# Automatic Generation of Ontologies from Canonicalized Web Tables

Stephen Lynn and David W. Embley*

Brigham Young University, Provo, Utah 84602, U.S.A.

**Abstract.** Ontology creation is a daunting task—manual creation is tedious and time consuming, and automatic creation is disappointingly inaccurate. But for applications such as the semantic web or making web content directly queriable, we must facilitate ontology creation, making it reasonable to produce the vast number and variety of ontologies required for future web applications. In this paper we describe a tool developed to automate the generation of ontologies from ordinary web tables. The process is akin to reverse-engineering relational tables to conceptual models, but must account for a much greater variety of table layout patterns. The tool uses auxiliary knowledge and heuristic rules to select, enhance, and modify ontology elements discovered in tables. Although designed to operate fully automatically, the tool allows users to intervene to direct and correct when necessary so that they can always acquire a satisfactory ontology from a given web table. Experimental evaluations show that the automatic ontology acquisition process can perform well, yielding F-measures of 90% for concept recognition, 77% for relationship discovery, and 90% for constraint discovery in web tables selected from the geopolitical domain.

## 1 Introduction

From libraries filled with millions of books to the Internet available for anyone with a web browser, the amount of information available in the world is growing exponentially. With this information explosion comes new challenges in organizing and finding information relevant to a user's needs. Most of the available information does not follow any consistent format or structure, making it difficult to extract in a way that supports queries beyond common keyword searching. One possible solution to this problem is structuring the information on the web into standardized ontologies which represent the inherent concepts, relationships, and constraints found in the information. Exposing the information in an ontological model enables an entire new class of search algorithms allowing queries to be expressed more completely and more explicitly, well beyond anything currently available in standard keyword searches available today.

Few people and organizations use ontology-based representations to organize information on the web, however, because creating an ontology takes too much

* Supported in part by the National Science Foundation under Grant #0414644.

**Region and State Information**

| Location | Population (2000) | Latitude | Longitude |
|---|---|---|---|
| **Northeast** | 2,122,869 | | |
| **Delaware** | 817,376 | 45 | -90 |
| **Maine** | 1,305,493 | 44 | -93 |
| **Northwest** | 9,690,665 | | |
| **Oregon** | 817,376 | 45 | -90 |
| **Washington** | 817,376 | 45 | -90 |

**Fig. 1.** Sample Table.

time and effort and requires a high degree of expertise. TANGO (Table ANalysis for Generating Ontologies [16]) is a project aimed at reducing the time, effort, and degree of expertise needed by automating the process of creating an ontology from the concepts, relationships, and constraints found in sets of tabular data. A component of the overall TANGO project is to transform an individual table into a conceptual model. We call the conceptual model acquired from a table a *mini-ontology*, "mini" because it is small relative to an ontology for an entire domain and "ontology" because the conceptual model is formal, based fundamentally on predicate calculus but limited to be equivalent to description-logics over which reasoners can operate. We call our tool MOGO (a Mini-Ontology GeneratOr); it implements the necessary algorithms and user interfaces for automatically, semi-automatically, or manually generating mini-ontologies from canonicalized tables of data.[1]

Given a table like the one in Figure 1, MOGO generates a conceptual model (a mini-ontology) that accurately represents the table of data by iterating through a number of heuristic rules. Each heuristic deals with one of three main tasks: concept recognition, relationship discovery, and constraint discovery. During each step of the process, MOGO populates the conceptual model with the data in the original table. Figure 2 shows the conceptual model (mini-ontology) MOGO

[1] The first component of the TANGO project reorganizes raw tables found on the web and elsewhere and into tables in a canonical form. The third component merges groups of mini-ontologies into a large ontology representing some body of knowledge.

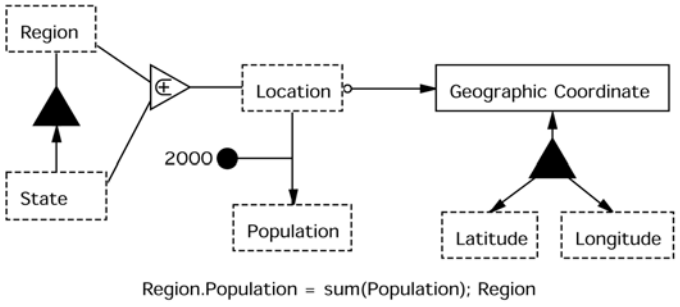


**Fig. 2.** Generated Mini-ontology for the Sample Table in Figure 1.

generates from the table in Figure 1. The four US states in Figure 1 are members of the *State* object set in Figure 2. The two regions are in the *Region* object set. Together the regions and states constitute the elements of the *Location* object set. The states aggregated together constitute the different regions. The values in the population, latitude, and longitude columns of the table in Figure 1 are members of the *Population*, *Latitude*, and *Longitude* object sets respectively. Latitude and longitude values aggregated together constitute the *Geographic Coordinate* object set. Each US state has a population and a geographic coordinate, and each region has a population computed as the sum of the populations from the states in the region.

