AUTOMATING MINI-ONTOLOGY GENERATION FROM CANONICAL TABLES

by

Stephen G. Lynn

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

August 2008

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Stephen Lynn

This thesis proposal has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

_____   _____
Date   David W. Embley, Chair

_____   _____
Date   Deryle Lonsdale

_____   _____
Date   Dan Ventura

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Stephen Lynn in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the University library.

_____   _____
Date   David W. Embley
                                 Chair, Graduate Committee

Accepted for the Department   _____
                                 Parris Egbert
                                 Graduate Coordinator

Accepted for the College   _____
                                 Thomas W. Sederberg
                                 Associate Dean
                                 College of Physical and Mathematical Sciences

ABSTRACT

Automating Mini-Ontology Generation from Canonical Tables

Stephen Lynn

Department of Computer Science

Master of Science

In this thesis work we develop and test MOGO (a Mini-Ontology GeneratOr.) MOGO automates the generation of mini-ontologies from canonicalized tables of data. This will help anyone trying to organize large amounts of existing data into a more searchable and accessible form. By using a number of different heuristic rules for selecting, enhancing, and modifying ontology elements, MOGO allows users to automatically, semi-automatically, or manually generate conceptual mini-ontologies from canonicalized tables of data. Ideally, MOGO operates fully automatically while allowing users to intervene to direct and correct when necessary so that they can always satisfactorily complete the translation of canonicalized tables into mini-ontologies. Experimental results show that MOGO is able to automatically identify the concepts, relationships, and constraints that exist in arbitrary tables of values with a relatively high level of accuracy. This automation significantly reduces the work required to translate canonicalized tables into mini-ontologies.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

From libraries filled with millions of books to the Internet accessible to 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 that is 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 Internet 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 today's standard keyword searches.

Few use ontology-based representations to organize information on the Internet because creating an ontology takes too much time and effort and requires a high degree of expertise. TANGO [21] is a project which will reduce 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. As the second component of the overall TANGO project, MOGO (a Mini-Ontology GeneratOr) develops and implements the necessary algorithms and user interfaces for automatically, semi-automatically, or manually generating mini-ontologies from canonicalized tables of data. (The first component of the TANGO project interprets raw tables found on the web and elsewhere and reorganizes them as canonical tables. The third component merges a set of

mini-ontologies into a large ontology representing a body of knowledge that is usable as a

means of organizing information on the Internet.)

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	3,559,547	45	-120
Washington	6,131,118	43	-120

Figure 1. Sample Table.

Given a table like the one in Figure 1, MOGO generates a conceptual model

(mini-ontology) that accurately represents the table of data by iterating through a set of

heuristics. Each heuristic deals with one of three main tasks: concept recognition,

relationship discovery, or 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 generates from the table in Figure 1. The four

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. For each location there are associated populations and geographic coordinates.
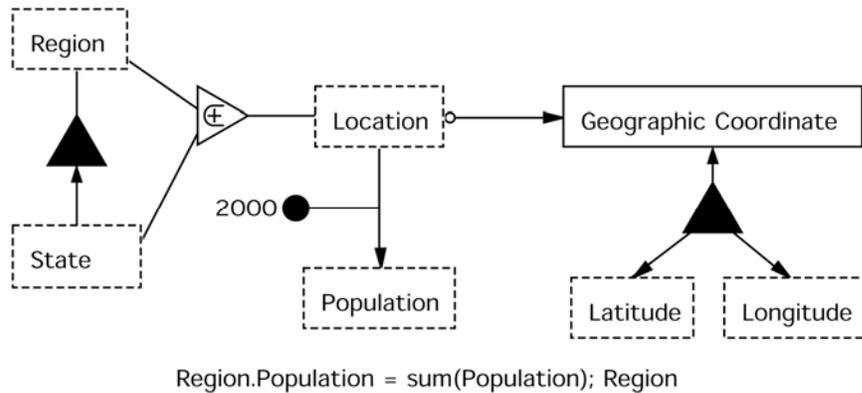
Figure 2. Sample mini-ontology, produced by MOGO for the table in Figure 1.

Our contribution is a tool, called MOGO, that accurately generates mini-ontologies from canonicalized tables of data automatically, semi-automatically, or manually. This tool is unique in that it combines both spatial and linguistic clues for generating the conceptual model, and it is easily extensible, allowing the addition of new algorithms at run time without the need for program recompilation.

The remainder of this thesis contains an overview of previous work and how it relates to this project. After that, we provide a detailed implementation description including an architectural overview, as well as detailed explanations of each of the heuristics MOGO uses to generate conceptual models. We conduct an evaluation study of MOGO on a set of randomly chosen tables and present precision and recall results in the areas of concept/value recognition, relationship discovery, and constraint discovery.

CHAPTER 2: RELATED WORK

Automating the creation of ontologies has become a widely researched area over the past few years, and researchers from many different backgrounds have contributed a variety of solutions. A common approach in the area of natural language processing (NLP) attempts to "learn" ontologies by finding the terms, concepts, relations, and concept hierarchies existing in large collections of unstructured text documents. 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 actually be used [5]. These approaches rely on a variety of methods to identify concepts in free form text documents including: word co-occurrence [9], formal concept analysis [23] for extracting concept hierarchies [6], and even fuzzy logic principles [20]. These methods often result in concept recognition but do little by way of understanding the relationships and constraints between these concepts. Our approach differs from typical NLP approaches by using tabular data as the source information. Using tabular data is useful in the creation of ontologies because the data has been structured by humans into a form representing the relationships found in the data. This structure makes the automatic discovery of relationship information much more effective than algorithms based solely on unstructured text documents.

Some researchers in the area of reverse engineering have worked on the problem of automatic generation of ontologies. Benslimane et al. [2] focus on generating OWL ontologies using HTML web forms in conjunction with the database schema associated with the forms. While this method shows promise, their approach differs from ours in that it relies on access to an underlying database schema and is based on web forms rather

than tables. Significant work has been done over the years on reverse engineering databases into conceptual models [1, 4, 7, 13, 15, 17, 19]. This work has focused on using an existing database schema and deriving the concepts, relationships, and constraints from the information implicit in the schema. While these projects have an output goal similar to MOGO (generating conceptual models), the input data is drastically different in that a database schema is a highly formalized structure which significantly increases the predictability of the data. In the case of MOGO, tables can have an arbitrary number of dimensions (unlike databases schemas which deal exclusively with tables having only column headers), and for MOGO there are no guarantees about the uniformity of table data values.

Pivk et al. [18] have approached automatic ontology creation in a manner similar to MOGO. Their approach (implemented as a system called TARTAR) uses tabular data as the input in the same way MOGO does, with the eventual output being an ontology representation using F-Logic frames. F-Logic frames have their roots in object-oriented program modeling and constitute a formal way to represent object identity, complex objects, inheritance, polymorphic types, query methods, and encapsulation [16]. TARTAR focuses primarily on using 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 to discover concepts, relationships, and constraints in a table.

CHAPTER 3: MINI-ONTOLOGY GENERATION

MOGO takes as input canonicalized tables of data based on Wang notation [22]. 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 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 information beyond 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. For practical reasons we also keep track of the original source URL of the document.

Based on the canonicalized input data, MOGO tries to produce a mini-ontology that conforms to the OSM data modeling language [12]. OSM provides a standard way of representing concepts, relationships, and constraints. 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 uses the following basic steps to automatically generate a mini-ontology:

1. *Concept/Value Recognition:* MOGO extracts the set of concepts found in each of the dimensions, and associates the table's data values with the appropriate concepts.
2. *Relationship Discovery:* MOGO adds relationship information to the concepts using structural and linguistic clues.
3. *Constraint Discovery:* MOGO adds constraint information to the mini-ontology by examining the table's data values.

MOGO performs all of these steps automatically and allows the user to: accept the mini-ontology without review, make adjustments to the mini-ontology, or manually rebuild the mini-ontology.

To illustrate how MOGO works, we use the table of geopolitical data in Figure 1 as an example. 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 the sample table in Figure 1 in canonicalized form in an XML document. The input XML must validate against an XML-Schema specification previously developed by others as part of the TANGO project. The *Table* tag contains a number of attributes useful to the overall TANGO project for uniquely identifying different tables. It also contains a title attribute which contains the table's title if there is one. Each element in the XML document has an object identifier (OID) for uniquely identifying the different nodes. *CategoryNodes* contain all of the labels found in the table. The *CategoryParentNodes* section captures the tree structure of the labels in each dimension. The *DataCells* section contains all of the data values in the table as well as references back to the labels that give the values a meaningful context. The final section, *Augmentations*, describes all of the augmentations found in the table which can include row, column, data, or table augmentations such as footnotes, values in labels (like the value 2000 in Figure 1), and units of measure. MOGO uses JAXB 2.0, an XML binding

framework available as part of Java 6, to read and validate canonicalized XML input and convert that input into simple Java objects.

Figure 4 shows a graphical representation of the canonicalized table in Figure 3. Each dimension of the table forms a tree structure with the depth of the tree determined by how many levels of label nesting exist in the dimension. The second dimension in the canonicalized table has no label value so a placeholder label of "[Dimension2]" is used. Each label in the dimension represents a node in the tree and connects to other tree nodes using a solid black line. Data values, at the bottom of the figure, connect to one node from each dimension using a dashed line. The dotted line connecting the "Population" node and the value "2000" indicates that the "2000" is an augmentation of the "Population" node. The title of the table is also captured and marked as such.

3.1 Auxiliary Services

Many of MOGO's algorithms rely on access to a base set of common services. These services provide access to basic lexical information and data frame classification operations.

