Julian Eberius, Maik Thiele, Katrin Braunschweig, Wolfgang Lehner

Top-k entity augmentation using consistent set covering

Erstveröffentlichung in / First published in:

SSDBM 2015: International Conference on Scientific and Statistical Database Management, La Jolla 29.06. – 01.07.2015. ACM Digital Library, Art. Nr. 11. ISBN 978-1-4503-3709-0

DOI: https://doi.org/10.1145/2791347.2791353

Final edited form was published in "SSDBM 2015: International Conference on Scientific and Statistical Database Management. La Jolla 2015", Art. Nr. 8,
ISBN 978-1-4503-3709-0
https://doi.org/10.1145/2791347.2791353

Top-k Entity Augmentation using Consistent Set Covering

Julian Eberius, Maik Thiele, Katrin Braunschweig, and Wolfgang Lehner
Technische Universität Dresden
[firstname.lastname]@tu-dresden.de

ABSTRACT

Entity augmentation is a query type in which, given a set of entities and a large corpus of possible data sources, the values of a missing attribute are to be retrieved. State of the art methods return a single result that, to cover all queried entities, is fused from a potentially large set of data sources. We argue that queries on large corpora of heterogeneous sources using information retrieval and automatic schema matching methods can not easily return a single result that the user can trust, especially if the result is composed from a large number of sources that user has to verify manually. We therefore propose to process these queries in a Top-k fashion, in which the system produces multiple minimal consistent solutions from which the user can choose to resolve the uncertainty of the data sources and methods used. In this paper, we introduce and formalize the problem of consistent, multi-solution set covering, and present algorithms based on a greedy and a genetic optimization approach. We then apply these algorithms to Web table-based entity augmentation. The publication further includes a Web table corpus with 100M tables, and a Web table retrieval and matching system in which these algorithms are implemented. Our experiments show that the consistency and minimality of the augmentation results can be improved using our set covering approach, without loss of precision or coverage and while producing multiple alternative query results.

1. INTRODUCTION

In recent years, a large body of work studied Web tables, i.e., HTML tables published in Web pages that contain relational information, and their usage in data management. One application of special interest for data analysis are so called *entity augmentation queries*, in which, given a set of entities and an attribute not defined for them, the attribute's values for each entity is to be returned. With this query type, the missing attribute is usually specified just by its name, while the entity augmentation system decides on how to lookup Web tables, how to match them to the existing table

DOI: http://dx.doi.org/10.1145/2791347.2791353

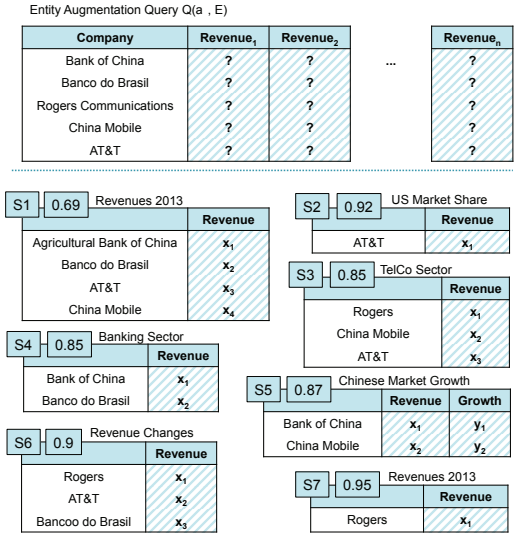


Figure 1: Example of a top-k augmentation scenario: query table and available data sources

and possibly how to merge multiple candidate Web tables into one result. In principle, entity augmentation queries could also be answered based on integrated knowledge bases or ontologies. However, using relations extracted from Web corpora offers a lot more long tail information and does not rely on the missing attribute being defined in some central repository. These properties make this type of querying both powerful as well as user-friendly, and thus interesting for exploratory analysis queries. While many methods for identifying matching Web tables and ranking them exist, two aspects have been studied less well so far: The first is constructing *minimal consistent results* with respect to the number of sources used. The second is computing several alternative results, or in other words, *Top-k Entity Augmentation*. We will discuss and motivate these two aspects using the example shown in Figure 1. The queried entities and the attribute to be augmented are shown in the top table, while the other tables represent available data sources. These sources are annotated with source number, a quality score and some table context.

Minimal Consistent Results: Existing methods compose the query result from many data sources on a per-entity basis. Though more complex algorithms exist (see Section 5), consider as an introductory example an algorithm that picks for

Final edited form was published in "SSDBM 2015: International Conference on Scientific and Statistical Database Management. La Jolla 2015", Art. Nr. 8,
ISBN 978-1-4503-3709-0
https://doi.org/10.1145/2791347.2791353

every queried entity the value from the best-ranked source, according to some measure of source-quality. Here, this naive algorithm would pick values from the sources *S7* for "Rogers", *S2* for "AT&T", *S5* for "Bank of China" and "China Mobile", and finally *S4* for "Banco do Brasil", using the highest ranked available source for each queried entity. This means that the algorithm picks a large number of data sources, almost one distinct source for each entity. More sophisticated methods for pruning candidate tables and picking values from the remaining candidates can be used, such as correlating or clustering sources, mapping sources to knowledge bases, or quality-weighted majority voting (again, see Section 5), but they will not fundamentally change this fact as long as values are picked on a per-entity basis. We argue that using a large number of sources is a problem for two reasons:

Firstly, it promotes issues of consistency, as the sources may be of high quality when considered on their own, but still do not form a consistent answer. For example, the sources may subtly vary in meaning, such as *S2*, which in contrast to the other sources shows US-revenue only. This difference has to be extracted from the table context in this case, which is not trivial for automatic methods. Even though better methods for creating consistent augmentations for some important dimensions such as time and unit of measurement have been proposed [16, 12], picking fewer candidate sources will help to improve consistency in general.

A second argument is concerned with the usability of an entity augmentation result: When coercing multiple data source's values into one result, properties important for data analysis such as transparency, lineage and trustworthiness of the result are lost. This is because in many domains the user can not blindly trust an automatic integration result, no matter how sophisticated the method used to create it. Rather he or she will use this result as a starting point, check the sources proposed by the system, and correct mistakes or switch some sources manually. If the system fuses data from many sources, this process is complicated.

In this paper, we therefore investigate methods that produce not only consistent augmentation results from several sources, but *minimal* augmentations, i.e., augmentations that use a minimal number of sources to facilitate the usage of the result. In the running example, one such result would be $S3, S4$, as it only uses two sources to augment all entities, even though the sources' average score is slightly worse than the score of the naive solution above.

Top-k Results: Entity augmentation queries are exploratory in nature, and the information need of the user, given by just a keyword, is relatively underspecified. Even though entity augmentation operates on structured data such as a big Web table corpus, the user still queries data unknown to him or her, and may thus not be able to form a precise query, or may even want to stumble on new aspects of his or her information need, similarly to working with a document search engine.

Consider the example query: the attribute was specified as a simple keyword *"revenue"*. However, the real world concept is more complex, with subtle variants such as *"sales revenue"*, different years of validity and different sources. A user may not even know on the spot which variant he or she is interested in. Information retrieval systems solve this problem of uncertainty in sources and unclear user intent by presenting not one exact, but multiple, ranked, alternative answers.

In the running example, instead of returning only one result based on $S3, S4$, one alternative would be a result based on $S1, S7$, which has a worse average score, but has clearly marked year in both sources, which may be more useful for the user on manual inspection. Another aspect are attribute variations, which, due to the exploratory nature of entity augmentation queries, may also be of interest to the user. An example would be a third result based on $S6$ and the second column of $S5$, representing changes in revenue instead of absolute revenue. Effectively, this means we aim to extend entity augmentation to a Top-k operation.

Note that we want to generate solutions that are real alternatives, such as the three examples above. We want to avoid slight variations of one solution that just switch a single data source for another, and also avoid creating multiple solutions from very similar datasets, as this would add little information to the top-k result. A meaningful top-k list would need to consider a *diverse* set of sources, exploring the space of available data sources while still creating correct and consistent answers. We will define our notion of result diversity in the following sections.