Our MOGO/TANGO approach to generating ontologies differs fundamentally from typical research on ontology generation, which focuses on "learning" ontologies by finding the terms, concepts, relations, and concept hierarchies existing in large collections of unstructured text documents (see [4], a recent survey that summarizes much of this research). The lack of structure and appropriate metadata in these documents has so far made these approaches less than accurate, thus requiring significant human post-processing before the results can be used. More closely related to our approach is the reverse engineering of relational database tables into conceptual models [1–3, 5, 11, 12, 14]. Relational tables, however, have far fewer degrees of freedom than unconstrained web tables and reverse-engineering techniques for relational tables rely mainly on the regularity and constraints of database schemas.

In an effort quite similar to our research with MOGO, Pivk et al. [13] have implemented as a system called TARTAR, which takes ordinary tabular data as the input and produces F-Logic frames as output. TARTAR focuses primarily on using pattern-learning and statistical methods for string recognition and grouping to discover concepts and relationships in a table. Our approach makes use of some similar pattern matching heuristics but also includes a strong emphasis on heuristics employing linguistic clues and narrowly defined data-type recognizers to discover concepts, relationships, and constraints in a table.

Our contribution in this paper is a tool, called MOGO, that can generate mini-ontologies from canonicalized web tables of data. We provide the details of our contribution as follow. In Section 2 we describe our implementation including an architectural overview, as well as detailed explanations of each of the heuristics MOGO uses to generate conceptual models from tables. In Section 3, we give experimental results, which indicate the success of our approach, and we make concluding remarks in Section 4.

## 2 Mini-Ontology Generation

MOGO takes as input canonicalized tables of data based on Wang notation [17]. This notation preserves the labels found in the source table as well as their associated data values. The notation organizes label information in simple data structures called dimensions. Each dimension corresponds to a different axis of the table similar to the different axes of a multi-dimensional array. Combining

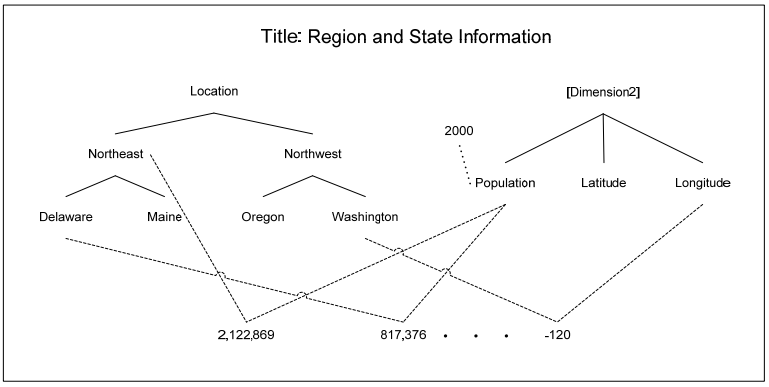


**Fig. 3.** Graphical View of the Sample Table in Figure 1 in Canonical Form.

these dimensions allows every data cell to be referenced using an element from each dimension. Because Wang notation can represent any set of tabular data independent of layout, MOGO is agnostic to the data's original form. To further enhance MOGO's ability to produce a useful mini-ontology, we enhance standard Wang notation so that information beyond just row and column labels and data values is preserved in a canonicalized form. These enhancements include the identification of a table's title, caption, and footnotes as well as row, column, and value augmentations such as units of measure.[2] We capture canonicalized tables in XML documents.

Based on canonicalized input data, MOGO produces a mini-ontology that conforms to the OSM data modeling language [8]. Thus the input to MOGO is a canonicalized table in an XML document, and the output of MOGO is a conceptual model in OSM. MOGO operates automatically and, after producing an OSM model instance, allows a user to accept the mini-ontology as produced, make adjustments to the mini-ontology using the OSM ontology editor, or manually rebuild the mini-ontology with the OSM ontology editor.

We illustrate how MOGO works with the table of geopolitical data in Figure 1. We compiled a small amount of data from multiple tables to create a single sample table that illustrates the various facets of MOGO's processing abilities. Figure 3 shows a graphical representation of the canonicalized table in Figure 1. Each dimension of the table forms a tree. The second dimension in the canonicalized table has no label value, so we add a placeholder label, *[Dimension2]*. Data values, at the bottom of the figure, connect to one node from each dimension via a dashed line. The dotted line connecting the *Population* node and the value *2000* indicates that the *2000* is a value augmentation of the *Population* node. The representation also includes the title of the table.

[2] For the TANGO project, we are developing a canonicalizing tool [10]. We have also developed TISP [15], which can successfully canonicalize tables on the hidden web, where sibling tables are available in rich abundance.

### 2.1 Auxiliary Services

Many of the algorithms MOGO uses require access to external lexical information. Rather than tie the system directly to a specific implementation of some lexical resource, MOGO establishes an implementation independent lexical service interface. Supported generic operations include term normalization, and testing whether one word is a hypernym, hyponym, meronym, or holonym of another word. Term normalization, for example, allows the system to treat different word forms such as "Iowa," "Hawkeye State," and "IA" as equivalent words, and hypernym checking allows the system to recursively check for term generalizations. In our current implementation, MOGO uses WordNet [9] as its backend lexical resource.