3.1.1 Lexical Service

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 (in the form of a Java interface) to access lexical information. This Java interface defines what operations this service can perform, what parameters are required for each operation, and what information will be returned by each operation.

```
<InterpretedTable>
 <Table TableOID="Table1" Title="Region and State Information" Number="1" DocumentCitation="">
  <CategoryNodes>
   <CategoryNode CategoryNodeOID="C1" Label="Location" />
   <CategoryNode CategoryNodeOID="C1.1" Label="Northeast" />
   <CategoryNode CategoryNodeOID="C1.1.1" Label="" />
   <CategoryNode CategoryNodeOID="C1.1.2" Label="Delaware" />
   <CategoryNode CategoryNodeOID="C1.1.3" Label="Maine" />
   <CategoryNode CategoryNodeOID="C1.2" Label="Northwest" />
   <CategoryNode CategoryNodeOID="C1.2.1" Label="" />
   <CategoryNode CategoryNodeOID="C1.2.2" Label="Oregon" />
   <CategoryNode CategoryNodeOID="C1.2.3" Label="Washington" />
   <CategoryNode CategoryNodeOID="C2" Label="" />
   <CategoryNode CategoryNodeOID="C2.1" Label="Population" />
   <CategoryNode CategoryNodeOID="C2.2" Label="Latitude" />
   <CategoryNode CategoryNodeOID="C2.3" Label="Longitude" />
  </CategoryNodes>
 </Table>
 <CategoryParentNodes>
  <CategoryParentNode CategoryParentNodeOID="C1">
   <CategoryNodes>
    <CategoryNode CategoryNodeOID="C1.1" />
    <CategoryNode CategoryNodeOID="C1.2" />
   </CategoryNodes>
  </CategoryParentNode>
  <CategoryParentNode CategoryParentNodeOID="C2">
   <CategoryNodes>
    <CategoryNode CategoryNodeOID="C2.1" />
    <CategoryNode CategoryNodeOID="C2.2" />
    <CategoryNode CategoryNodeOID="C2.3" />
   </CategoryNodes>
  </CategoryParentNode>
  <CategoryParentNode CategoryParentNodeOID="C1.1">
    <CategoryNodes>
       <CategoryNode CategoryNodeOID="C1.1.1"/>
       <CategoryNode CategoryNodeOID="C1.1.2"/>
       <CategoryNode CategoryNodeOID="C1.1.3"/>
    </CategoryNodes>
  </CategoryParentNode>
   .  .  .
 </CategoryParentNodes>
 <DataCells>
  <DataCell DataCellOID="D1" DataValue="2,122,869">
   <CategoryLeafNode CategoryLeafNodeOID="C1.1.1" />
   <CategoryLeafNode CategoryLeafNodeOID="C2.1" />
  </DataCell>
  <DataCell DataCellOID="D2" DataValue="">
   <CategoryLeafNode CategoryLeafNodeOID="C1.1.1" />
   <CategoryLeafNode CategoryLeafNodeOID="C2.2" />
  </DataCell>
  <DataCell DataCellOID="D4" DataValue="817,376">
   <CategoryLeafNode CategoryLeafNodeOID="C1.1.2" />
   <CategoryLeafNode CategoryLeafNodeOID="C2.1" />
  </DataCell>
   .  .  .
 </DataCells>
 <Augmentations>
  <Augmentation Augmentation="2000" AugmentationType="value">
   <CategoryNodes>
    <CategoryNode CategoryNodeOID="C2.1" />
   </CategoryNodes>
  </Augmentation>
 </Augmentations>
</InterpretedTable>
```

Figure 3. XML version of canonicalized table.

Title: Region and State Information

Location   [Dimension2]

Northeast   Northwest   2000

Population   Latitude   Longitude

Delaware   Maine   Oregon   Washington

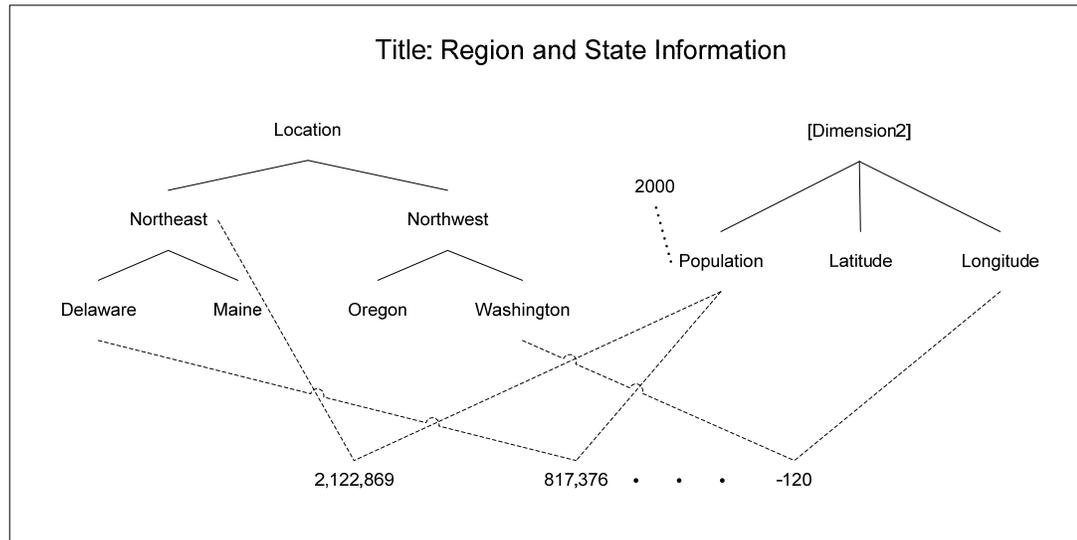2,122,869   817,376   ·   ·   ·   -120

Figure 4. Graphical view of canonicalized sample table.

Supported operations include term normalization, and testing whether one word is a hypernym, hyponym, meronym, or holonym of another word. Because all access to lexical information in MOGO is done through this interface, the user can modify, augment, or replace the underlying lexical service implementation without requiring source code changes to MOGO's various heuristic procedures.

In this implementation, MOGO uses WordNet, an electronic lexical database [14], for accessing lexical information. WordNet provides a number of freely available APIs enabling programmatic access to the underlying lexical repository. MOGO uses the Java API for WordNet Searching (JAWS) as its API for accessing WordNet resources. Because JAWS does not provide a mechanism for looking up a word's inherited hypernym list directly, MOGO's lexical service implementation builds these lists by looking up a word in WordNet and then recursively looking up each of the hypernyms of all senses of that word until a word with no hypernyms is reached. Similar operations are available for looking up a word's inherited hyponym and holonym lists.

To help increase the accuracy of lexical operations, MOGO's lexical service provides a term normalization routine. When doing term comparisons, it is important that the operations are performed using a unified lexicon so that matches can be correctly identified. MOGO normalizes all terms by looking terms up in WordNet and capturing all the associated word forms. Comparison operations involve looking for an exact match in at least one word form of a normalized term. For example, when the lexical service normalizes the term "Iowa", WordNet returns the word forms: "Iowa", "Ioway", "Hawkeye State", and "IA". For term comparisons, these different word forms are all treated as equivalent and an exact match with any one of them will return a valid match for the entire normalized term.

3.1.2 Data Frame Library Service

Another service that MOGO uses is the data frame library service. Data frames provide a mechanism for recognizing different types of objects from strings of data using regular expression recognizers [10]. 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 (and the associated ontology fragment) that matches that string. Each ontology fragment associated with a data frame contains one or more concepts, zero or more relationship sets, and zero or more constraints. In every case, one concept in the fragment is marked as the primary concept for the fragment. This primary concept serves as the connection point for MOGO's data frame related algorithms.

To illustrate how this works, suppose the string '12-08-2007' needs to be classified. MOGO's data frame service takes the string and loops through each of the data frame recognizers looking for a match. In this case the *Date* data frame recognizes

dates in the form MM-DD-YYYY and will successfully match the search string. The

data frame service returns the specific object set (concept) on which the search terms

match, as well as a reference to the entire ontology fragment associated with this data

frame.

3.1.3 Name Finding Service

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 which 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 a 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 the name finding service

does not find any matches to words in the pool then MOGO attempts to identify an

appropriate label by looking for the first common word in the inherited hypernym lists of

each of the concept's first ten data values. If a common word is found, MOGO assigns

that word as the concept's name.

3.2 Concept/Value Recognition

MOGO extracts concepts from a canonicalized table using a set of concept

recognition algorithms and assigns the appropriate data values to those concepts. Each

concept recognition algorithm conforms to a standard interface making it easy to augment

MOGO with additional heuristic algorithms. MOGO implements six concept recognition

algorithms. We execute each of the algorithms until each table label and table data value of the canonicalized table (Figure 2) is recognized as either a concept or a value for a concept. Each algorithm classifies the table labels and table data values it recognizes, and subsequent algorithms only evaluate unclassified labels and values until all labels and values have been classified, at which point MOGO skips any subsequent algorithms.

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 a 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 [12]. Object sets, either lexical or non-lexical, are the ontological elements representing the different concepts found in a table. A lexical object set is one whose members are printable and represent themselves (e.g., telephone numbers, names of companies). In OSM a lexical object set is visually represented by a box with a dashed border. A non-lexical object set's members are object identifiers that are non-printable (e.g., identifiers that stand for persons or companies). In OSM a non-lexical object set is visually represented by a box with a solid border.

The first concept recognition algorithm uses lexical clues to determine to which dimension labels the table's data values belong. MOGO uses its lexical service to compare each data value to its corresponding dimension labels. A data value is said to "belong" to a label if the data value is a hyponym of at least one of the label's 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 belong to the same dimension then it marks all of the labels in that dimension as

lexical object sets and associates the corresponding data values with those object sets.

Otherwise, MOGO clears the flags and proceeds to the next algorithm.

The first concept recognition algorithm fails to discover any concepts for the table

in Figure 1 because all 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.

