussed in section 4.2, they limit the ways in which concepts in one schema can be combined with concepts in another schema.

Finally, we note that schema matching techniques have been extensively studied [22, 10, 2]. Our schema integration method is complementary to schema matching, since it uses the outcome of schema matching. Moreover, the emphasis here is on using directed similarities rather than the more common undirected similarities.

8. CONCLUDING REMARKS

In this paper, we aim at reducing the manual effort that is required in generating an integrated schema. Our schema integration method is based on the use of directed and weighted correspondences between the concepts that appear in the source schemas. A key component of our approach is a novel top-k ranking algorithm for the automatic generation of the best candidate schemas.

The algorithm gives more weight to schemas that combine the concepts with higher similarity or coverage. Thus, the algorithm makes certain decisions that otherwise would likely be taken by a human expert. We show that the algorithm runs in polynomial time and moreover it is effective in practice.

One important future direction is to apply and extend schema integration techniques such as the ones developed in this paper to situations where the number of schemas to be integrated is much larger (e.g., in the hundreds). In particular, we would like to be able to automate the problem of generating unified schemas when extracting and integrating large data sets from the web (e.g., from DBPedia, Freebase, etc). Even when the data has structure, it is often the case that the schema underlying an entity of interest varies, sometimes significantly, from one individual object to another, and coming up with a unified schema will pose challenges.

Acknowledgements Popa and Stanoi are partially funded by the U.S. Air Force Office for Scientific Research under contracts FA9550-07-1-0223 and FA9550-06-1-0226. Part of the work was done while Radwan was visiting IBM Almaden Research Center.

We thank the anonymous reviewers for their suggestions and comments on this work, and we also thank our colleagues who helped us with the user experiment.

9. REFERENCES

[1] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Comput. Surv.*, 18(4):323–364, 1986.

[2] P. Brown, P. J. Haas, J. Myllymaki, H. Pirahesh, B. Reinwald, and Y. Sismanis. Toward Automated Large-Scale Information Integration and Discovery. In *Data Management in a Connected World*, pp 161–180, 2005.

[3] P. Buneman, S. B. Davidson, and A. Kosky. Theoretical Aspects of Schema Merging. In *EDBT*, pp 152–167, 1992.

[4] L. Chiticariu, P. G. Kolaitis, and L. Popa. Interactive Generation of Integrated Schemas. In *SIGMOD*, pp 833–846, 2008.

[5] T. M. Connolly and C. E. Begg. *Database Systems: A Practical Approach to Design, Implementation and Management*. Addison-Wesley, 2004.

[6] M.-P. Dubuisson and A. K. Jain. A Modified Hausdorff Distance for Object Matching. In *Proc. Int. Conf. on Pattern Recognition*, pp 566–568, 1994.

[7] A. Gal. Managing Uncertainty in Schema Matching with Top-K Schema Mappings. *J. Data Semantics*, 6:90–114, 2006.

[8] H. Hamacher and M. Queyranne. K-best solutions to combinatorial optimization problems. *Annals of Operations Research*, 4:123–143, 1985/6.

[9] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge. Comparing Images Using the Hausdorff Distance. *IEEE Trans. PAMI*, 15:850–863, 1993.

[10] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In *ICDE*, pp 117–128, 2002.

[11] R. J. Miller, D. Fisla, M. Huang, D. Kymlicka, F. Ku, and V. Lee. The Amalgam schema and data integration test suite. www.cs.toronto.edu/ miller/amalgam, 2001.

[12] R. J. Miller, Y. Ioannidis, and R. Ramakrishnan. The Use of Information Capacity in Schema Integration and Translation. In *VLDB*, pp 120–133, 1993.

[13] www.dbis.informatik.uni-goettingen.de/Mondial.

[14] J. Munkres. Algorithms for the Assignment and Transportation Problems. *Journal of the Society of Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[15] J. R. Munkres. *Topology*. Prentice Hall, Inc., 2000.

[16] K. G. Murty. An algorithm for ranking all the assignments in order of increasing cost. *Operations Research*, 16:682–687, 1968.

[17] N. F. Noy and M. A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *AAAI/IAAI*, pp 450–455, 2000.

[18] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating Web Data. In *VLDB*, pp 598–609, 2002.

[19] R. Pottinger and P. A. Bernstein. Merging Models Based on Given Correspondences. In *VLDB*, pp 826–873, 2003.

[20] R. Pottinger and P. A. Bernstein. Schema Merging and Mapping Creation for Relational Sources. In *EDBT*, pp 73–84, 2008.

[21] A. Radwan, A. Younis, M. A. Hernández, H. Ho, L. Popa, S. Shivaji, and S. Khuri. BioFederator: A Data Federation System for Bioinformatics on the Web. In *IIWeb Workshop*, pp 92–97, 2007.

[22] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10(4):334–350, 2001.

[23] S. Spaccapietra and C. Parent. View Integration: A Step Forward in Solving Structural Conflicts. *Knowledge and Data Engineering*, 6(2):258–274, 1994.

[24] G. Stumme and A. Maedche. FCA-MERGE: Bottom-up merging of ontologies. In *IJCAI*, pp 225–234, 2001.

[25] O. Udrea, L. Getoor, and R. J. Miller. Leveraging Data and Structure in Ontology Integration. In *SIGMOD*, pp 449–460, 2007.