Another service MOGO uses is a data-frame library. Data frames provide a mechanism for recognizing different types of objects from strings of data representations using regular-expression recognizers [7]. MOGO's data frame library service takes a string as input, iterates over a collection of data-frame recognizers attempting to classify the string, and returns the data frame that matches that string. To illustrate how this works, suppose the string 12-08-2007 needs to be classified. In looking for a match MOGO's data frame service discovers that the *Date* data frame recognizes dates in the form MM-DD-YYYY. The data frame service returns the specific object set (concept) to which the search terms match, as well as any ontology fragment associated with the *Date* data frame.

The final general service MOGO provides is a name finding service available at each step of the process for assigning names to unnamed concepts. Titles, footnotes, captions, and augmentations can contain words that are helpful for naming unnamed concepts. The combined set of words from these sources forms a pool of possible concept names. Given an unnamed concept, MOGO uses the lexical service to retrieve the inherited hypernym list of each value assigned to an unnamed concept, compares the list with each of the words in the naming pool, and assigns the concept a name if one of the words in the pool is a direct match to a word in the hypernym list. If this search does not find any matches, MOGO attempts to identify an appropriate label for an unnamed concept by looking for the first common word in the inherited hypernym lists of a selection of the concept's data values. If a common word is found, MOGO assigns that word as the concept's name.

### 2.2 Concept/Value Recognition

MOGO extracts concepts from a canonicalized table using six concept-recognition algorithms (*CR#1 – CR#6*) and appropriately assigns data values from the table to those concepts. We execute each algorithm in the order given below until each table label and table data value of the canonicalized table is recognized as either a concept or a value for a concept. Note that table labels can either be concepts or data values for a concept. In Figure 1, the label *Delaware* is a data value for the concept *State* and *Northwest* is a data value for the concept *Region*,

but the label *Population* is a concept containing population values. Unlike table labels, table data values are always data values for some concept.

A concept is synonymous with an object set in the OSM data modeling language. According to OSM an object set identifies a group of objects or values [8]. Object sets, either lexical or non-lexical, are the ontological elements representing the concepts found in a table. A lexical object set is one whose members are displayable and represent themselves (e.g., telephone numbers, names of companies). In OSM a lexical object set appears visually as a box with a dashed border. A non-lexical object set's members are object identifiers (e.g., identifiers that stand for persons or companies). In OSM a non-lexical object set appears visually as a box with a solid border.

*CR#1.* The first concept-recognition algorithm uses lexical clues to determine to which dimension labels the tables data values belong. A data value is said to *belong* to a label if the data value is a hyponym of at least one of the labels senses, and is not a hyponym of any other dimension label associated with that data value. If the majority of the data values belong to an associated label, MOGO flags the label as a potential object set. After evaluating all the dimension labels, if all the labels MOGO flags are part of the same dimension, then MOGO marks all the labels in that dimension as lexical object sets and associates the corresponding data values with those object sets. *CR#1* fails for the table in Figure 1 because the data values in the table are numbers and there is no way to determine, using only lexical clues, if those numbers belong to their associated labels. *CR#1* would succeed, for example, for a table of cities (e.g., Salt Lake City, Los Angeles, San Francisco) and states (e.g., Utah, California) whose labels are *City* and *State*.

*CR#2.* The second concept-recognition algorithm also uses the lexical service, but in this case the objective is to determine if a label is an instance of its parent label. Each dimension has one label referred to as the root label. In Figure 3, for example, *Location* is the root of the first dimension. Below the root, a dimension can contain several levels of label nesting. A label is said to *be an instance of its parent label* if the parent label is in the label's inherited hypernym list. If the majority of the labels at one level of label nesting are instances of that level's parent label, MOGO marks them all as values and assigns the values to a lexical object set created to hold them. In dimensions with only one level of label nesting, MOGO names the created object set with the dimension's root label (if present). When a dimension has several levels of nesting, MOGO uses the name finding service to find appropriate names for object sets. For the sample table in Figure 1, MOGO creates lexical object sets for the second and third level of nesting for the *Location* dimension. The inherited hypernym list for *Northeast* and *Northwest* contains *Region*, which also appears in the title of the table, so *Region* becomes the label for the object set containing the values *Northeast* and *Northwest*. Similarly, each of the inherited hypernym lists for the state values contains the word *State* which is also a word in the title of the table, so *State* becomes the label for the object set containing *Delaware*, *Maine*, *Oregon*, and *Washington*.

| | 2002 | 2003 |
|---|---|---|
| **Province** | **Nr. of Deaths** | |
| **Quebec** | 54,896 | 56,411 |
| **Ontario** | 83,410 | 84,155 |

**Fig. 4.** Table with a Label Spanning Multiple Columns.

*CR#3.* The third algorithm checks for labels at the same level of nesting that have the same name. This is usually manifest in tables with labels that span multiple columns or rows (which are replicated for each column or row in canonicalized tables). In the table in Figure 4, for example, *Nr. of Deaths* spans the two year columns and appears, canonicalized, as two labels with the same name at the same level of nesting. If all the labels at one level of label nesting are the same, MOGO creates a named object set using the common name of the source labels and assigns all the values associated with those labels to the newly created object set. For the sample table in Figure 4, MOGO creates the object set *Nr. of Deaths* and assigns it all four numeric values.