This algorithm does succeed, however, for the sample table in Figure 5. Figure 6 shows

the object sets the algorithm creates out of each set of data values in the table. MOGO

examines each table value to see if that value belongs to the dimension label associated

with the value. In the case of the first column, the label "City" is found to be a hypernym

of the table values "Salt Lake City" and "Provo", so MOGO flags the label "City" as a

potential object set.   Similarly, the label "State" is found to be a hypernym of the value

"Utah", so MOGO flags "State" as a potential object set as well. Because all of the

flagged labels are from the same dimension, MOGO creates an object set for each of the

flagged labels and assigns the associated table values to the appropriate object set.

City	State
Salt Lake City	Utah
Los Angeles	California
San Francisco	California

Figure 5. Sample table where values "belong" to their labels.

City   State

---- Assigned Data Values ----

Salt Lake City   Utah
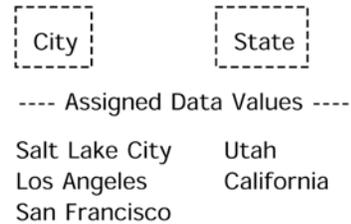Los Angeles   California
San Francisco

Figure 6. Object sets with values.

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

14

dimension has one label referred to as the root label. Below that, a dimension can contain several levels of label nesting. Beginning with the labels directly under the dimension's root label, MOGO uses its lexical service to look up each unmarked label in a dimension and retrieve that label's list of inherited hypernyms. A label is said to be an instance of its parent label if either the parent label, or the name of the object set the parent label is assigned to, is found 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 all the labels at that level as values, creates an unnamed lexical object set, and assigns the values to the object set. MOGO evaluates each succeeding level of label nesting in like manner until the leaf labels have been evaluated. MOGO uses the name finding service to find an appropriate name for any unnamed object sets produced by this algorithm. In cases where labels are found to belong to their parent label and the dimension only contains one level of label nesting, MOGO creates a single object set, names that object set using the dimension's root label, and assigns all of the labels of the dimension as values to that object set.

For the sample table in Figure 1, Figure 7 shows the object sets and associated data values the second algorithm creates for each level of label nesting in the "Location" dimension. The inherited hypernym list for each label in the "Location" dimension (Northeast, Northwest, Delaware, Maine, etc.) contains the word "Location." MOGO marks the labels at each level of nesting as values, creates unnamed lexical object sets for each level of nesting, and assigns the values from each level to the corresponding object set. The naming service extracts possible names for these object sets from word tokens in the title of the table. The inherited hypernym lists for each of the values assigned to the

region object set contain the word "Region" which is also a word that appears in the title of the table. 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. MOGO finds these matches and assigns the names "Region" and "State" to the two unnamed object sets.

```
┌ ─ ─ ─ ─ ─ ─ ─ ┐   ┌ ─ ─ ─ ─ ─ ─ ─ ┐
│ Region        │   │ State         │
└ ─ ─ ─ ─ ─ ─ ─ ┘   └ ─ ─ ─ ─ ─ ─ ─ ┘

---- Assigned Data Values ----

Northeast          Delaware
Northwest          Maine
                   Oregon
                   Washington
```
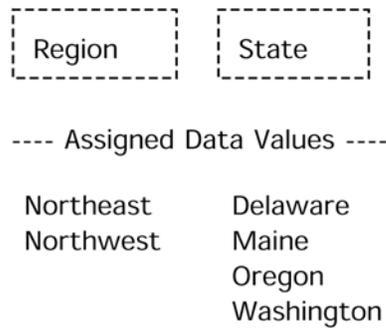
Figure 7. Object sets that the second algorithm creates.

The third algorithm checks for labels at the same level of nesting that have the exact same name. Tables often contain multiple columns with the same type of information. This is usually manifest in tables that have labels that span multiple columns or rows and usually only appears in tables with more than one level of label nesting. Beginning with the labels directly under the dimension's root label, MOGO compares each unmarked label with the other labels at that same level to see if all of the labels are exactly the same. 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 of the values associated with those labels to the newly created object set. MOGO evaluates each succeeding level of label nesting in like manner until the leaf labels have been evaluated.

The labels in Figure 1 are all different, so this third algorithm does not apply to the table in Figure 1. For the sample table in Figure 8, Figure 10 shows the object sets

the third algorithm creates. While the label "Number of Deaths" does not appear twice in the source table, it does appear twice in the canonicalized version of the table. Figure 9 shows how the canonicalization process duplicates the single source label "Number of Deaths". This replication is an artifact of the canonicalization process encountering labels that span multiple columns. In this case, MOGO recognizes the label duplication, merges the duplicate labels into one object set, assigns the common label as the name of the object set, and assigns any values associated with the labels to the newly created object set.

	2002	2003
Province	Number of Deaths	
Quebec	54,896	56,411
Ontario	83,410	84,155

Figure 8. Sample table with label column span.
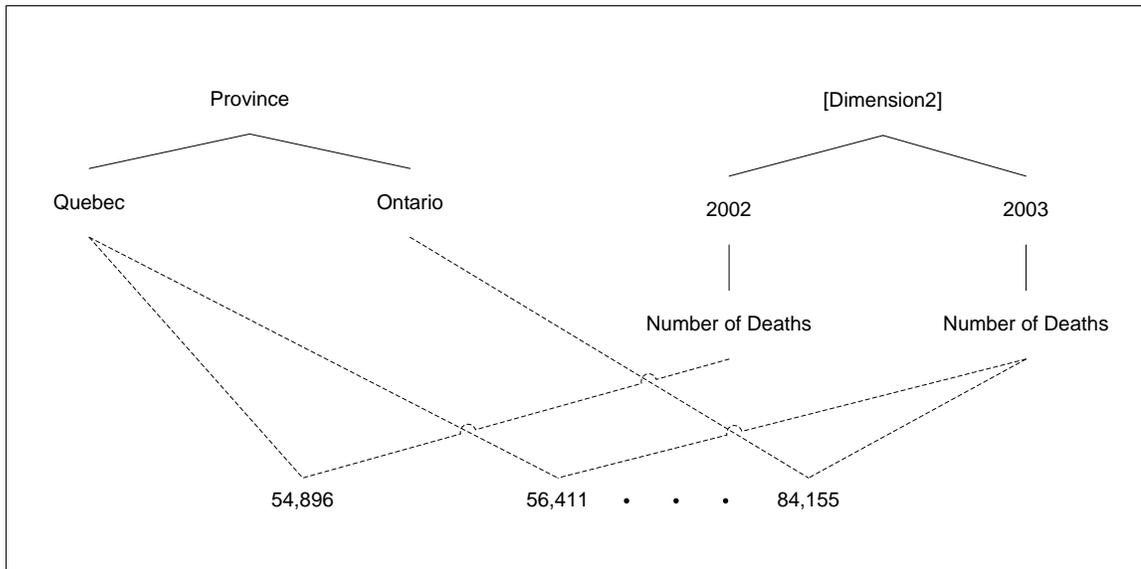
Figure 9. Canonicalized version of sample table in Figure 8.

17

```
+--------------------+
| Number of Deaths   |
+--------------------+

--- Assigned Values ---

        54,896
        83,410
        56,411
        84,155
```
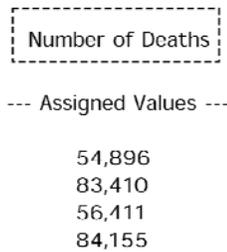
Figure 10. Object sets that algorithm three creates.

The fourth concept recognition algorithm takes each unmarked dimension label and attempts to classify all the data values associated in a row or column with that label using MOGO's data frame service to determine to which dimension labels the table's data values belong. If all the data values in a row or column have the same type, MOGO temporarily associates that type with the dimension label. After MOGO classifies all the labels for a dimension using its data frame service, if there are at least two labels in the dimension of different types, MOGO flags all of the labels in the dimension as lexical object sets and associates the corresponding data values with the object sets. Requiring two different types avoids 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.

Using the sample table in Figure 11 as the source table, Figure 12 shows the object sets the fourth algorithm creates. Using the data frame library service, MOGO classifies each of the data values found in the source table. For the sample table in Figure 11, the values "Tire", "Transmission", and "Steering Wheel" all match the car part data frame. The values "$115.60", "$356.45", and "$32.34" all match the currency data frame. Because the values in two different columns match two separate data frames, MOGO creates object sets for each of the columns, uses the label associated with that column to name the object set, and assigns the values associated with those columns to the newly created object sets.

18

Car Part	Price
Tire	$115.60
Transmission	$356.45
Steering Wheel	$32.34

Figure 11. Sample table of car parts.

Figure 12. Object sets the fourth algorithm creates.

The fifth concept recognition algorithm tries to identify concepts among sibling labels. MOGO first classifies each unmarked dimension label using its data frame service. For each set of sibling labels that have the same data frame classification, MOGO marks the labels as values, creates an object set, names the object set with the same name as the matching concept returned by the data frame service, and associates the sibling labels with the new object set.

For the sample table in Figure 8, MOGO classifies the labels "2002" and "2003" as instances of a "Year" object set using the data frame library service. Because both sibling labels are classified as the same type, MOGO creates an object set, uses the name of the matching concept to name the object set "Year", and assigns the labels "2002" and "2003" as data values to the newly created object set.

If the prior algorithms do not successfully mark all items in the canonicalized table as object sets or values, MOGO processes the remaining unmarked items based on whether or not the dimension's root has a real label or a placeholder label. For dimensions whose root nodes contain a placeholder labels, MOGO flattens any label nesting in the dimensions by prepending parent labels to child labels, removes parent labels until there is no more nesting, and marks all of the unmarked labels as lexical object sets. If the data values associated with those labels are currently unassigned,

19

MOGO assigns the data values to the newly created object sets. For any unmarked 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

each group of labels, associates the values with the newly created object sets, and uses

the name finding service to find appropriate names for the object sets. For any remaining

data values that are not currently assigned to an object set, MOGO creates a new

unnamed object set and assigns the values to that object set.