Finally, in addition to supporting exploratory querying, we also argue that this extension helps to reach sufficient levels of precision for data analysis tasks. Entity augmentation is based on information retrieval operations, such as lookup in a data source index, coupled with automatic schema- and instance matching algorithms. Even though these methods are ever-improving, they are not perfect and may still produce subtle mistakes, e.g., use $S2$ in a general revenue query as discussed above. We will show how the trustworthiness and usefulness of entity augmentations can be improved by extending it to a Top-k operation.

Note that the two aspects, construction of minimal consistent single results, and providing meaningful alternative solutions are complementary to each other. Based on these insights, we make the following contributions:

- We propose new entity augmentation algorithms that construct multiple minimal and consistent augmentations for a given entity set and attribute. For that purpose, we formalize the problem of top-k entity set augmentation by extending the classic weighted set cover problem to the *Top-k consistent set cover problem*.
- We propose a greedy algorithm that picks consistent data sources to construct individual augmentations, while maximizing the diversity between the results to provide meaningful alternative solutions.
- We improve on this first algorithm with a genetic set covering approach that naturally models the creation of a diverse set of individually strong solutions.
- We implement the algorithms in a new Web table retrieval and matching system called REA, and evaluate the system on a new corpus of about 100 million Web tables extracted from a public Web crawl, and measure the effects of our proposed algorithms on precision, coverage and runtime, but also the new dimensions consistency and diversity of the Top-k query results. We make the implementation and the Web table corpus available for other researchers.

In the following, we will formalize entity augmentation, discuss baseline implementation methods, our data model and

Final edited form was published in "SSDBM 2015: International Conference on Scientific and Statistical Database Management. La Jolla 2015", Art. Nr. 8,
ISBN 978-1-4503-3709-0
https://doi.org/10.1145/2791347.2791353

general strategies to construct augmentation results from several data sources in Section 2. We will then introduce our algorithms in abstract terms in Section 3.1 and then describe our system and the implementation details in Section 3.2. Evaluation, related work and conclusion appear in Sections 4, 5 and 6 respectively.

2. ENTITY AUGMENTATION QUERIES

This section will introduce entity augmentation queries, and the methods typically used to process them. We discuss issues of consistency in entity augmentation and the necessity of a Top-k extension, as well as the challenges that these imply.

An entity augmentation query is a query of the form

$$Q_{EA}(a_+, E_{a_1,...,a_n})$$

The result of such a query is the same set of entities with the augmented attribute added $E_{a_1,...,a_n,a_+}$.

An entity augmentation system manages a corpus of data sources, and is able to select a subset D relevant to a query Q_{EA}. Sources $d \in D$ can provide values for some subset of E, denoted $cov(d)$. The next section discusses the baseline approach to this problem.

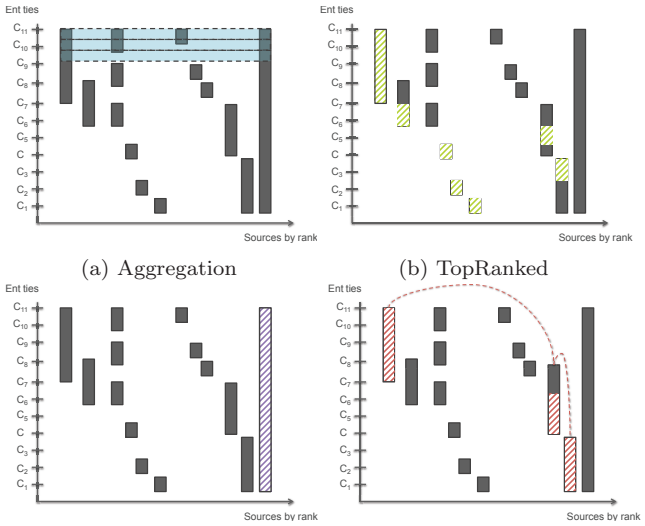
2.1 Baseline Method Analysis

Entity augmentation is typically implemented based on a table corpus extracted from the Web, usually in the order of several hundred million datasets [3, 14, 16]. The tables are indexed to enable quick retrieval based on attribute names, entities that the tables describe, and various forms of metadata, such as table or page titles. To process a query, the system retrieves tables with an attribute fuzzily matching a_+ and at least one entity from E. Then, various standard schema and instance matching methods such as string distances and synonym dictionaries are used to calculate a mapping between queried entities and entities in the Web table, as well as between the queried attribute and those of the candidate table. This process is repeated for further hits in the Web table index until matching tables for each entity are found.

Note, that if the entity augmentation system is implemented on top of a large body of unreliable data sources such as Web tables, the system will return a set of possible values for a given combination (a_+, e_n) instead of a single value as in a regular database query.

The main difficulty is deciding which value candidate to return as the augmented value for each entity. Most systems from literature assume that they can reconcile the multiple results into one result (see Section 5). As stated in Section 1, we argue that automatically merging the results of an inherently uncertain IR-based operation such as entity augmentation will not lead to a result the user can blindly trust and use in an analysis query. Furthermore, automatic methods of schema- and instance matching will never achieve perfect accuracy, and are always volatile with respect to thresholds, so it is never certain that a single augmentation will be valuable to the user. Instead of assuming that the perfect result can be created, one possibility would be to just return the whole set of possible values per entity, i.e., leaving out the error-prone coercion step.

$$Q_{EA}(a_+, \{e_1, ..., e_n\}) = \{V_1, ..., V_n\}$$



(a) Aggregation (b) TopRanked

(c) MinSources (d) Consistent

Figure 2: Strategies for creating augmentations

This, however, is of little use when the operation is used in context of an analytical query, as the user would have to choose the correct value on a per-entity basis, and the result would be unlikely to be very consistent, as every value might originate from a different Web table. In this paper, we use a more general definition of an EA query result, in which a set of k possible value sets is returned:

$$Q_{EA}(a_+, \{e_1, ..., e_n\}) = [\{v_{1_1}, ..., v_{n_1}\}, ..., \{v_{1_k}, ..., v_{n_k}\}]$$

Note that the result does not consist of k possible values for each instance, but k sets of values, each one containing exactly one new attribute value for each respective entity. We call one such set an *augmentation*, and the set of augmentations the query result. The user can then choose the most promising augmentation from the ranked list of alternatives for their information need, treating each one of them as one consistent source, just as in classic search engines. This, of course, leads to the question of how to construct useful Top-k entity augmentations from a set of partly matching data sources.

2.2 Constructing Augmentations

In this section, we discuss several naïve ways of constructing augmentations from a ranked list of data sources and analyze their weaknesses. Reconsider the running example (Figure 1), in which the user requires the addition of a new dimension *revenue* to an existing table containing various companies. We assume that a set of possible data sources D has been identified, matched and scored by an entity augmentation system. We will give details on how our system proceses these steps in Section 3.2. For now, consider the visualization of this intermediate result shown in Figure 2, in which the vertical axis shows the queried companies c_i, the gray boxes denote sources and what entities they provide a candidate value for, while the horizontal axis plots the sources' rank. The usual approach from related work is visualized in Figure 2a for the first three entities: each one is considered separately by aggregating all candidate values into one result, for example by performing some form of majority voting, or by clustering values and then picking a representative from the highest ranked cluster [14].

Final edited form was published in "SSDBM 2015: International Conference on Scientific and Statistical Database Management. La Jolla 2015", Art. Nr. 8,
ISBN 978-1-4503-3709-0
https://doi.org/10.1145/2791347.2791353

Let us now consider alternative strategies. One naïve strategy, called *TopRanked*, is shown in Figure 2b. Starting from the highest ranked data source, pick all the values it provides for entities that do not have a value yet. While this strategy does not pick values individually but according to their source's rank, it may still piece together data sources not fitting each other at all. Furthermore, a large number of data sources might be picked, making it harder for the user to understand the query result and assess its quality.
A naïve approach to solve this last problem would be to priorize data sources with large coverage of the queried entities. This strategy, called *Minimality*, is illustrated in Figure 2c. Although the selected data source has a low score, it was picked because it can single-handedly cover all input entities. Obviously, with this strategy the augmentation consists of a minimal number of data sources, but the quality according to the chosen ranking function might be low.
What is needed is a strategy that creates augmentations on the level of data sources, not entities, but without the disadvantages of *TopRanked* or *MinSources* . In Figure 2d, we give an intution for our approach: by introducing a similarity function between the data sources in addition to the scoring function, we will construct augmentations that are both minimal in the number of sources, maximal in the score of the datasets used, but also consistent with each other, as we will show in the next section. We will also show how such a function can be used to construct not only one augmentation, but several meaningful alternative solutions to present to the user. In the next section, we will present our Top-k consistent entity augmentation algorithms.