*CR#4.* The fourth concept-recognition algorithm attempts to classify all the data values in a row or column with a header label using MOGO's data-frame service. If separate data frames uniquely recognize data values in at least two rows or columns for the same dimension, MOGO marks all the labels in the dimension as lexical object sets and associates them with their corresponding data values. Requiring at least two different data frames to recognize data values in at least two different rows or columns prevents MOGO from misidentifying object sets in a table uniformly populated by data of the same type such as a table full of percentages or of currency values. Data frames for *Latitude* and *Longitude*, for instance, could recognize the values in the last two columns of the table in Figure 1. In which case, MOGO would correctly classify the latitude and longitude values in the table and also the population values. For the sake of continuing the example, however, we assume that data-frame recognizers fail to make this classification (perhaps because there are no latitude and longitude data frames, or perhaps because the numbers in the latitude and longitude columns are written so generally that multiple data frames recognize them and thus the latitude and longitude data frames fail to uniquely enough identify the columns).

*CR#5.* The fifth concept-recognition algorithm applies data-frame recognizers to sibling labels, such as the year labels in Figure 4. For each group of sibling labels that have the same data-frame classification, MOGO accepts the labels as values, creates a lexical object set, names the object set with the data-frame's concept name, and associates the sibling labels (as values) with the new object set. Thus, for example, the year labels in Figure 4 would become year values for an object set *Year*.

*CR#6.* If algorithms CR#1–#5 do not successfully classify all items in the canonicalized table as either object sets or values for object sets, MOGO creates lexical object sets from all the unclassified labels in any dimension whose root node contains a placeholder label (e.g., *[Dimension 2]* in Figure 1) and assigns to them any unassigned data values associated with those labels. For any unclassi-

fied labels in the remaining dimensions, MOGO groups the labels that are at the same level of nesting in each dimension, treats the labels as values, creates unnamed object sets for them, and uses the name finding service to name them. For any remaining unassigned data values, MOGO creates a new unnamed object set for these values. For the sample table in Figure 1, this concept-recognition algorithm creates lexical object sets for each of the labels under *[Dimension2]* and assigns them their associated data values—the values under them in the three columns.

### 2.3 Relationship Discovery

With concepts identified and values assigned to those concepts, MOGO next identifies the relationships that exist among the concepts. MOGO adds relationship information to the object sets using five relationship discovery algorithms (*RD#1*–*RD#5*). Unlike the concept-recognition algorithms, which only run until all labels and values have been classified, all relationship recognition algorithms run—each successively refines the results of the previous algorithm.

*RD#1.* The first relationship-discovery algorithm obtains relationship information from the dimension trees. For a dimension, MOGO creates relationship sets between the object sets from the dimension anywhere an edge exists in the dimension tree. When labels at one level of nesting have been merged into a single object set, MOGO only creates one relationship set between the parent object set and the child object set. If sibling object sets (object sets coming from labels at the same level of label nesting) do not have any related object sets higher in the tree, MOGO creates an object set of unknown type, labels it with the dimension's name (if there is one), and creates relationship sets between this new object set and each of the sibling object sets. Figure 5 shows the relationship sets MOGO adds between the different object sets for our running example beginning with Figure 1. MOGO associates the *Region* and *State* object sets because they come from different levels of the same dimension—*State* from the leaf level and *Region* from the intermediate level of the *Location* tree in Figure 3. The *Population*, *Latitude*, and *Longitude* object sets are sibling object sets whose parent object set is the placeholder *[Dimension 2]* meaning that *Population*, *Latitude*, and *Longitude* have no identified conceptual parent object set. In this case, MOGO creates an object set of unknown type, and associates the sibling object sets with the newly created object set. We represent object sets of unknown type visually as a shaded box with no border.

*RD#2.* The second relationship-discovery algorithm modifies the generated ontology relationship sets using lexical clues. MOGO checks to see if the labels or values indicate the presence of hypernyms, hyponyms, holonyms, and meronyms. Hypernyms and hyponyms translate to generalization/specialization relationships (represented by a triangle). Holonyms and meronyms translate to aggregation relationships (represented by a filled-in triangle). If aggregations are found between the object sets from one dimension, MOGO checks for generalization/specializations that might exist over the entire aggregation. Using its lexical service, MOGO finds the inherited hypernym list of each object-set label

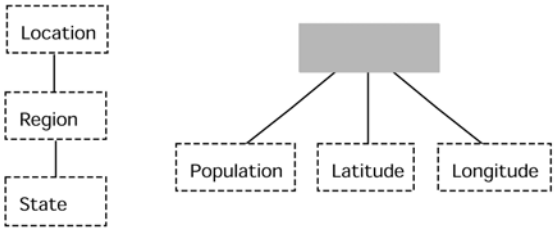


**Fig. 5.** Relationship Sets from Dimension Trees.

participating in the aggregation. If the dimension's root label is in the inherited hypernym lists of all the object sets in the aggregation, MOGO creates a new lexical object set, labels it with the dimension's root label, and associates this new object set with each of the object sets that participate in the aggregation using generalization/specialization. For our running example, Figure 6 shows the results after MOGO modifies them using lexical clues. MOGO finds that *Delaware* is an instance of an *American State* which is a hyponym of *Region*. Thus, MOGO creates an aggregation constraint from the *Region* object set to the *State* object set. Because the *Region* and *State* object sets come from the same dimension, MOGO checks to see if the dimension's root label is in the inherited hypernym list of those object sets. MOGO successfully finds the root label *Location* in the inherited hypernym lists, so it transforms the root object set into a generalization and associates it with the aggregation object sets as specializations.