Using the sample table in Figure 1 as the source table, Figure 13 shows the results

of running the final algorithm. The algorithm creates lexical object sets for each of the

labels in the "[Dimension2]" dimension because the dimension's root node contains a

placeholder label and none of its labels are marked as either an object set or a data value

by any of the previous algorithms. MOGO also assigns the associated data values, none

of which are assigned to an object set by previous algorithms, to the newly created object

sets.

```
  ┌─────────────┐    ┌─────────────┐    ┌─────────────┐
  ¦ Population  ¦    ¦ Latitude    ¦    ¦ Longitude   ¦
  └─────────────┘    └─────────────┘    └─────────────┘

  ──────────────────── Assigned Data Values ────────────

    2,122,869            45                 -90
    817,376             44                 -93
    1,305,493            45                 -120
    9,690,665            43                 -120
    3,559,547
    6,131,118
```

Figure 13. Object sets created by final algorithm.

Figure 14 shows all the object sets MOGO identifies in our sample table from

Figure 1 after all of the concept/value recognition algorithms have completed.

Region	State	Population	Latitude	Longitude

---------------------------------- Assigned Data Values ----------------------------------

Northeast	Delaware	2,122,869	45	-90
Northwest	Maine	817,376	44	-93
	Oregon	1,305,493	45	-120
	Washington	9,690,665	43	-120
		3,559,547		
		6,131,118		

Figure 14. Discovered object sets and value assignments.

3.3 Relationship Discovery

With all of the concepts identified and the values assigned to those concepts, MOGO next identifies all of the relationships that exist between the different concepts. MOGO adds relationship information to the object sets using a set of relationship discovery algorithms. Each relationship discovery algorithm conforms to a standard interface making it easy to augment MOGO with additional heuristic algorithms. MOGO implements five relationship discovery algorithms. We execute the algorithms in order, passing the newly discovered relationship information on to the next algorithm until each of the algorithms has successfully run. Unlike the concept recognition algorithms which only run until all labels and values have been classified, the full set of relationship recognition algorithms runs — each successively refines the results of the previous.

The first relationship discovery algorithm extracts relationship information from the dimension trees. For each dimension, MOGO creates relationship sets between the object sets from that dimension anywhere an edge exists in the dimension trees. 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 in the same level of label nesting) do

21

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 15 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 4. 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. Object sets of unknown type are visually

represented as a shaded box with no border.
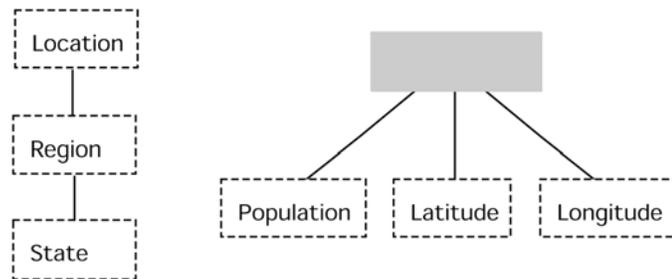

Figure 15. Relationship sets from dimension trees.

The second relationship discovery algorithm modifies the generated ontology

relationship sets using lexical clues. MOGO's lexical service provides a way to analyze

object set labels and sets of values associated with object sets to discover semantic

relationship information like hypernyms, hyponyms, holonyms, and meronyms.

Hypernyms and hyponyms translate to generalization/specialization relationships (represented as an empty triangle). Holonyms and meronyms translate to aggregation relationships (represented as a filled-in triangle). MOGO looks for more specific relationship information by examining each object set involved in a relationship set to see if the labels or values in the two object sets contain any of these semantic relationships. If they do, MOGO adjusts the relationship set by replacing it with an aggregation or generalization/specialization.

If aggregations are found between the different object sets from one dimension, MOGO looks for any generalization/specializations that might exist in the table. Using its lexical service, MOGO looks up the inherited hypernym list of each object set label participating in the aggregation. If the dimension's root label is in the inherited hypernym lists of all the different object sets, 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.

Figure 16 shows the sample table's ontology elements in Figure 15 after MOGO modifies them using lexical clues. Using its lexical service, MOGO finds that "Delaware" is an instance of an "American State" which is a hyponym of "region." MOGO uses this information to create 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 this object set with the existing object sets as

specializations.

Figure 16. Relationship sets after linguistic processing.

The third relationship discovery algorithm uses MOGO's data frame service to

find relationships between the object sets. MOGO first attempts to recognize each object

set label using the data frame service and stores any matches found. When all of the

object sets have been classified, MOGO searches the list of matches looking for object

sets that match different concepts in the same data frame ontology fragment and merges

these matches. For each data frame match, MOGO adds the ontology fragment

associated with the data frame to the mini-ontology, removes all of the previous object

sets represented by the new ontology fragment, and updates any relationship sets

associated with the removed object sets to point at the primary object set found in the

data frame ontology fragment. In cases where all of the labels at a given level of label

nesting are classified as the same data frame type, MOGO adds the ontology fragment

associated with the data frame to the mini-ontology, removes all of the previous object

sets represented by the new ontology fragment and assigns their labels as data values to

the appropriate object set in the ontology fragment, and updates any relationship sets

associated with the removed object sets to point at the object set generated from the

parent label.

24

Figure 17 shows the results of MOGO's applying this algorithm. MOGO finds

data frame matches on the "Latitude" and "Longitude" object sets. These object sets

match different object sets in the same data frame so MOGO merges the two matches

into one. The matching data frame contains information about geographical coordinate

objects. MOGO adds the ontology fragment for this data frame to the mini-ontology,

removes the previous "Latitude" and "Longitude" object sets, and transfers any

relationship sets previously connected to the "Latitude" and "Longitude" object sets to

the primary object set of the data frame ontology fragment which in this case is the
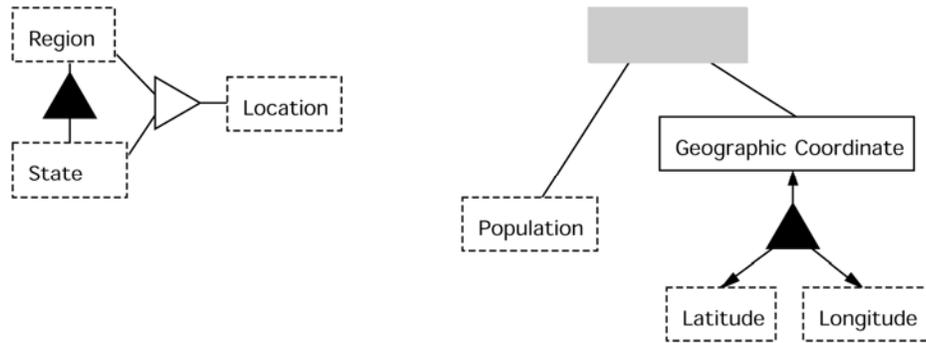
"Geographic Coordinate" object set.

Figure 17. Relationship sets after data frame recognizers.

For the sample canonicalized table in Figure 18, Figure 19 shows the relationship

sets that the third relationship discovery algorithm finds. MOGO's data frame service

recognizes the "2005" and "2006" labels as values in a "Year" object set. Because the

"2005" and "2006" labels represent all of the labels at that level of label nesting, MOGO

adds the "Year" object set associated with the matched data frame to the mini-ontology,

removes any previous object sets associated with the "2005" and "2006" labels, assigns

these labels as data values to the "Year" object set, assigns the values previously

associated with those labels to the "Passengers" object set which was generated from these labels' parent label, and creates a ternary relationship set among the "Airport", "Year", and "Passengers" object sets.
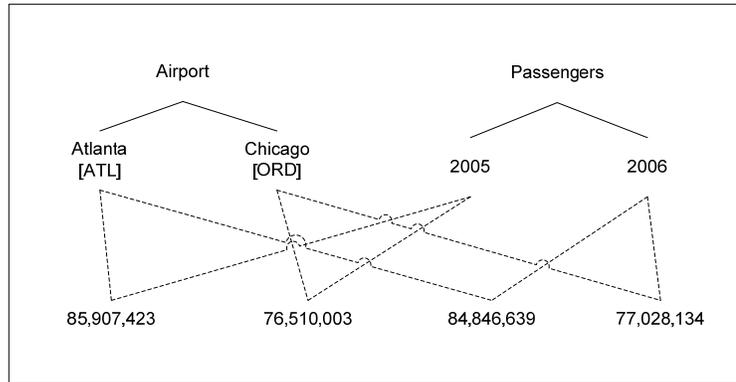
Figure18. Sample canonicalized table with nested year labels.
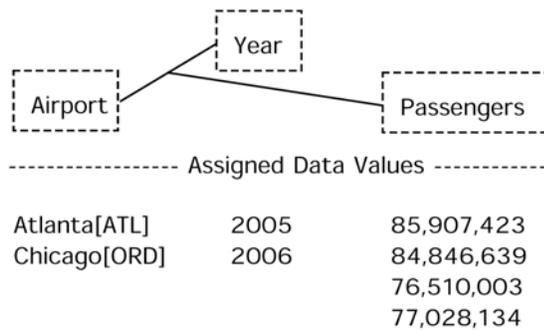
Figure19. Relationship sets for nested label table after data frame recognizers.

The fourth relationship discovery algorithm processes any augmentations that exist in the canonicalized table. For each row and column augmentation that is a value and not a unit, footnote, or a parenthetical remark as indicated by the canonicalized table, MOGO creates a singleton object with the value found in the augmentation, and forms an n-ary relationship set among the singleton object and the object sets associated with the augmentation.