3. TOP-K CONSISTENT ENTITY AUGMENTATION

We will now first describe Top-k consistent *set covering* as an abstract solution to the problem of Top-k consistent entity augmentation in Section 3.1, and then apply it to the specifics of Web table-based augmentation in Section 3.2.

3.1 Ranked Consistent Set Cover

We propose a new approach for constructing entity set augmentations by modeling it as an extended form of the *Weighted Set Cover Problem*. Given a universe of elements U and a family of subsets of this universe S, each associated with a weight w_i, the Weighted Set Cover problem is to find a subset s of S whose union equals U, such that $\sum_{i \in s} w_i$ is minimized.
In our specific problem domain, the algorithm input consists of a set of entities E that are to be augmented, corresponding to U in the original problem, and a set of data sources $D = \{d_1, ...d_n\}$, as retrieved and matched by the underlying entity augmentation system, which corresponds to S. A single cover is then defined as an ordered subset of D that covers E, i.e., $c = [d_i, ..., d_x]$ with $\bigcup_{d \in c} cov(d) = E$. A score associated with each data source representing its quality with respect to the query is then used in place of the weights w.
So far, we could trivially map our problem to the well known Set Cover problem. Still, there are some crucial differences: In contrast to the original problem, where only a single minimal cover is required, for reasons given in Section 2.1, the output we aim for is a ranked list of covers, denoted $C = [c_1, ..., c_n]$. Furthermore, as illustrated in Section 2, the entity augmentation use case does not only require small covers with high scores, but *consistent* covers, i.e., covers consisting of datasets that fit each other well according to a similarity function $sim()$.

$$\forall c \in C \quad \text{maximize} \sum_{d_i, d_j \in c} sim(d_i, d_j) \qquad (Consistency)$$

We will discuss the definition of this function $sim()$ in Section 3.2. And lastly, the covers created should be *diverse*, i.e., they should not consist of the same or similar datasets throughout the Top-k list, but be complementary alternatives.

$$\text{minimize globally} \sum_{c_i, c_j \in C} sim(c_i, c_j) \qquad (Diversity)$$

We now discuss the reasoning behind these properties and incrementally build up to our proposed algorithms for Top-k consistent set covering.

3.1.1 Basic Framework

To reach these properties, we start from the well known greedy algorithm for the Weighted Set Cover problem, which, given a universe U, a set of sets S with weights w, and a set of yet uncovered elements F, in each iteration picks the set which minimizes:

$$\frac{w_i}{|S_i \cap F|}$$

Applying this to entity augmentation requires a scoring function for datasets yielding the weights w, which we call $score(d)$. This function assigns a score to each dataset reflecting its relevance to the query. We give a specific instance of this function for scoring Web tables in respect to an entity augmentation query in Section 3.2. Using this function, an initially empty cover C and a free entity set initially $F = E$, we can use the original greedy Set Cover algorithm to produce an ordered subset of D, by picking in each iteration the dataset d that maximizes:

$$score(d_i) \cdot |cov(d_i) \cap F| \qquad (1)$$

until $F = \emptyset$. Note that we maximize scores instead of minimizing weights as this is more intuitive for the problem domain.
An augmentation constructed in this way would roughly correspond to a middle ground strategy between the *TopRanked* and *MinSources* strategies discussed in Section 2.2, which means it may create single augmentations from very heterogeneous data sources.
Furthermore, if only the individual $score(d)$ per dataset is used, there is no intuitive way of creating useful Top-k augmentations. While it would be possible to create alternative augmentations using this algorithm by, for example, removing datasets already used in previous augmentations from consideration for further executions, there is no clear benefit, as it would likely just construct a worse version of the first augmentation.
• **Cover Consistency**: To counteract these effects, we model consistency between the datasets that make up a cover explicitly. We first require a similarity function between two datasets $sim(d_1, d_2)$ in $[0, 1]$, which operates both on the tables themselves as well as the extracted metadata. We give a specific instance of this function for Web tables in Section 3.2. Then, given an initially empty cover c and

Final edited form was published in "SSDBM 2015: International Conference on Scientific and Statistical Database Management. La Jolla 2015", Art. Nr. 8,
ISBN 978-1-4503-3709-0
https://doi.org/10.1145/2791347.2791353

a similarity aggregation function $\mathcal{F}$ such as average or max, we can greedily generate covers using consistent datasets by picking in each iteration the dataset d that maximizes:

$$score(d) \cdot |cov(d) \cap F| \cdot sim_{\mathcal{F}}(d, c) \tag{2}$$

This means we encourage picks of data sources that are similar to data sources that were already picked for the current cover. We assume as a special case that $sim_{\mathcal{F}}(d_i, \emptyset) = 1$, which implies that the first data source picked will be the same as in regular set covering. Subsequent picks on the other hand will be influenced by already picked sources. This also implies that datasets with a low inherent score, that are not be picked early because of this, may still be picked in a later iteration, if they fit very well with the datasets picked at first. Since we still require $|cov(d) \cap F|$ to be greater than zero, the algorithm will still make progress with every pick, as only datasets that provide at least one new value can be picked.

• **Result Diversity**: Using objective function 2, the algorithm picks datasets to create covers that not only score well with regard to the query, but also fit well together according to $sim(d_1, d_2)$. Still, the question how to create multiple and complementary augmentations is unanswered.

Let C denote the set of previously created covers. Our core idea is to do consecutive runs of the greedy algorithm using the same input datasets, with each run placing greater emphasize on datasets that are dissimilar to datasets picked in previous iterations, i.e., dissimilar to datasets in $\bigcup_C$. Implementing this idea naïvely however, for example by dividing function 2 by $\sum_{d_i \in \bigcup_C} sim(d, d_i)$ does not yield the expected results. While the second iteration might then choose datasets from a different part of the similarity space than the first iteration, the score becomes more and more meaningless with more iterations as $\bigcup_C$ grows and the newly picked datasets are compared to a larger and larger subset of the candidate set D, leading to more and more uniform values for $\sum_{d_i \in \bigcup_C} sim(d, d_i)$.

Instead, we introduce a more complex dissimilarity metric based on individual entities in E and the datasets that were used to cover them in previous iterations. We define a function $coveredBy(e, C)$ which yield the datasets that were used to augment entities in covers created in previous iterations. We can then define out final scoring function as

$$\frac{score(d) \cdot |cov(d) \cap F| \cdot sim_{Agg}(d, c)}{redundancy(d, F, C)} \tag{3}$$

where

$$redundancy(d, F, C) = \sum_{e \in F \cap cov(d)} sim_{\mathcal{F}}(d, coveredBy(e, C)) \tag{4}$$

This means we penalize picks that would cover entities with data sources that are similar to datasets that were already used to cover these entities in previous iterations, and even more so if they were used multiple times. This scheme produces the same cover as the previous scheme in its first run, as C is empty initially. However, in further iterations it picks the highest scoring datasets that are dissimilar to those picked in the first iteration. By penalizing similarity to previous covers, we avoid using the same top-ranking datasets again and again for all covers in the Top-k list we aim for.