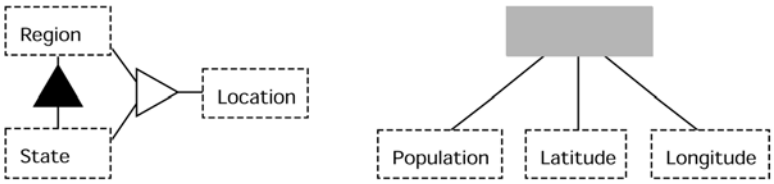


**Fig. 6.** Relationship Sets after Linguistic Processing.

*RD#3.* The third relationship-discovery algorithm uses MOGO's data-frame service to find relationships between the object sets. MOGO applies data-frame recognizers to match object sets with data frames. Then, if MOGO finds object sets that match different concepts in the same ontology fragment, it replaces these object sets with the discovered ontology fragment. In our running example, MOGO discovers that the two object sets *Latitude* and *Longitude* in Figure 6 belong to the same ontology fragment—*Geographic Coordinate*. MOGO therefore removes the existing *Latitude* and *Longitude* object sets, adds the *Geographic Coordinate* ontology fragment, and transfers the relationship sets previously connected to *Latitude* and *Longitude* to the primary object set of the ontology fragment, which in this case is the *Geographic Coordinate* object set. Figure 7 shows the result.

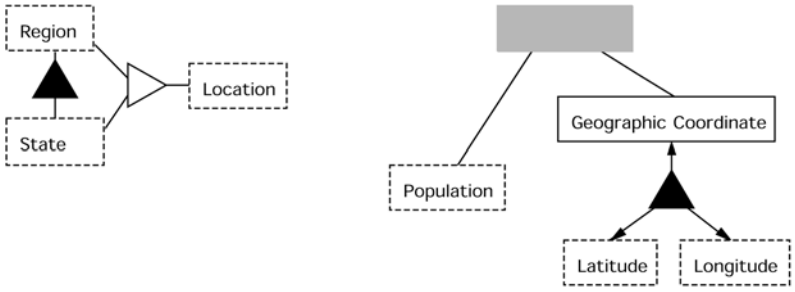


**Fig. 7.** Relationship Sets after Data Frame Recognizers.

*RD#4*. The fourth relationship-discovery algorithm processes any augmentations that exist in the canonicalized table. For each row and column augmentation that is a value (not a unit, footnote, or a parenthetical remark) as indicated in the canonicalized table, MOGO creates a singleton object for the value and forms an $n$-ary relationship set among the singleton object and the object sets associated with the augmentation. For example, Figures 1 and 3 show that *Population* has the value *2000* as an augmentation. Thus, as Figure 8 shows, MOGO creates a singleton object whose value is *2000* and creates a ternary relationship set among the object of value *2000*, the *Population* object set, and the unnamed object set already related to the *Population* object set.

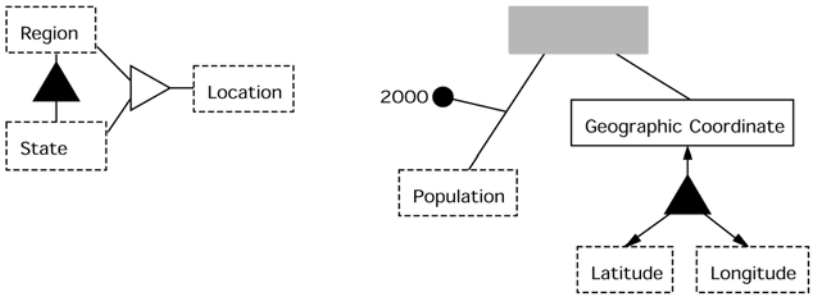


**Fig. 8.** Relationship Sets after Processing Augmentations.

*RD#5*. The final relationship-discovery algorithm joins the ontology fragments for each of the tables $n$ dimensions into a single mini-ontology. MOGO joins the fragments by creating an $n$-ary relationship set among the fragment link-in points. An ontology fragment's link-in point is the object set in the fragment that comes from the highest level in the dimension—typically the object set associated with the dimension's root label. If one of the link-in points is a placeholder object set and there is only one other ontology fragment, MOGO merges the placeholder object set with the link-in object set of the other ontology fragment. Figure 9 shows the result of merging the two ontology fragments in Figure 8.