26

Figure 20 shows the results of MOGO's applying this algorithm. As Figures 1, 3, and 4 show, the "Population" column has the augmentation "2000". MOGO creates a singleton object of value 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.
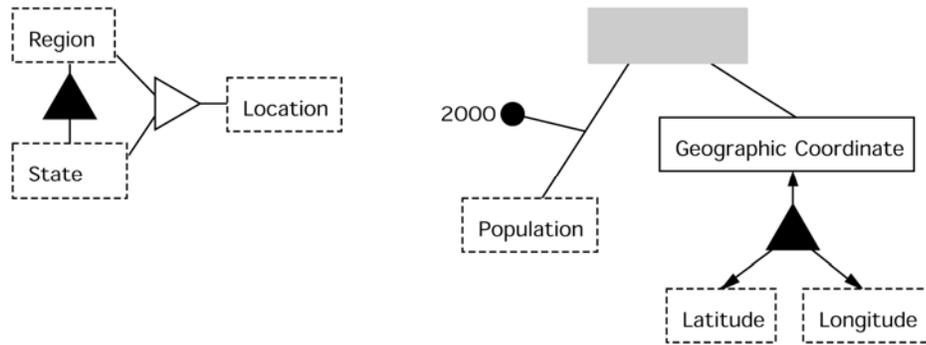
Figure 20. Relationship sets after processing augmentations.

The final relationship discovery algorithm merges ontology fragments into a mini-ontology. Ontology fragments are made up of all of the ontology elements that are inter-connected via some type of relationship set. MOGO joins the ontology fragments by creating an *n*-ary relationship set among the ontology fragment link-in points. An ontology fragment's link-in point is the object set in the fragment that came from the highest level label or labels 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 removes the placeholder object set and the *n*-ary relationship set, and transfers all of the removed object set's relationships to the remaining ontology fragment's link-in point.

In our sample table, there are two ontology fragments. Figure 21 shows the result of merging the two sample ontology fragments into a mini-ontology. MOGO removes the placeholder object set from the one ontology fragment because there is only one other ontology fragment. MOGO then assigns the orphaned relations to the link-in point of the other ontology fragment (the "Location" object set).
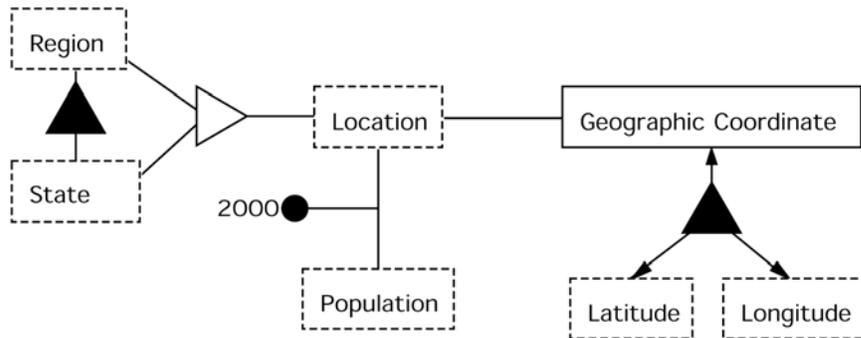
Figure 21. Mini-ontology results from fragment merge.

3.4 Constraint Discovery

MOGO adds constraints to the mini-ontology using a set of constraint discovery algorithms. Each constraint discovery algorithm conforms to a standard interface making it easy to augment MOGO with additional heuristic algorithms. MOGO implements four constraint discovery algorithms. We execute each algorithm until each has run successfully. Each checks for a single kind of constraint; if an algorithm finds that the constraint it is checking holds, it adds the constraint to the mini-ontology being created.

The first constraint discovery algorithm adds constraints to generalization/specialization relationships that exist in the mini-ontology. A generalization/specialization relationship can be constrained to be a union, a mutual exclusion, or a partition. MOGO constrains a generalization/specialization relationship to

28

be a union (represented as a triangle containing a ∪) if all values in the generalization object set are also in at least one of the specialization object sets. MOGO adds a mutual exclusion constraint (represented as a triangle containing a +) if there is no overlap in the values in each of the specialization object sets. When the generalization/specialization is constrained by both union and mutual exclusion, MOGO assigns a partition constraint (represented as a triangle containing both a ∪ and a +) to the relationship.

Figure 22 shows the results of running this algorithm on our sample table. MOGO determines that there are no values assigned to the "Location" object set that are not also assigned to the "Region" or "State" object sets. The values in the "Region" and "State" object sets are also found to be mutually exclusive. Thus, MOGO assigns a partition constraint to the generalization/specialization relationship in the mini-ontology.
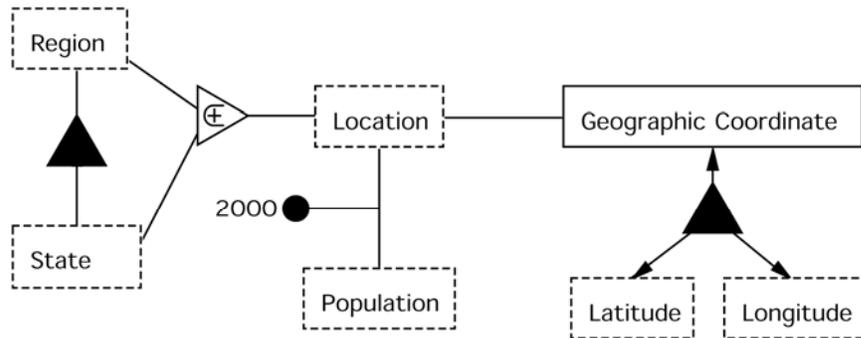
Figure 22. Mini-ontology with generalization/specialization constraints.

The second constraint discovery algorithm looks for any computed values in the table. Tables often include columns or rows that contain the summation, average, or other aggregates of values in the table. MOGO examines the values related to object sets that come from dimensions with label nesting. By computing aggregates of the values from related object sets and comparing them to given values to test whether the

29

aggregates hold, MOGO captures these constraints and adds them as annotations to the

mini-ontology.

Figure 23 shows the results of running this algorithm on our sample table.

Looking for possible aggregate values, MOGO determines that the population values

related to the "Region" object set values are the summation of the population values

related to the "State" object set. MOGO thus adds the constraint "Region.Population =

sum(Population); Region" to the mini-ontology. (The notation here means that a region's

population is the sum of the population values grouped by Region; it is adapted from [8],

which defines computational expressions over ER conceptual models).
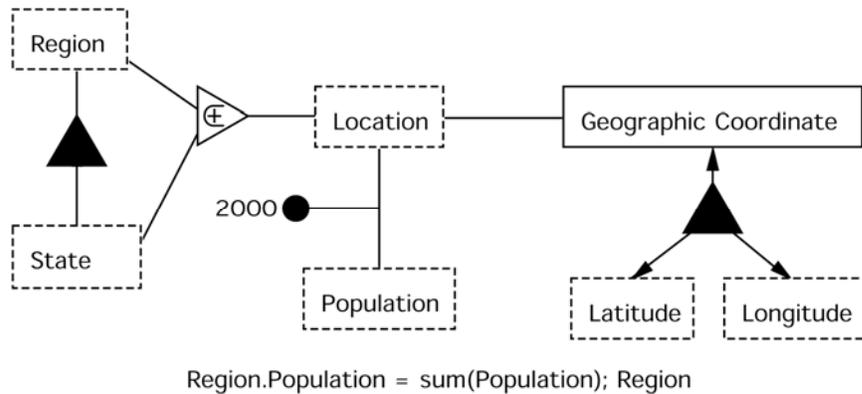
Figure 23. Mini-ontology with computed value constraint.

The third constraint discovery algorithm looks for functional relationship sets.

Each of the data values in a table is functionally determined by the set of dimension

labels associated with those values. MOGO identifies the object sets that contain the

table's data values and marks the relationship sets coming into those object sets as

functional. Object sets assigned values that are dimension labels are handled separately.

MOGO 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. 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.

Figure 24 shows the results of running this algorithm on our sample table. The "Population", "Latitude", and "Longitude" object sets contain the data values from the canonicalized tables, so MOGO marks the relationship sets coming into these object sets as functional. Because the "Latitude" and "Longitude" object sets were replaced by an ontology fragment associated with a data frame, MOGO marks the relationship sets coming into the "Geographic Coordinate" object set (the ontology fragment's primary object set) as functional. The "Region" and "State" object sets contain values from dimension labels. Because the values assigned to the "State" object set appear to functionally determine the values assigned to the "Region" object set (i.e. there is only one region for each state), MOGO marks the relationship set from the "State" object set to the aggregation connecting it to the "Region" object set as functional.
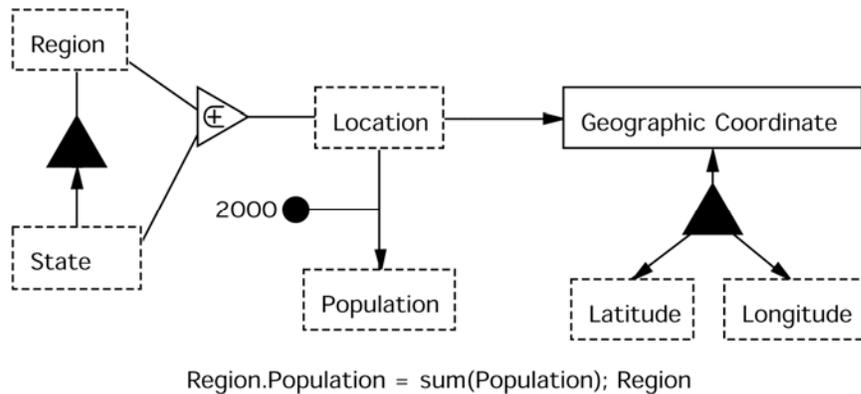
Figure 24. Mini-ontology with functional constraints.

The final constraint discovery algorithm determines if objects in an object set participate mandatorily or optionally in associated relationship sets. Optional participation is represented in OSM as an o 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 mapped in the mini-ontology

and marks participation in any relationship sets between one of these object sets and any

other object set as optional.