Algorithm 1 Top-k consistent set covering: Greedy

function GREEDY-TOPK-COVERS(k, E, D)
  $C \leftarrow \emptyset$
  $U \leftarrow \begin{pmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{pmatrix}_{|E| \times |D|}$ ▷ Usage matrix
  while $|C| < k$ **do**
    $c \leftarrow$ COVER(E, D, U)
    for all $(e \rightarrow d) \in c$ **do** ▷ Update Usage Matrix
      $U[e, d] \leftarrow U[e, d] + 1$
    if $c \notin C$ **then** ▷ Remove duplicates
      $C \leftarrow c$
  return C

function COVER(E, D, U)
  $c \leftarrow \emptyset$
  $F \leftarrow E$ ▷ Free set, uncovered entities
  while `True` **do**
    if $|F| = 0$ **then**
      return c
    $d \leftarrow \arg\max_{d \in D} \frac{score(d) \cdot |cov(d) \cap F| \cdot sim_{Agg}(d,c)}{\text{REDUNDANCY}(d, D, F, U)}$
    for all $e \in F \cap cov(d)$ **do**
      $F \leftarrow F \setminus e$ ▷ Update free set
      $c \leftarrow c \cup (e \rightarrow d)$ ▷ Update cover
  return c

function REDUNDANCY(d, D, F, U)
  $r, norm = 0, 0$
  for all $e \in F \cap cov(d)$ **do** ▷ Coverable by d
    $u \leftarrow U[e]$ ▷ Sources used to cover e
    $r \leftarrow r + \sum_{i=0}^{|u|} u[i] * sim(d, D[i])$
    $norm \leftarrow norm + u[i]$
  return $\frac{r}{norm}$

With this scoring function, we can construct a greedy consistent set covering Algorithm 1 that produces both consistent individual augmentations, as well as diversified solutions when run with $k > 1$. In Algorithm 1, the function *Greedy-TopK-Covers* produces k covers by calling the function *Cover* k times, while keeping a $|E| \times |D|$ matrix called U as state between the calls. While the *Cover* function performs the basic greedy set cover algorithm with the new scoring function defined above, the main function updates the matrix U after each iteration by increasing the entry for each entity/dataset combination that is part of the produced cover. Note that the main function also discards duplicate solutions, which may occur if the influence of the *redundancy* function is not strong enough to steer the search away from an already existing solution. Still, the matrix U is updated even if a solution is rediscovered, so that further choices of the same data sources become more and more penalized, guiding the search into a different part of the solution space.

The greedy approach described above, while being easy to implement and fast to execute, will not necessarily construct the best possible list of solutions, as our evaluation in Section 4 will show. Therefore, we introduce two further algorithms as extensions of the basic framework in the next two sections.

Final edited form was published in "SSDBM 2015: International Conference on Scientific and Statistical Database Management. La Jolla 2015", Art. Nr. 8,
ISBN 978-1-4503-3709-0
https://doi.org/10.1145/2791347.2791353

Algorithm 2 Top-k consistent set covering: Greedy*

function GREEDY*-TOPK-COVERS(k, s, E, D)
  $C \leftarrow$ GR-TOPKCOVERS($k * s, E, D$) ▷ As in Alg. 1
  $C \leftarrow$ SELECT(k, C)
  return C

3.1.2 Extension: Greedy*

The first extension of the basic framework is based on the observation that the first k solutions produced by Algorithm 1 may not necessarily be the best solutions. After the first solution has been produced the search is mainly guided by using different datasets for each solution, and thus new combinations of used data sets are often not considered in the basic greedy algorithm. One simple extension is called *Greedy**-algorithm, which uses the basic greedy algorithm to create more covers than requested, and then introduces a second phase to the query processing called *Select*, in which the k best solutions are selected from a pool of $s \times k$ possible solutions, as shown in Algorithm 2, with s being the scale factor.

This raises the question of how to select the Top-k solutions. Following [7] we evaluated two ways to pick diversified Top-k solutions from a pool of possible solution: one again based on a *greedy* approach and one based on *replacement*. The greedy approach picks the best solution by score and consistency, then picks further solutions that maximize these criteria while using dissimilar datasets. This works in the same manner as the greedy approach from Section 1, but with the scoring functions defined on the level of covers instead of the level of data sources. We skip the detailed definitions for space reasons, as they follow from what is shown in Section 3.1.1, and the full implementation is available (see Section 4.1).

We also evaluated solutions based on replacement, in which the first k covers as produced by the greedy approach are used as the initial Top-k list, which is then iteratively refined. Refinement is done by removing the cover least fit according to the scoring functions and replacing it with a new cover from the pool of possible answers until a stop criterion is reached, e.g., when the diversity of the Top-k list has fallen under a threshold.

In comparison to the *Greedy* algorithm, the *Greedy** approach should find better solutions as it searches a larger portion of the search space, at the cost of a runtime that increases with the scale factor s, plus some overhead for the selection phase.

3.1.3 Extension: Genetic Approach

There is a large amount of literature dedicated to the set covering problem, in which various optimization methods apart from the greedy approach are studied (see Section 5). The genetic approach seems especially viable for our specific versions of the problem, as it intrinsically generates a pool of solutions from which k can be picked, and both consistency and diversity of the results can be modelled intuitively, the first as part of the fitness function and the second as part of the population replacement strategy. To apply the genetic framework to a problem, one needs to define the representation of individuals and their genomes, a fitness function, a cross over and a mutation function, as well as a strategy for creating an initial population and for choosing whic individuals to cross and which to replace in each generation.

Algorithm 3 Top-k consistent set covering: Genetic

function GENETIC-TOPK-COVERS(k, s, E, D)
  $U \leftarrow \begin{pmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{pmatrix}_{|E| \times |D|}$ ▷ Usage matrix
  $Pop \leftarrow$ INIT-GREEDYTOPKCOVERS($k * s, E, D, U$)
  for $i \in 0..k * s$ **do**
    $c_1, c_2 \leftarrow pickRandom(Pop)$ ▷ Select Parents
      ▷ Tournament vs. most similar solution
    $w_1, l_1 \leftarrow tournament(c_1, \arg\max_{x \in Pop \setminus c_2} sim(c_1, x))$
    $w_2, l_2 \leftarrow tournament(c_2, \arg\max_{x \in Pop \setminus c_1} sim(c_2, x))$
    $c_+ \leftarrow$ COVER($E, w_1 \cup w_2, U$) ▷ As in Alg. 1
    MUTATE(c_+)
    FILLANDPRUNE(c_+)
    $removed \leftarrow min_{score}(l_1, l_2)$
    $Pop \leftarrow (pop \setminus removed) \cup c_+$
    $U \leftarrow$ UPDATEUSAGE($U, c_+, removed$) ▷ As Alg. 1
  $C \leftarrow$ SELECT(k, Pop) ▷ As in Alg. 2
  return C

While our approach is inspired partly by [1], our problem domain makes different choices for almost all of these decision necessary. Obviously, we can represent an individual's genome as the set of data sources it is comprised of. We can then use the final scoring function 3 as the fitness function for individuals. The population is initialized by creating covers with the Greedy algorithm until all candidate sources have been used in at least one cover. The Greedy algorithm is modified in this use case to strongly favor unused data sources in later iterations to quickly produce such an initial population.

The most interesting step is the crossover function, as combining two sets of data sources does not necessarily yield a set that is again a set cover. If the wrong genes are passed to the descendant solution, the solution may not be feasible, i.e., there may be uncovered entities, or if too many genes are passed, the cover may not be minimal. In [1], genes shared by both parents are passed on definitively, and those only present in one parent are passed with a probability corresponding to their weight, which leads to potentially infeasible or non-minimal solutions as mentioned above. To correct this, another step of pruning redundant genes and adding random genes from the pool that cover missing entities was proposed.

However, this solution does not take consistency of the generated descendant into account. Instead, we again use the basic Greedy approach as a building block, and take the union of two parent genes as the set of candidates, instead of the whole set D. We use the same mutation strategy, i.e. randomly flipping bits in the gene, and use the prune and fill approach from [1] to remove datasets that may have become redundant or add random datasets to fill holes created by mutation.