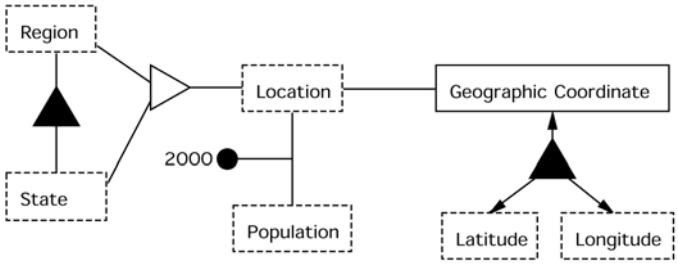


**Fig. 9.** Mini-ontology Results from Fragment Merge.

### 2.4 Constraint Discovery

MOGO has four constraint-discovery algorithms (*CD#1–CD#4*). Each checks for a single kind of constraint; if an algorithm finds that the constraint holds, MOGO adds the constraint to the mini-ontology being created.

*CD#1.* The first constraint-discovery algorithm adds constraints to generalization/specialization relationships. It considers generalization/specialization relationships identified by step *RD#2* along with the values in the table that have been assigned to object sets by steps *CR#1–6*. MOGO constrains a generalization/specialization relationship to have a union constraint (represented by a triangle containing a $\cup$) if all values in the generalization object set are also in at least one of the specialization object sets. MOGO constrains a generalization/specialization to have a mutual-exclusion constraint (represented by a triangle containing a $+$) if there is no overlap in the values in the specialization object sets. When the generalization/specialization is constrained by both union and mutual exclusion, MOGO assigns a partition constraint (represented by a triangle containing a $\uplus$, the symbol with both a $\cup$ and a $+$) to the relationship. The $\uplus$ in Figure 2 appears as a result of running this algorithm. MOGO adds the $\uplus$ because it determines that every value assigned to the *Location* object set is also assigned to the either the *Region* object set or the *State* object set, and further that the intersection of the values in the *Region* and *State* object sets is empty.

*CD#2.* The second constraint-discovery algorithm looks for computed values in a table. Tables often include columns or rows that contain the summation, average, or other aggregates of values in the table. MOGO examines the values indexed by non-leaf nodes in the dimensions such as *2,122,869* which is indexed by *Northeast* as Figure 3 shows. By computing aggregates of values from related object sets and comparing them to these values to test whether the aggregates hold, MOGO captures these constraints and adds them as to the mini-ontology. Looking for possible aggregate values, MOGO determines that the *Population* values for the *Region* values are the summation of the *Population*

values for the *State* values. MOGO thus ads the constraint *Region.Population = sum(Population); Region*[3] to the mini-ontology as Figure 2 shows.

*CD#3.* The third constraint-discovery algorithm looks for functional relationships. Each of the table's original data values is functionally determined by the dimension labels that index those values. Thus, MOGO identifies the object sets that contain the table's data values and marks the relationship sets coming into those object sets as functional. In Figure 2, the arrowheads on the relationship set coming from *Location* into *Population* and on the relationship set coming from *Location* into *Geographic Coordinate* appear as a result of running this algorithm—*Location* functionally determines *Geographic Coordinate* and, along with the year, functionally determines *Population*. MOGO processes object sets assigned values from table labels differently—it evaluates each of these object sets to see if the values assigned to the object set functionally map to values assigned to any related object sets (i.e., it checks each domain value or combination of domain values to see if there is at most one range value). If so, MOGO marks the relationship set as functional. Since the *Region* and *State* object sets contain values from dimension labels, and since the values assigned to the *State* object set functionally map to the values assigned to the *Region* object set (i.e., there is only one region for each state), MOGO marks the relationship set from *State* to the aggregation connecting it to *Region* as functional as Figure 2 shows.

*CD#4.* The final constraint-discovery algorithm determines whether objects in an object set participate mandatorily or optionally in associated relationship sets. Optional participation is represented in OSM by the symbol o (an "o" for optional) placed near the object set's connection point to a relationship-set line. MOGO identifies object sets whose objects have optional participation in relationship sets by considering empty value cells in the canonicalized table. MOGO determines where these non-existing values are in the mini-ontology and marks participation in any relationship sets adjoining these relationship sets as optional. Figure 2 shows that MOGO discovers that *Location* optionally participates with *Geographic Coordinate* because some locations, namely *Northeast* and *Northwest* have no associated latitude and longitude values.

## 3 Experimental Evaluation

We evaluated MOGO using a test set of tables found on the Internet by a third-party participant. We asked the participant to capture the URL of twenty different web pages that contain tables. Because tables can vary drastically in form and complexity, we asked that the test tables meet the following criteria: the tables should come from at least three distinct sites; the tables should contain a mix of simple tables (one dimensional with no label nesting) and complex tables (multi-dimensional with or without label nesting); and the tables should all be from the geopolitical domain. For each test URL gathered by the participant, we

[3] The notation here means that a region's population is the sum of the population values grouped by *Region*; it is adapted from [6], which defines computational expressions over conceptual models.

saved a local copy of the page's source HTML and used the tools created in the first component of the TANGO project [10] to canonicalize the tables. MOGO processed each of the twenty canonicalized tables and saved the resulting mini-ontologies for evaluation. We evaluate each mini-ontology in three different areas: concept/value recognition, relationship discovery, and constraint discovery.[4]