Figure 25 shows the results of running this algorithm on our sample table. The

canonicalized table contains four empty data cells. These non-existing values "belong"

to the "Longitude" and "Latitude" object sets. MOGO thus marks participation of object

sets in any relationship sets coming into either of these object sets as optional. Because

these object sets were replaced by an ontology fragment associated with a data frame,

MOGO marks the connection between the "Location" object set and the relationship set

that comes into the data frame's primary object set, the "Geographical Coordinate" object

set, as optional.
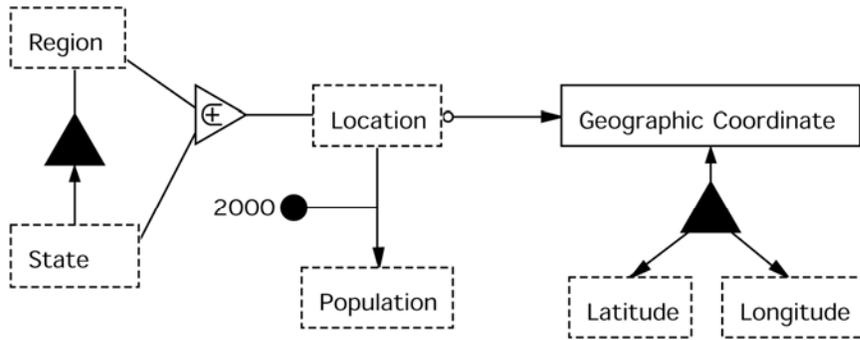
Figure 25. Final mini-ontology MOGO produces.

CHAPTER 4: EXPERIMENTAL RESULTS

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 that all the tables be from the geopolitical domain.

For each test URL gathered by the participant, we saved a local copy of the page's source HTML and used the tools created in the first component of the TANGO [21] project to canonicalize the tables. MOGO processed each of the twenty canonicalized tables and the resulting mini-ontologies were formatted and saved for evaluation. We evaluate each mini-ontology in three different areas: concept/value recognition, relationship discovery, and constraint discovery.

It is necessary to point out that when building ontologies, there is often no "right" answer. For any given set of data there can be multiple ontologies that are valid conceptualizations of the data set. For that reason, it is necessary for the evaluation to be done manually by a trained expert in the field of data conceptualization.

4.1 Concept/Value Recognition

Every table has a fixed number of concepts, concept labels, and data values. We observe how many concepts, concept labels, and data values MOGO correctly identifies, how many it misses, and how many it proposes that are invalid. We compute precision

values with respect to concept/value recognition by dividing the total number of correct

concepts, labels, and data values MOGO finds by the total number of actual concepts,

labels, and data values combined with the incorrect concepts, labels, and data values

MOGO proposes. We compute recall values with respect to concept/value recognition by

dividing the total number of correct concepts, labels, and data values MOGO finds by the

total number of actual concepts, labels, and data values found in the canonicalized table.

4.2 Relationship Discovery

We evaluate relationship discovery by observing how many valid relationship

sets, aggregations, and generalization/specializations MOGO discovers, how many it

proposes that are invalid, and how many MOGO does not discover. In cases where a

relationship set, aggregation, or generalization/specialization should exist but does not

because MOGO did not correctly identify a concept, we count the missing relationship

set, aggregation, or generalization/specialization as one that MOGO did not discover.

We compute precision values with respect to 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 incorrect relationship sets, aggregations,

and generalization/specializations MOGO proposes. We compute recall values with

respect to 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 found in the

canonicalized table.

4.3 Constraint Discovery

We evaluate constraint discovery by observing how many valid constraints MOGO discovers, how many invalid constraints it proposes, and how many valid constraints MOGO does not discover. Observations are made for each of the following types of constraints: functional dependencies, generalization/specialization constraints, computed values, and optional participation of objects in object sets and their associated relationship sets. In cases where constraints should exist but do not because MOGO did not correctly identify a concept or relationship, we count the missing constraint as one that MOGO did not discover. We compute precision values with respect to constraint discovery by dividing the total number of correct constraints MOGO finds by the total number of actual constraints combined with the incorrect constraints MOGO proposes. We compute recall values with respect to constraint discovery by dividing the total number of constraints MOGO finds by the total number of actual constraints found in the canonicalized table.

4.4 Results

Appendix A contains all of the original HTML tables and the mini-ontologies MOGO generated as part of the evaluation. We report the accuracy of MOGO with respect to precision and recall values. Table 1 shows the precision and recall values for each area of evaluation. MOGO achieves a precision of 87% and recall of 94% for the concept recognition task, a precision of 73% and recall of 81% for the relationship discovery task, and a precision of 89% and recall of 91% 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%.

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

Table 1. Precision and recall values for evaluation tables.

Unfortunately, a direct comparison of MOGO's results with results achieved by TARTAR [18], a similar system for converting tables to conceptual models, is not possible. TARTAR's results take into account both the table canonicalization process and the conversion to a conceptual model. MOGO's results are based on a set of canonicalized tables that were checked to be accurately canonicalized before being processed by MOGO. So while at first glance it might appear that MOGO performs significantly better than TARTAR, because the results measure different objects/targets, it is very difficult to compare the two systems in a meaningful way.

4.5 Issues

One issue that MOGO encounters in concept recognition is generating duplicate concepts. In our set of evaluation tables, a number of tables had multiple columns that corresponded to the same concept. The only difference in the columns was in units of the data values. For example, a table about mountain peaks contains two columns labeled *height* but in one column the height is given in meters and in the other column the height appears in feet. MOGO was unable to correctly merge these concepts into one. Further enhancements to MOGO's use of data frames would likely yield the desired results in these types of cases.

The other concept recognition area MOGO struggled with is in identifying a valid label for a concept. There are a number of reasons why this occurs. Sometimes a valid label for the concept does not even exist in the table. Many tables assume that the reader can infer the correct label based on the context in which the table occurs. Unfortunately, this contextual information is not available in the canonicalized table which MOGO uses as input. 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, MOGO cannot identify a label for the concept that contains these values.

In the relationship discovery task, MOGO occasionally struggles with identifying aggregations and generalization/specializations. The main case for which MOGO was not able to identify aggregations and generalization/specializations is when these types of relationships exist between two sibling labels. MOGO only looks for these types of relationships when there is label nesting present in the dimension. In cases where sibling labels form an aggregation, such as a table with a column full of city names and another column full of state names, MOGO's heuristics do not cover checking for aggregations or generalization/specializations.

The other main area that MOGO struggles with in relationship discovery is assigning relationship sets to invalid concepts. Errors in earlier phases of mini-ontology generation have a cascading effect on errors in later parts of the process. Invalid concepts found in the concept recognition phase invariably lead to invalid relationship sets in the relationship discovery phase.

In the constraint discovery task, MOGO occasionally misclassifies a nonfunctional relationship set as a functional one. The most common cause of this was when the canonicalized table values contained lists of items instead of a single value. MOGO treats all table values as singleton objects and uniformly constrains relationship sets with the object sets that contain these values as functional. In cases where table values contain lists of objects, this behavior is incorrect.

The final area in the constraint discovery task that MOGO struggles with is in tables that contain arbitrary rows and columns that contain totals. 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. Only when the row or column represents a conceptual aggregation of the other values, such as a state population containing the computed sum of the populations of the cities in that state, does MOGO correctly identify the computed value.

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

We have created a system called MOGO that automates the generation of mini-ontologies from canonicalized tables of data. MOGO uses a novel approach to ontology generation by combining both spatial and linguistic clues for generating conceptual models, and it is easily extensible allowing the addition of new algorithms at run time without the need for program recompilation.

Experimental results show that MOGO is able to automatically identify the concepts, relationships, and constraints that exist in arbitrary tables of values 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. This automation can significantly reduce the work required to generate ontologies from canonicalized tables.

5.1 Future Work

The base set of algorithms MOGO uses to generate mini-ontologies cover many of the common patterns found in tables but they do not constitute an exhaustive set of algorithms for table conversion. The following possibilities for future work include both algorithm refinements as well as other possible applications for MOGO.

5.1.1 Linguistic Processing

Many of the algorithms in MOGO take advantage of external lexical resources using MOGO's lexical service. Our current implementation of this lexical service only uses WordNet [14] for querying lexical information. Future work could be done not only

in how WordNet is used by the lexical service, but also in the incorporation of other external linguistic resources.

5.1.2 Data Frame Library

MOGO makes frequent use of a data frame library for recognizing complex data types using its data frame library service. The idea of data frames has been well thought out and successfully implemented [10] but the set of data frame recognizers in the library is still somewhat limited. Increasing the coverage of the set of data frame recognizers in the data frame library could significantly increase MOGO's effectiveness.

5.1.3 Domain Specific Algorithms

All the algorithms MOGO uses to generate mini-ontologies are designed to be very general-purpose algorithms. It is very likely that tables from specific domains would benefit from algorithms written specifically for that domain. Such algorithms might be able to recognize common abbreviations, formats, or terms specific to a particular domain. These algorithms could be written to run in addition to or in place of MOGO's algorithms.

5.1.4 MOGO as Part of a Semantic Web Annotation System

MOGO's primary goal is to function as the second component of the larger TANGO [21] project. The TANGO project focuses on automating the process of creating an ontology from the concepts, relationships, and constraints found in sets of tabular data. Another possible application for MOGO would be to use the resultant mini-ontologies as extraction ontologies [11]. Extraction ontologies are useful for annotating source tables with ontology information thereby making those tables accessible as part of the semantic web [3].