Diversity is achieved through several mechanisms. Firstly, it is achieved implicitly by using the Greedy cover algorithm as a building block, as it uses a global usage matrix U throughout the run of the genetic algorithm. Secondly, the mutation step introduces randomness into the population, which increases diversity. Finally, the parent selection and

Final edited form was published in "SSDBM 2015: International Conference on Scientific and Statistical Database Management. La Jolla 2015", Art. Nr. 8,
ISBN 978-1-4503-3709-0
https://doi.org/10.1145/2791347.2791353

replacement strategy we employ encourages diversity. Specifically, we select two parents randomly to create a descendant, but before crossover, we pair them with the most similar existing solution in the population, to form two simple binary tournaments. The respective winners w_1 and w_2 are determined by the *score()*-function, and are used for creating a new cover, while the losers l_1 and l_2 are candidates for replacement in the next generation. Algorithm 3 gives an overview of our genetic consistent Top-k set covering approach and all the mentioned mechanics. In this algorithm description, *Select* is the selection phase that picks k covers as discussed in Section 3.1.2.
Having introduced our abstract algorithms for Top-k diversified and consistent set covering in this section, the next step is to instantiate them for the problem of Web table-based entity augmentation by giving the Web table-specific scoring and similarity functions $score(d)$ and $sim(d_1, d_2)$ used in our entity augmentation system REA.

3.2 REA System Overview

For space reasons, and since we do not propose novel methods in this respect, we only give a very short overview our REA system in this section and refer the reader to the same section in the longer version of this paper[1] for details.
REA uses both relational Web tables, also called "entity-value" tables, as well as entity-tables, that focus on one entity only, also called "attribute-value" tables. This allows consistent results for domains where relational tables are available, but increases recall in domains where no or only few such tables are ever published.
Concerning the $score(d)$ function, we employ an average of the following factors: quality of the schema-/instance match between the web table and the query table using standard matching techniques, the overlap of query terms and the table's metadata such as top terms extracted from page context, title and URL, and finally the overall trustworthiness of the source, approximated from an external Web domain popularity database.
Concering the $sim(d_1, d_2)$ function, we use similar measures as those proposed in [14, 16]: we consider the similarity between the attribute names, the similarity between the values provided and the tables' page contexts. We also employ similar extractors for units of measurement, orders of magnitude and time, whose results are then compared.

4. EVALUATION

We conducted an experimental evaluation of the proposed techniques to study the following questions:

- Do the proposed algorithms reduce the weaknesses of the baseline algorithms regarding consistency and minimality?
- Are the Top-k results produced diverse while maintaining individually correct and consistent augmentations?
- How do the basic measures such as coverage and precision change over the Top-k results?
- How do the proposed algorithms compare with respect to runtime performance?

[1]http://wwwdb.inf.tu-dresden.de/misc/publications/rea-long.pdf

4.1 Repeatability

To enable repeatability we publish the implementation[2], but also include the web table corpus used for the evaluation[3]. This corpus contains 100M Web tables extracted from a publicly available Web crawl[4]. The Scala-based implementation includes state-of-the-art Web table indexing and retrieval, as well as schema- and instance matching systems suitable for working with Web tables, as well as implementations of the Set Covering-based Entity Augmentation algorithms described in this paper. The corpus includes not only the raw table- and meta data, but also the code used for extraction and a Java library to quickly get started with using it for other research. We also include the queries and queried entities, as well as the results and the human judgments (see Section 4.2) used in the form of an SQLite database with the implementation code.

4.2 Experimental Setup

This section introduces our experimental setup, specifically the dataset and queries we use, the baseline algorithms and most importantly, the indicators we measure and how they are measured.

Dataset: All queries are run against the full corpus, described above in Section 4.1.

Queries: To enable comparable results, we used similar domains and queried attributes as in related work[14, 16]: companies from the Forbes 2000 list[5] with the attributes (revenue, employees, founded), countries with the attributes (population, population growth, area) and large cities (top 3000 worldwide) based on the Mondial database[6] with the attribute (population). When not mentioned otherwise, the default query set used for the experiments has 4 queries for each domain-attribute pair, with 20 entities ($|E| = 20$) each, and $k = 10$. For the company and city domain we also differentiate between top and random entities, i.e. four queries from the top 100 of the Forbes list, and four with randomly picked entities. In addition to the standard queries, we also vary $|E|$ and k for the performance experiments.

Gold Standards: To establish whether an answer given by any algorithm is correct, compiling a Gold Standard, i.e. the set of correct answers for the test queries, is one possible approach. However, we do not think this method is adequate as there is usually no single truth even for basic facts. Even seemingly simple facts such as population counts can refer to different years, or may use different ways of measurement, e.g., different ways of defining city borders for the city domain. Furthermore, there may be slight variations of the queried attribute, such as "under 25 years" for the attribute "population", which may still be of interest for an exploratory query. As discussed in Section 1, this is one of the reasons why we extended entity augmentation to a Top-k paradigm in the first place.
We therefore asked human judges to classify answers as relevant to the query or not. Specifically, for each entity in

[2]http://github.com/JulianEberius/REA
[3]http://wwwdb.inf.tu-dresden.de/misc/dwtc
[4]http://commoncrawl.org
[5]http://www.forbes.com/global2000/
[6]http://www.dbis.informatik.uni-goettingen.de/Mondial/

Final edited form was published in "SSDBM 2015: International Conference on Scientific and Statistical Database Management. La Jolla 2015", Art. Nr. 8,
ISBN 978-1-4503-3709-0
https://doi.org/10.1145/2791347.2791353

each result set we asked the judges to evaluate that both the data source used to cover the entity was relevant to the query, and also the entity was matched correctly. Note that while we collected judgments to cover all the domains and attributes introduced above, to keep the amount of manual work manageable, the number of augmentations created was smaller than in the automatic evaluation (only up to $k = 5$).

Indicators: Due to the nature of the cover problem, it is not enough to measure precision and coverage of the query results to asses the quality of the proposed algorithms. We further need to determine measures such as consistency and minimality of the individual augmentations, as well as the diversity of the result list. We will now discuss the indicators measured in our evaluation in detail.

• *Coverage:* We measure the percentage of the queried entities that are augmented with a value. Note that we do not aim at measuring the quality of the Web tables corpus or the schema- and instance matching system used, but want to asses the quality of the covering algorithms and the influence of different coverage thresholds (explained below) on the query result. For this reason, if not stated otherwise, we only measure the indicators for entities for which our combination of table corpus and matching system can return at least one (potentially incorrect) result value. We do, however, also provide absolute coverage percentages for reference and comparison with other entity augmentation systems in Section 4.4.

• *Precision:* We measure the percentage of entities for which a relevant value was retrieved, as described above in the discussion of Gold Standards. Note that this considers augmented values for each entity individually, i.e., an augmentation with precision 1.0 means each entity was augmented with a value that was judged relevant with respect to the query keyword, but the augmentation is not necessary consistent.

• *Consistency:* We measure consistency as the average similarity between the datasets that were used to create the respective cover. A higher value means that the respective augmentation was created from more similar datasets.

• *Minimality:* Measures the number of datasets used in a cover in relation to the number of entities it covers and invert, i.e., $1.0 - \frac{|c|}{\sum_{d \in c} |cov(d)|}$ or 1.0 if $|c| = 1$. The measure is purposefully not related to the set of all entities E, as a low coverage would be rewarded with a high Minimality. Thus, a high value means that the augmentation was created from a small number of data sources, while 0.0 means that each entity in the set was covered using a different data source. As we established in Section 1, a smaller number of sources (and thus a higher *Minimality* score) is better, as the user will have to check less sources to be able trust the result.

• *Score:* For this indicator, we measure the average ranking score each individual dataset received from the scoring function defined in Section 3.

• *Diversity:* This indicator is measured not for a single cover, but a set of covers. It is calculated as the average distance between all covers in the set. Distance in this context is defined using the inverse of the similarity function defined in Section 3.1.1. Since the similarity function is defined on pairs of datasets, a similarity aggregation function is needed. In our experiments, we use the average distance between the datasets of two covers as the distance between the covers.