*Concept/Value Recognition.* Every table has a fixed number of concepts to which the data values belong. We observe how many concepts MOGO correctly identifies, how many it misses, and how many concepts it identifies that are not really concepts. In addition, each concept has a label. We observe how many labels MOGO correctly assigns and how many it incorrectly assigns. We also observe how many data-value groups (i.e., data rows, data columns, label-data groups) are assigned to a correct concept and how many are incorrectly assigned. We compute precision values for concept/value recognition by dividing the total number of correct concepts, labels, and data-value groups MOGO finds by the total number of actual concepts, labels, and data-value groups combined with the total number of incorrect concepts, labels, and data-value groups MOGO finds. We compute recall values for concept/value recognition by dividing the total number of correct concepts, labels, and data-value groups MOGO finds by the total number of actual concepts, labels, and data-value groups in the canonicalized table.

*Relationship Discovery.* We evaluate relationship discovery by observing how many valid relationship sets MOGO discovers, how many relationship sets it discovers that are invalid, and how many valid relationship sets MOGO does not discover. Additionally, we observe how many aggregations and generalization/specializations MOGO discovers, how many it discovers that are invalid, and how many it does not discover. In cases where a relationship set, aggregation, or generalization/specialization should exist but does not because MOGO does not correctly identify a concept, we count the missing relationship set, aggregation, or generalization/specialization as one that MOGO does not discover. We compute precision values for relationship discovery by dividing the total number of correct relationship sets, aggregations, and generalization/specializations MOGO finds by the total number of actual relationship sets, aggregations, and generalization/specializations combined with the total number of incorrect relationship sets, aggregations, and generalization/specializations MOGO finds. We compute recall values for relationship discovery by dividing the total number of relationship sets, aggregations, and generalization/specializations MOGO finds by the total number of actual relationship sets, aggregations, and generalization/specializations in the canonicalized table.

*Constraint Discovery.* We evaluate constraint discovery by observing how many valid constraints MOGO discovers, how many invalid constraints it discovers, and how many valid constraints MOGO does not discover. Observations are

[4] It is necessary to point out that when building ontologies, there is often no "right" answer. For most tables there can be multiple ontologies that are valid conceptualizations of the data. Our manual evaluation permitted only valid conceptualizations, but did allow for reasonable alternatives.

| | Precision | Recall | F-measure |
|---|---|---|---|
| Concept Recognition | 87% | 94% | 90% |
| Relationship Discovery | 73% | 81% | 77% |
| Constraint Discovery | 89% | 91% | 90% |

**Table 1.** Precision, Recall, and F-measure for Experimental Evaluation.

made for each of the following types of constraints: generalization/specialization constraints, functional dependencies, computed values, and optional participation of objects in associated relationship sets. In cases where constraints should exist but do not because MOGO does not correctly identify a concept or relationship set, we count the missing constraint as one that MOGO does not discover. We compute precision values for constraint discovery by dividing the total number of correct constraints MOGO finds by the total number of actual constraints combined with the total number of incorrect constraints MOGO finds. We compute recall values for constraint discovery by dividing the total number of constraints MOGO finds by the total number of actual constraints in the canonicalized table.

Table 1 shows the precision and recall values for each evaluation. For the experimental test, MOGO achieved 87% precision and 94% recall for the concept-recognition task, 73% precision and 81% recall for the relationship-discovery task, and 89% precision and 91% recall for the constraint-discovery task. As a combined measure of precision and recall, we add F-measures to Table 1. Concept recognition and constraint discovery both have an F-measure of 90% while relationship discovery has an F-measure of 77%.

Unfortunately, a direct comparison of MOGOs results with results achieved by TARTAR [13]—the only other similar system for converting tables to conceptual models we know about—is not possible. TARTAR's maximum F-measure of 74.18% should not be construed to indicate that TARTAR has worse performance than MOGO. Not only are TARTAR's results measured in a different way, but TARTAR's results also take into account both the table canonicalization process as well as the conversion to a conceptual model. MOGOs results, on the other hand, are based on a set of canonicalized tables that were checked for accuracy before being processed by MOGO. For some tables, in particular sibling tables found in hidden-web pages, we can automatically canonicalize in preparation for running MOGO with F-measures running as high as 94.5% [15]. Hidden-web pages, however, are not arbitrary pages and their characteristics simplify the general canonicalization problem and thus render comparison with TARTAR's results problematic even when we run our hidden-web-page cannonicalizer in series with MOGO. As a hypothesis for further work, we believe that a combination of TARTAR's pattern-learning and statistical methods and MOGO's linguistic/type-recognition/spatial-layout methods would result in an enhancement of both TARTAR and MOGO.

As a result of our experimental evaluation, we observed a number of issues, which we report here.

*Duplicate Concepts.* Some tables have multiple columns that corresponded to the same concept. For example, a table about mountain peaks can contain two columns labeled height—in one column the height appears in meters and in the other column the height appears in feet. MOGO is currently unable to correctly merge these concepts into one.