BIBLIOGRAPHY

1. Alhajj, R., Extracting the Extended Entity-Relationship Model from a Legacy Relational Database, *Information Systems* **28** (6), 2003, 597-618.
2. Benslimane, S.M., Benslimane, D., and Malki, M., Acquiring OWL Ontologies from Data-intensive Web Sites, *Proceedings of the 6th International Conference on Web Engineering*, Palo Alto, California, July 2006, 361-368.
3. Berners-Lee, T., Hendler, J., and Lassila, O., The Semantic Web, *Scientific American* **5** (284), 2001, 34-43.
4. Chiang, R.H.L., Barron, T.M., and Storey, V.C., Reverse Engineering of Relational Databases: Extraction of an EER Model from a Relational Database, *Data & Knowledge Engineering* **12** (2), 1994, 107-142.
5. Cimiano, P., *Ontology Learning and Population from Text: Algorithm, Evaluation and Applications*, Springer, New York, New York, 2006.
6. Cimiano, P., Hotho, A., and Staab, S., Learning Concept Hierarchies from Text Corpora Using Formal Concept Analysis, *Journal of Artificial Intelligence Research* **24** 2005, 305-339.
7. Comyn-Wattiau, I., and Akoka, J., Reverse Engineering of Relational Database Physical Schema, *Proceedings of the 15th International Conference on Conceptual Modeling*, Cottbus, Germany, October 7-10, 1996, 372-391.
8. Czejdo, B., and Embley, D.W., An Approach to Computation Specification for an Entity-Relationship Query Language, *Proceedings of the 6th Entity-Relationship Conference*, New York, New York, November 1987, 307-321.
9. Dittenbach, M., Berger, H., and Merkl, D., Improving Domain Ontologies by Mining Semantics from Text, *Proceedings of the First Asian-Pacific Conference on Conceptual Modelling*, Dunedin, New Zealand, Jan, 2004, 91-100.
10. Embley, D., Programming with Data Frames for Everyday Data Items, *National Computer Conference*, Anaheim, California, May, 1980, 301-305.
11. Embley, D.W., Campbell, D.M., Jiang, Y.S., Liddle, S.W., Lonsdale, D.W., Y.-K., N., and Smith, R.D., Conceptual-Model-Based Data Extraction from Multiple-Record Web Pages, *Data & Knowledge Engineering* **31** (3), 1999, 227-251.
12. Embley, D.W., Kurtz, B.D., and Woodfield, S.N., *Object-oriented Systems Analysis: A Model-driven Approach*, Prentice-Hall, 1992.
13. Embley, D.W., and Xu, M., Relational Database Reverse Engineering: A Model-Centric, Transformational, Interactive Approach Formalized in Model Theory, *DEXA'97 Workshop Proceedings*, Toulouse, France, September 1-5, 1997, 372-377.
14. Fellbaum, C., *WordNet: An Electronic Lexical Database*, MIT Press, 1998.
15. Johannesson, P., A Method for Transforming Relational Schemas Into Conceptual Schemas, *Proceedings of the Tenth International Conference on Data Engineering*, Houston, Texas, February 14-18, 1994, 190-201.
16. Kifer, M., Lausen, G., and Wu, J., Logical Foundations of Object-oriented and Frame-based Languages, *Journal of the Association for Computing Machinery* **42** (4), 1995, 741-843.

17. Lammari, N., Comyn-Wattiau, I., and Akoka, J., Extracting Generalization Hierarchies from Relational Databases: A Reverse Engineering Approach, *Data & Knowledge Engineering* **63** (2), 2007, 568-589.

18. Pivk, A., Sure, Y., Cimiano, P., Gams, M., Rajkovic, V., and Studer, R., Transforming Arbitrary Tables into Logical Form with TARTAR, *Data & Knowledge Engineering* **60** 2007, 567-595.

19. Premerlani, W.J., and Blaha, M.R., An Approach for Reverse Engineering of Relational Databases, *Communications of the ACM* **37** (5), 1994, 42-49.

20. Raymond, Y.K.L., Yuefeng, L., and Yue, X., Mining Fuzzy Domain Ontology from Textual Databases, *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*, Fremont, California, November 2007, 156-162.

21. Tijerino, Y.A., Embley, D.W., Lonsdale, D.W., Ding, Y., and Nagy, G., Toward Ontology Generation from Tables, *World Wide Web: Internet and Web Information Systems* **8** (3), September 2004, 251-285.

22. Wang, X., *Tabular Abstraction, Editing and Formatting*, PhD dissertation, Department of Computer Science, University of Waterloo, 1996.

23. Wille, R., Formal Concept Analysis as Mathematical Theory of Concepts and Concept Hierarchies, *Formal Concept Analysis, Foundations and Applications*, LNCS 3626, Springer, 2005, 1-33.

APPENDIX A – EVALUATION TABLES

This appendix includes images of all the original HTML evaluation tables along with the mini-ontologies generated by MOGO for each table. Object sets with labels beginning with "osmx" followed by a number are equivalent to unlabeled object sets.

⋈	River ⋈	Length (km) ⋈	Length (miles) ⋈	Drainage area (km²) ⋈	Average discharge (m³/s) ⋈	Outflow ⋈	Countries in the drainage basin ⋈
1.**	Nile	6,650	4,135	3,349,000	5,100	Mediterranean Sea	Ethiopia, Eritrea, Sudan, Uganda, Tanzania, Kenya, Rwanda, Burundi, Egypt, Democratic Republic of the Congo
2.**	Amazon	6,400	3,980	6,915,000	219,000	Atlantic Ocean	Brazil, Peru, Bolivia, Colombia, Ecuador, Venezuela, Guyana
3.	Yangtze (Chang Jiang)	6,300	3,917	1,800,000	31,900	East China Sea	P.R. China
4.	Mississippi - Missouri	6,275	3,902	2,980,000	16,200	Gulf of Mexico	United States (98.5%), Canada (1.5%)
5.	Yenisei - Angara - Selenga	5,539	3,445	2,580,000	19,600	Kara Sea	Russia, Mongolia
6.	Yellow (Huang He)	5,464	3,398	745,000	2,110	Bohai Sea (Balhae)	P.R. China

43
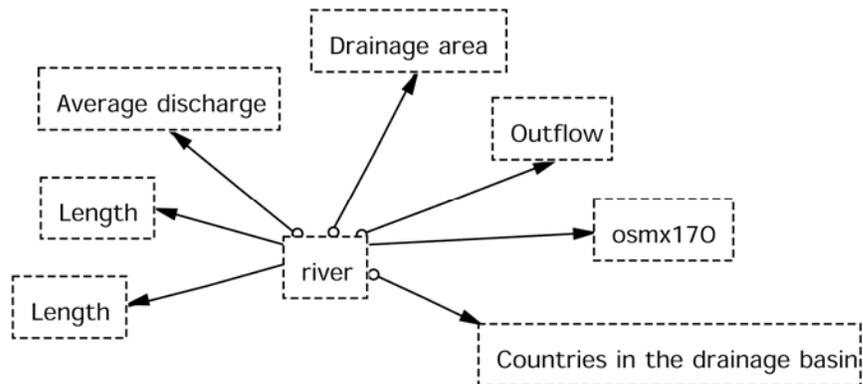
Rank	Mountain	Height (m)[1]	Height (ft)	Range	Region[2]	Coordinates[3]	Prominence (m)[4]	Parent mountain[5]	First ascent	Ascents[6] (attempts)
1	Mount Everest/ SagarMatha / Chomolungma	8,848[7]	29,028	Mahalangur Himalaya	Nepal / Tibet	27°59′17″N 86°55′31″E	8,848	none	1953	145 (121)
2	K2/ Godwin Austen	8,611	28,251	Baltoro Karakoram	Kashmir (Pakistan / Xinjiang)	35°52′57″N 76°30′48″E	4,017	Mount Everest	1954	45 (44)
3	Kangchenjunga	8,586	28,169	Kangchenjunga Himalaya	Nepal / India	27°42′09″N 88°08′49″E	3,922	Mount Everest	1955	38 (24)
4	Lhotse	8,516	27,940	Mahalangur Himalaya	Nepal / Tibet	27°57′42″N 86°55′59″E	610	Mount Everest	1956	26 (26)
5	Makalu	8,485	27,838	Mahalangur Himalaya	Nepal / Tibet	27°53′21″N 87°05′19″E	2,386	Mount Everest (Lhotse)	1955	45 (52)
6	Cho Oyu	8,188	26,864	Mahalangur Himalaya	Nepal / Tibet	28°05′39″N 86°39′39″E	2,340	Mount Everest	1954	79 (28)
7	Dhaulagiri	8,167	26,795	Dhaulagiri Himalaya	Nepal	28°41′45″N 83°29′36″E	3,357	Mount Everest	1960	51 (39)
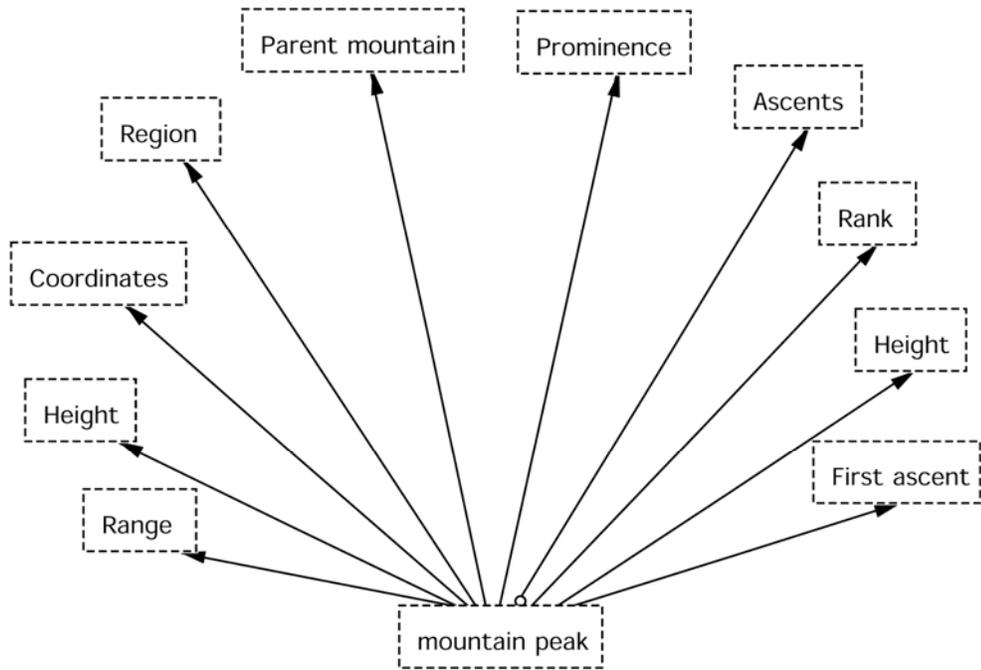
U.S. Muslim Population Table