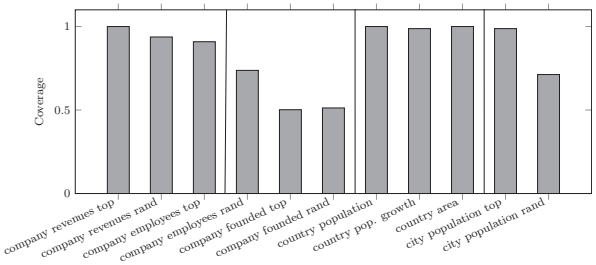


Figure 4: Coverable entities by query

Parameters: To study the trade-offs between coverage and consistency, we introduced early break parameters *thCov* and *thCons* to our algorithms. Specifically, we allow our algorithms to stop covering if at least *thCov* percent of the entities are covered, and the consistency would drop below *thCons* by continuing. If not mentioned differently, *thCov* is set to 1.0, i.e. we aim for full coverage of the queried entities regardless of consistency.
The search space factor s, used in the Greedy* and Genetic approaches (Algorithms 2 and 3) that determines the number of solutions to create in Greedy* and the number of generations in the Genetic approach is kept constant at 10 in the paper for space reasons.

4.3 Baseline Algorithms

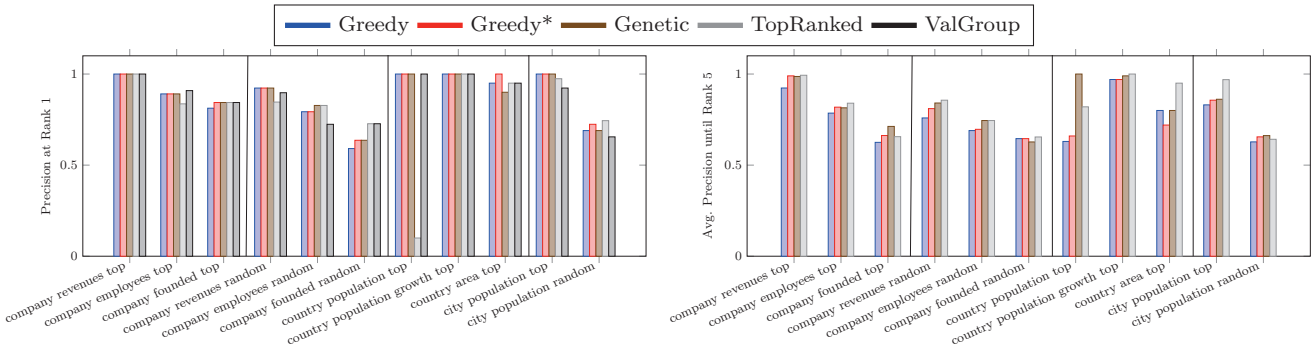
We include two baseline algorithms into our evaluation: *ValueGrouping* and *TopGrouping*. Both use exactly the same Web table retrieval and matching components as our proposed algorithms, so the set of candidate datasets and their scores are the same as well. We will now describe both in detail:

• **ValueGrouping:** This baseline is modeled after the value prediction strategy used in [14]. It collects all values found in all candidates into per-entity groups, and clusters these groups according to their values. It then uses the highest scoring value from the cluster with the highest sum of scores, thus favoring high ranking data sources that are supported by many other sources with similar values. It corresponds to the *aggregation* paradigm shown in Figure 2a, and produces only one augmentation.

• **TopRanked:** Here, the highest ranked data source that can produce a value for each entity is chosen to provide the value for the first augmentation. The next augmentation is then constructed from each second best ranked value for each entity respectively, until K results have been constructed. This approach does produce K different covers of the input entity set, but does not take consistency or diversity into consideration. It corresponds to the *TopRanked* paradigm shown in Figure 2b.

Having introduced data, queries, measures and baseline for our experiment, we will now discuss the results of our experiments, regarding coverage, precision, the cover quality indicators described above, and also runtime performance.

Final edited form was published in "SSDBM 2015: International Conference on Scientific and Statistical Database Management. La Jolla 2015", Art. Nr. 8,
ISBN 978-1-4503-3709-0
https://doi.org/10.1145/2791347.2791353



(a) Precision at rank 1 (b) Average precision up until rank 5

Figure 3: Results of manual precision evaluation for thCons=0.0, thCov=1.0

4.4 Evaluating Entity Coverage

As described above, coverage, i.e., the percentage of entities for which a value could be retrieved from the Web table corpus, is independent of the Top-k consistent set cover algorithms that we study. Nevertheless, for reference and comparison with related work we give coverage stats for our web table corpus and our matching system in Figure 4.
Generally, our results regarding coverage are in line with the results from [16]. In the city and country domain, coverage tends towards 1.0. For countries, the universe is sufficiently small so that most Web tables concerned with countries contain all, or a significant portion of them, making this domain the easiest, also for the following experiments. For cities, the task could of course be made arbitrarily hard by allowing very small cities, but as in [16] we only consider sufficiently populous cities, so that the coverage is still high. The cities domain is still harder than the countries domain, as the total number of possible entities is not as limited. Finally, the company domain is the hardest of our test set. This is mainly due to the number of companies that exist and the many possible name clashes with very similar company names containing generic terms such as "East Japan Railway" or "National Grid". Again, our numbers are in line with the related work: the top companies of the Forbes list can be easily found in our corpus, while the coverage drops towards 50% for tail companies.

4.5 Evaluating Single Entity Precision

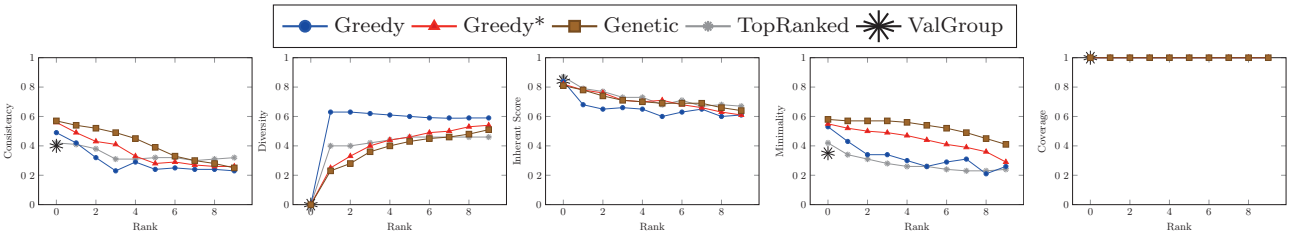
In the case of our consistent set covering algorithms, we are not only interested in the correctness of the augmented values for each individual entity, but in the consistency of these values across a single cover. Still, we will first evaluate the precision of the individual augmented values in this section, with no consideration of their composition to covers, and then study cover quality parameters as defined above in Section 4.2.
First, consider Figure 3, which depicts the average precision of each algorithm by queried domain/attribute combination. The left graph (3a) shows the average precision of the respective first augmentation produced by each algorithm, while the right graph (3b) shows the average precision over the top five results. As the baseline algorithm *ValueGrouping* can not produce more than one augmentation, it is not shown in the right plot. For the *company* and *city* domain, we differentiate between top and tail queries as described in Section 4.2.

	Best Result not on Rank 1	Average Improvement over Rank 1
Genetic	14%	12%
Greedy*	14%	10%
Greedy	23%	9%
TopRanked	32%	7%

Table 1: Percentage of queries with best solutions not on rank 1, and improvements over rank 1

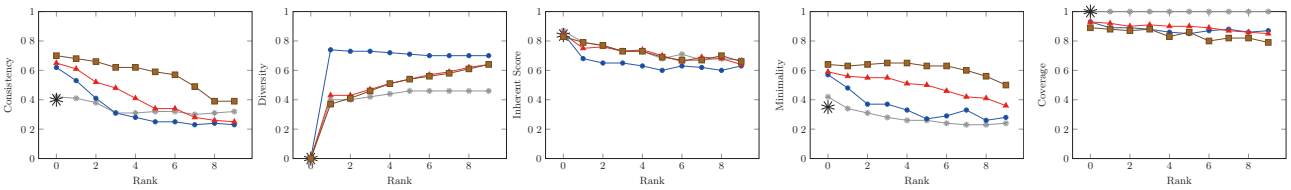
Note that the differences between the algorithms in this experiment are minor. This is because individual entity augmentation correctness is mainly dependent on the corpus and matching system used, and only less so on the way the data sources are composed. At rank 1 (Figure 3a), when removing the outliers, the average precision of all algorithms is almost equal at about 88% and standard deviations between 10% (TopRanked) and 14% (Greedy*). When looking at single entity precision averaged until rank 5 using similar outlier correction, the greedy set cover-based methods drop to between 75% and 77%, while the Genetic and TopRanked approaches reach 82% over all domains. The differences in general trend between the domains are similar to those in coverage (Section 4.4).
However, concerning our premise from Section 1 that the best solution may not always be the first solution generated by an entity augmentation method, we measured the percentage of queries for which the best solution was not on rank 1, and how much the best solution improved over the first solution. The results are given in Table 1. We can see that only in a minority of queries the best solution is found on later ranks, which is to be expected if the ranking function has been constructed carefully. Still, for all methods, a considerable amount of queries, from 14% for Genetic and Greedy* approaches to 32% for the baseline, is better answered with one of the later augmentations. This supports our argument that entity augmentation should use the Top-k result paradigm. Note that this just considers single entity precision. Our findings concerning consistency and diversity presented in Section 4.6 will give further arguments for the Top-k approach. Even more importantly, while the averaged single entity precision (Figure 3) may be very similar for all methods, and thus TopRanked may seem like a viable strategy because of its simplicity, we will see how the non set-cover based strategies achieve their individual entity result quality by not paying attention to the quality of the result as a whole.