*Concept Labeling.* Sometimes a valid label for the concept does not exist in the table. Many tables assume that the reader can infer the correct label based on the context in which the table occurs. In some cases, such as a table containing an unlabeled column of country names, MOGO is able to successfully identify a valid label using its lexical service. In other cases, such as an unlabeled column of numbers (perhaps under discussion in the text surrounding the table), MOGO cannot identify a label for the concept that contains these values.

*Aggregates and Generalization/Specializations.* MOGO only looks for aggregate and generalization/specialization constructs when there is label nesting present in a dimension. MOGO's heuristics, for example, do not currently check for aggregates in cases such as when a table has a column of county names and another column of state names.

*Functional Constraints.* Canonicalized table values sometimes contain lists of values instead of a single value. Currently, MOGO incorrectly treats these lists of values as singleton objects and usually discovers that lists are unique and thus that relationship sets connecting to object sets containing these lists are functional.

*Totals not Associated with Aggregates.* When a column or row only contains values that are the computed sums or averages of the other values in the table, MOGO does not correctly identify these computed values. Currently, only when the row or column represents a conceptual aggregation, such as a state population containing the computed sum of the populations of the counties in that state, does MOGO correctly identify the computed value.

*Cascading Errors.* Errors in earlier phases of mini-ontology generation have a cascading effect on errors in later parts of the process.

## 4 Concluding Remarks

We have created a system called MOGO that automates the acquisition (i.e., the generation) of mini-ontologies from canonicalized tables of data. MOGO uses a novel approach to ontology generation by combining spatial observations, linguistic clues, and narrowly defined data-instance recognizers (data frames) for generating conceptual models. Experimental results show that MOGO is able to automatically identify the concepts, relationships, and constraints that exist in arbitrary tables with a relatively high level of accuracy—with F-measures of 90%, 77%, and 90% respectively for concept/value recognition, relationship discovery, and constraint discovery in web tables selected from the geopolitical domain.

## References


1. R. Alhajj. Extracting the extended entity-relationship model from a legacy relational database. *Information Systems*, 28(6):597–618, 2003.
2. S.M. Benslimane, D. Benslimane, and M. Malki. Acquiring OWL ontologies from data-intensive web sites. In *Proceedings of the 6th International Conference on Web Engineering*, pages 361–368, Palo Alto, California, July 2006.
3. R.H.L. Chiang, T.M. Barron, and V.C. Storey. Reverse engineering of relational databases: Extraction of an EER model from a relational database. *Data and Knolwedge Engineering*, 12(1):107–142, 1994.
4. P. Cimiano. *Ontology Learning and Population from Text: Algorithm, Evaluation and Applications*. Springer Verlag, New York, New York, 2006.
5. I. Comyn-Wattiau and J. Akoka. Reverse engineering of relational database physical schema. In *Proceedings of the 15th International Conference on Conceptual Modeling*, pages 372–391, Cottbus, Germany, October 1996.
6. B. Czejdo and D.W. Embley. An approach to computation specification for an entity-relationship query language. In *Proceedings of the 6th Entity-Relationship Conference*, pages 307–321, New York, New York, November 1987.
7. D.W. Embley, D.M. Campbell, Y.S. Jiang, S.W. Liddle, D.W. Lonsdale, Y.-K. Ng, and R.D. Smith. Conceptual-model-based data extraction from multiple-record web pages. *Data & Knowledge Engineering*, 31(3):227–251, November 1999.
8. D.W. Embley, B.D. Kurtz, and S.N. Woodfield. *Object-oriented Systems Analysis: A Model-Driven Approach*. Prentice Hall, Englewood Cliffs, New Jersey, 1992.
9. C. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, Massachussets, 1998.
10. P. Jha. Interactive wang notation tool for web tables. Technical report, Rensselaer Polytechnic Insititue, Troy, New York, May 2007. Available at: http://tango.byu.edu/.
11. P. Johannesson. A method for transforming relational schemas into conceptual schemas. In *Proceedings of the 10th International Conference on Data Engineering*, pages 190–201, Houston, Texas, February 1994.
12. N. Lammari, I. Comyn-Wattiau, and J. Akoka. Extracting generalization hierarchies from relational databases: A reverse engineering approach. *Data & Knowledge Engineering*, 63(2):568–589, 2007.
13. A. Pivk, Y. Sure, P. Cimiano, M. Gams, V. Rajkovič, and R. Studer. Transforming arbitrary tables into logical form with TARTAR. *Data & Knowledge Engineering*, 60:567–595, 2007.
14. W. Premerlani and M. Blaha. An approach to reverse engineering of relational databases. *Communications of the ACM*, 37(5):42–49, May 1994.
15. C. Tao and D.W. Embley. Automatic hidden-web table interpretation by sibling page comparison. In *Proceedings of the 26th International Conference on Conceptual Modeling*, pages 556–581, Auckland, New Zealand, November 2007.
16. Y.A. Tijerino, D.W. Embley, D.W. Lonsdale, Y. Ding, and G. Nagy. Toward ontology generation from tables. *World Wide Web: Internet and Web Information Systems*, 8(3):261–285, September 2005.
17. X. Wang. *Tabular Abstraction, Editing, and Formatting*. PhD thesis, University of Waterloo, 1996.