Final edited form was published in "SSDBM 2015: International Conference on Scientific and Statistical Database Management. La Jolla 2015", Art. Nr. 8,
ISBN 978-1-4503-3709-0
https://doi.org/10.1145/2791347.2791353



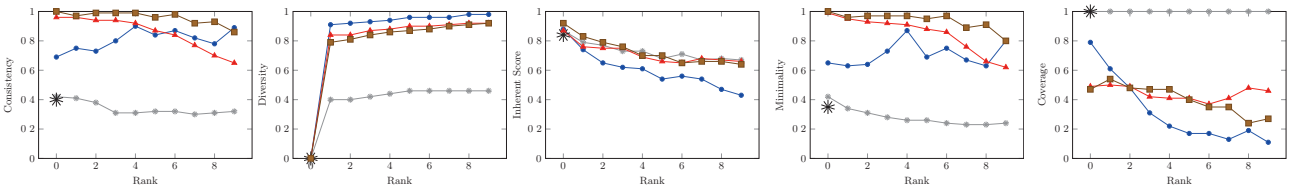
(a) Consistency (b) Diversity (c) Inherent Score (d) Minimality (e) Coverage

Figure 5: Cover quality indicators by rank, Full covers scenario: thCov=1.0 thCons=0.0



(a) Consistency (b) Diversity (c) Inherent Score (d) Minimality (e) Coverage

Figure 6: Cover quality indicators by rank, Relaxed Scenario: thCov=0.75 thCons=0.4



(a) Consistency (b) Diversity (c) Inherent Score (d) Minimality (e) Coverage

Figure 7: Cover quality indicators by rank, Full consistency scenario thCov=0.0 thCons=0.4

4.6 Evaluating Cover Quality

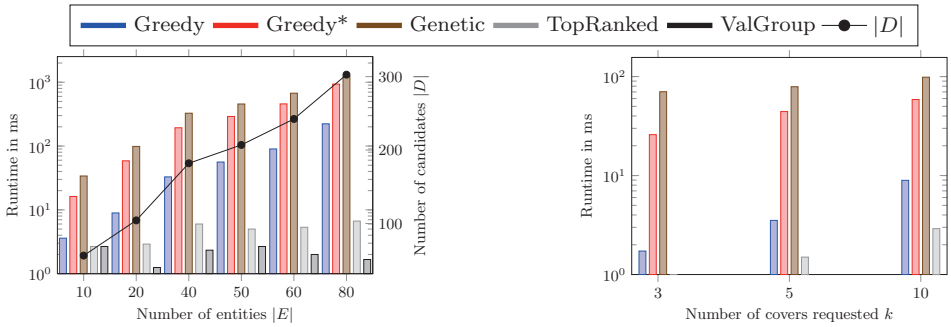
In this section, we will demonstrate where our novel entity augmentation methods differentiate themselves from the baseline approaches: consistency and diversity, as well as minimality of the number of sources used. First, consider Figures 5 to 7, which depict the five indicators consistency, diversity, score, minimality and coverage by rank, from 0, the first augmentation, to 9, the last one in the $k = 10$ case. Since the *ValueGrouping* baseline algorithm can only produce a single result using majority voting between all possible values for an entity, it is only plotted as a larger dot instead of a line.

In Figure 5, the special threshold *thCov* (see Section 4.2) is set to 1.0, demanding full covers and representing the default case. We can see that the baseline algorithms, while producing results with very high average scores, do so at the expense of using more different sources per cover, which is visible in the Minimality and the Consistency indicators. Here, the benefit of viewing the augmentation problem as a form of the set cover problem becomes apparent: the set cover-based algorithms can create solutions that are much smaller, without loosing noticeably in score. It is also quite noticeable that the basic Greedy approach, while constructing smaller solutions, is not able to outperform the baseline in terms of consistency or score. The Greedy* algorithm, by exploring more possible solutions is able to improve on the baseline, while the Genetic approach finally beats the baselines clearly with respect to minimality and consistency: It produces results that are 18% more consistent and achieve a 23% better minimality on average, while maintaining diversity of the result set and and average score of the data sources used. Note that these are average scores over all ranks, and the gains on the first ranks are even higher. Comparing the first cover created to the *ValueGrouping* baseline yields a 23% improvement in both consistency and minimality.

Now consider the diversity plots, which show at each rank the diversity for all results from the first one until the respective rank. This explains shape of the curves: they start at zero for one solution, then jump sharply at rank two, when distances between the two solutions can be measured. The plot shows that the algorithms behave differently with respect to using similar datasets for further solutions after the initial ones have been generated. The basic approach moves through the similarity space of the candidate sources as it fills its usage matrix U (see Algorithm 1), penalizing used candidates and those similar to it. Since it only creates at most k solutions, it is likely to pick new, unpenalized candidates in each iteration, which explains the strong diversity of the augmentations generated. The Greedy* approach on the other hand creates a multiple of the necessary solutions and will often "wrap around" the similarity space of candidates, reconsidering used sources in new combinations. The same is even more true for the Genetic approach, that will combine existing good solutions to new ones, promoting the survival of useful data sources.

While the benefit of reusing good partial solutions is obvious, this leads to more uniform results across the first results of the Top-k list, as shown in Figure 5b. Note however that all algorithms converge to a similar diversity level as more solutions are added to the Top-k list, as it becomes harder to keep a set diverse the larger it is. At $k = 10$, the Genetic

Final edited form was published in "SSDBM 2015: International Conference on Scientific and Statistical Database Management. La Jolla 2015", Art. Nr. 8,
ISBN 978-1-4503-3709-0
https://doi.org/10.1145/2791347.2791353



(a) Runtime by $|E|$, fixed $k = 10$

(b) Runtime performance by k, fixed $|E| = 20$

approach that dominates the baselines in all other aspects reaches similar levels of diversity when looking at the whole result list, even though it produces more similar solutions on the first few ranks.

Next, consider Figures 6 and 7. In these figures we demonstrate the useful trade-off between coverage and consistency that is possible with our algorithms: if we lower *thCov*, i.e., we allow a certain amount of uncovered entities if necessary to keep consistency over *thCons*. The baseline algorithms do not have early braking included and stay constant as a frame of reference. In Figure 6, we allow 25% uncovered entities, if necessary to keep consistency above 0.4, and in Figures 7 we allow the algorithms to leave as much uncovered as necessary to keep consistency above 0.4. Note that *thCons* and *thCov* are guidelines for the covering process, not hard limits. Allowing 25% uncovered entities can dramatically improve the results in all areas, especially consistency and minimality, which in this case improve 26% and 31% respectively, using the genetic approach. Interestingly, the results are even more diverse in this case, improving 12% over the default case for the genetic algorithm. These two observations can be explained by the fact that certain entities can only be covered by data sources that have no fitting "partner" sources, which reduces consistency if they have to be covered in one cover, while other entities are coverable by only a small number of sources, which then have to be included in almost every cover, reducing diversity.

In the third covering experiment shown in 7, we allow aggressive optimization of the result regarding consistency by sacrificing coverage completely. We can see that perfectly consistent and diverse results can mostly only be reached by not using more than one source: both Greedy* and Genetic return mostly single datasets as single augmentation on the first few ranks. Note how in this configuration the algorithms all select almost completely different datasets for every augmentation as the diversity stays above 0.8 throughout the higher ranks, but to do so have to choose smaller and smaller datasets, which is visible in the coverage quickly falling to unusable levels.

4.7 Evaluating Runtime Performance

We evaluate the runtime performance of the proposed methods on a Ubuntu 12.04 Virtual Machine running on 2,4GHz Intel Xeon CPUs. As mentioned in Section 4.1, the algorithms are implemented in Scala, using version 2.11, and are executed on a version 1.8.0 Oracle JVM. We do not measure the runtime of the Web table retrieval and the schema and instance matching step, as this is a constant overhead for all methods, but do include the similarity calculations, as they are part of our methods and not of the baseline algorithms. The set of queries is the same as in the main experiment reported on in Section 4.6. We give the runtime once as a function of $|E|$, the number of entities queried in Figure 8a, and once as a function of k, the number of covers to be created in Figure 8b. Note that general set cover algorithms' performance depends both on the size of the universe, or in our case $|E|$, and on the number of candidate subsets, or of candidate data sources $|D|$ in our case. In our system however, the number of candidates $|D|$ directly correlates with $|E|$, as our retrieval and matching system returns more candidate Web tables when more entities are queried, as shown on the secondary axis in Figure 8a. We thus measure the influence of both dimensions of the problem size in one experiment.

From the results it is clear that the baseline techniques are superior in their runtime performance due to their simplicity. Both basically consist of grouping possible values by entity, and ordering by score in the case of *TopRanked*, or by fuzzy majority vote on the values in the case of *ValueGrouping*. The set covering approaches are more complex, moving them into a different order of magnitude concerning runtime. Still, even our prototypical implementation was able to answer all our test queries in less than one second.

5. RELATED WORK

Notable first work on using the wealth of tables on the Web was done in [4]. The authors extracted a large scale corpus of Web tables and proposed several applications for such a corpus. In [3], this work was continued and an algorithm called *MultiJoin* was proposed that attempts to find matching Web tables for each queried entity independently, and then clusters the tables found to return the cluster with the largest coverage. However, it does not try to construct consistent solutions, but returns the set of possible values for each entity.

The most strongly related work is the InfoGather system[14], and its extension InfoGather+[16]. Infogather improved the state of the art especially by identifying more candidate tables than the direct matching approach by introducing Web table similarity measures and identifying tables indirectly matching the query through them. They also introduce methods for efficiently computing the similarity graph between all indexed tables offline, which could be applied to our approach as well, as we currently perform similarity calculations online. This is due to the main difference between the two approaches: InfoGather uses the similarity graph to find more candidate tables, but the actual choice of augmented value is done per entity and based on fuzzy grouping

Final edited form was published in "SSDBM 2015: International Conference on Scientific and Statistical Database Management. La Jolla 2015", Art. Nr. 8,
ISBN 978-1-4503-3709-0
https://doi.org/10.1145/2791347.2791353

of the values. On contrast, we use the similarity measures as evidence for selecting a minimal number of Web tables for our answer, and also produce alternative solutions, which would not be straightforward based on the fuzzy grouping value prediction.

InfoGather+ improves the system by tackling similar consistency issues as our work: it assigns labels for time and units of measurements to tables, and propagates these labels along the similarity graph described above to other tables where such labels can not be found directly. While InfoGather+ tackles the problem of producing more consitent results from various possible Web sources, it does not produce Top-k results, or minimize the number of sources used.

Web Tables have not only been used to augment attributes to a set of entities. There is a large body of work that uses Web table corpora for specific query types. For example, in [10] whole tables are materialized automatically just from keyword queries without any known entities, while [15] uses Web tables to answer single fact keyword queries.

Furthermore, there is more work about similarity between tables such as [5], where the authors propose methods to identify tables related to a given query table in a large corpus. Another class of work focuses on understanding the content of Web tables by trying to match table columns to an existing knowledge base such as [9, 13]. These works are complementary to ours: correct labels on candidate tables could be used for more precise similarity functions between tables, which would increase both consistency of single covers, as well as the diversity of the result lists.

Moving away from Web tables, there is a large body of related on general and web data fusion and truth discovery from multiple conflicting or correlating sources, see [2, 8] for surveys. This class of work is concerned with fusing many sources for a fact to single truth, while we aim at providing "alternative truths" for a set of facts using a minimal number of consistent sources. In [6], a similar argument regarding minimizing the number of sources to consider is made, though there the aim is to reduce the search space for data fusion techniques before applying them, while the aim in our case is to create results minimal covers of an entity set to make the result easier to understand and verify. In general, many methods from this area could be applied to our problem to prune low quality candidate tables, and to improve our candidate quality scoring.

Our work is also based on automatic schema matching. The survey [11] gives a good introduction to this large field. Diversity in search results is often studied in Web search and recommender systems research. A good overview is given in [7]. Finally, the set cover problem is a problem that has been extensively studied in theoretical computer science and in operations research with a wealth of works published. We only want to refer to [1], as our genetic set covering approach is inspired by this work, which treats the problem of how to represent and how to cross solutions to the set cover problem so that the new solutions are feasible set covers as well.

6. CONCLUSION

We have argued that entity augmentation queries based on large corpora of data sources should be processed as Top-k queries, while ensuring consistency and minimal size of the individual augmentations and the diversity of the result list as a whole. We presented new algorithms for *consistent, multi-solution set covering*, and then applied them to Web table-based entity augmentation. This publication includes our Web table corpus with 112M tables, as well as the source code of our Web table retrieval and matching system used to evaluate these algorithms.

Our experiments show that our genetic set covering-based approach improves both consistency and minimality of the results significantly, without loss of precision or coverage, and while producing a diverse set of results for the user to choose from. Possible future work includes a more theoretic analysis of the top-k consistent set cover problem, as well the investigation of the applicability of further set covering heuristics.

7. REFERENCES

[1] J. Beasley and P. Chu. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94(2):392–404, 1996.

[2] J. Bleiholder and F. Naumann. Data fusion. *ACM Comput. Surv.*, pages 1–41, 2009.

[3] M. J. Cafarella, A. Halevy, and N. Khoussainova. Data integration for the relational web. *VLDB*, pages 1090–1101, 2009.

[4] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *VLDB*, pages 538–549, August 2008.

[5] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu. Finding related tables. In *SIGMOD*, pages 817–828, 2012.

[6] X. L. Dong, B. Saha, and D. Srivastava. Less is more: selecting sources wisely for integration. In *VLDB*, pages 37–48, 2013.

[7] M. Drosou and E. Pitoura. Search result diversification. *SIGMOD Rec.*, pages 41–47, 2010.

[8] X. Li, X. L. Dong, K. Lyons, W. Meng, and D. Srivastava. Truth finding on the deep web: is the problem solved? In *VLDB*, pages 97–108, 2013.

[9] G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and searching web tables using entities, types and relationships. *VLDB*, pages 1338–1347, 2010.

[10] R. Pimplikar and S. Sarawagi. Answering table queries on the web using column keywords. *VLDB*, pages 908–919, 2012.

[11] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB*, pages 334–350, 2001.

[12] S. Sarawagi and S. Chakrabarti. Open-domain quantity queries on web tables: Annotation, response, and consensus models. In *KDD*, pages 711–720, 2014.

[13] P. Venetis, A. Halevy, J. Madhavan, M. Paşca, W. Shen, F. Wu, G. Miao, and C. Wu. Recovering semantics of tables on the web. *VLDB*, pages 528–538, 2011.

[14] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *SIGMOD*, pages 97–108, 2012.

[15] X. Yin, W. Tan, and C. Liu. Facto: a fact lookup engine based on web tables. In *WWW*, pages 507–516, 2011.

[16] M. Zhang and K. Chakrabarti. Infogather+: semantic matching and annotation of numeric and time-varying attributes in web tables. In *SIGMOD*, pages 145–156, 2013.