Ethnic Grouping	Population 1000 (1990)	Percent of Total Muslim Population	Definition of Terms
African-American	2,100	42.0	**African-Americans:** Those persons of African descent native to the United States of America.
South Asians	1,220	24.4	**South-Asians:** Those of Indian/Pakistani, Bangladeshi, Sri Lankan, or Afghan descent now residing in the United States as citizens or permanent residents.
Arabs	620	12.4	**Arabs:** People from Arabic-speaking countries of the Middle East and North Africa who are permanent residents or citizens of the United States.
Africans	260	5.2	**Africans:** People from the African continent who are citizens or permanent residents of the United States
Iranians	180	3.6	**Iranians:** People of Persian descent, usually from Iran, who are citizens or permanent residents.

Population | Percent of Total Muslim Population | Definition of Terms

1990

Ethnic Grouping
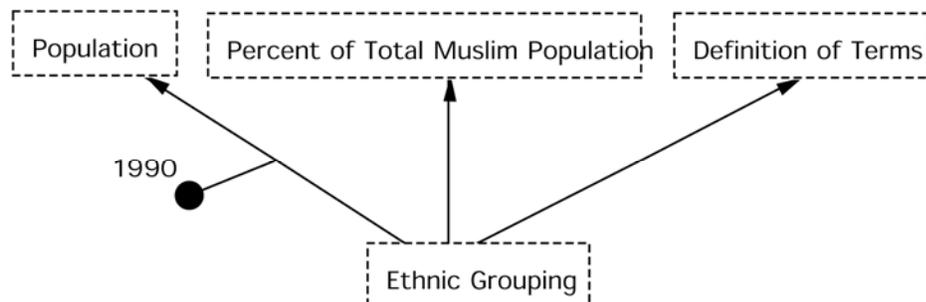
Muslim State Population Table

State	Muslim Population (1,000)	Percentage Total Muslim Population	Percent of Total State Population
California	1,000	20.0	3.4
New York	800	16.0	4.7
Illinois	420	8.4	3.6
New Jersey	200	4.0	2.5
Indiana	180	3.6	3.2
Michigan	170	3.4	1.8
Virginia	150	3.0	2.4
Texas	140	2.8	0.7
Ohio	130	2.6	1.2
Maryland	70	1.4	1.4

Muslim Population   Percentage Total Muslim Population   Percent of Total State Population

state
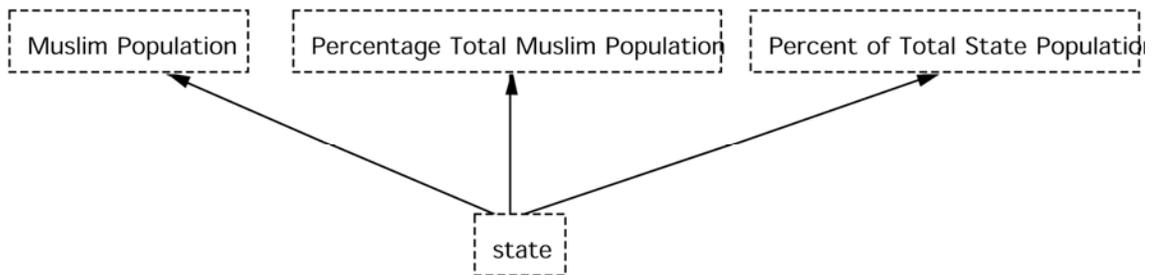
MEMBERSHIP	572,619
MISSIONS	15
TEMPLES	1
CONGREGATIONS	1,075
FAMILY HISTORY CENTERS	156
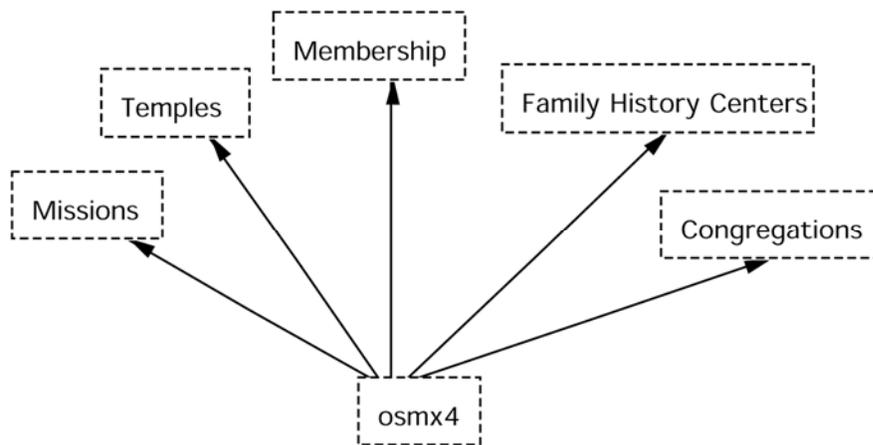
MEMBERSHIP	970,903
MISSIONS	27
TEMPLES	4
CONGREGATIONS	1,756
FAMILY HISTORY CENTERS	284
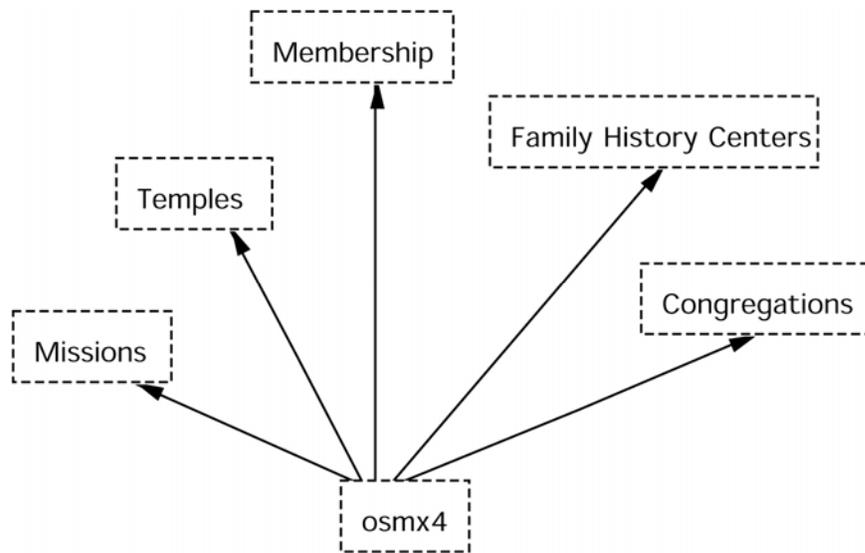
Tax Rates Around the World

(Note: Only the __underlined__ countries are currently ready)

Country	Income Tax		VAT
	Corporate	Individual	
Argentina	35%	9-35%	21%
Australia	30%	17-47%	10%GST
Austria	25%	21%-50%	20%GST
Belgium	33.99%	25-50%	21%
Brazil	34%	15-27.5%	17-25%
Bulgaria	10%	10%	20%
Canada	19.5%(federal)	15-29%(Federal)	5%(gst)
China	25%	5-45%	17%
Cyprus	10%	20-30%	15%
Czech Rep.	21%	15%	19%
Denmark	24%	38-59%	25%
Egypt	40%	20-40%	-

Income Tax Individual

VAT

Income Tax Corporate

country

49
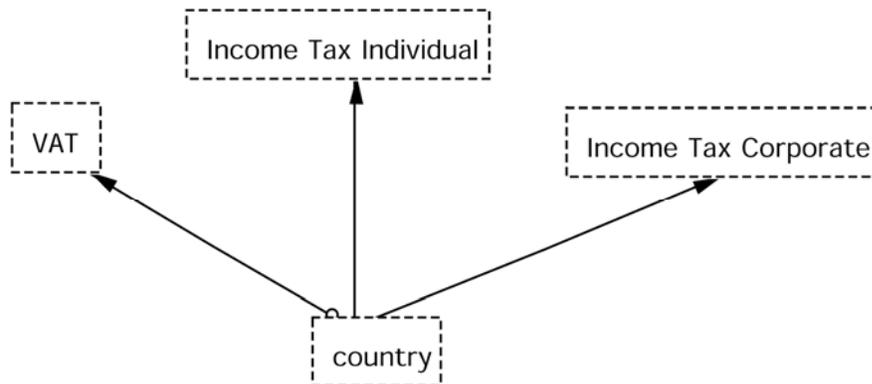
Table 1: Distribution of net worth and financial wealth in the United States, 1983-2001

Total Net Worth

	Top 1 percent	Next 19 percent	Bottom 80 percent
1983	33.8%	47.5%	18.7%
1989	37.4%	46.2%	16.4%
1992	37.2%	46.6%	16.3%
1995	38.5%	45.4%	16.1%
1998	38.1%	45.3%	16.6%
2001	33.4%	51.0%	15.5%

Total Net Worth Next 19 percent

Total Net Worth Top 1 percent

Total Net Worth Bottom 80 percent

Year
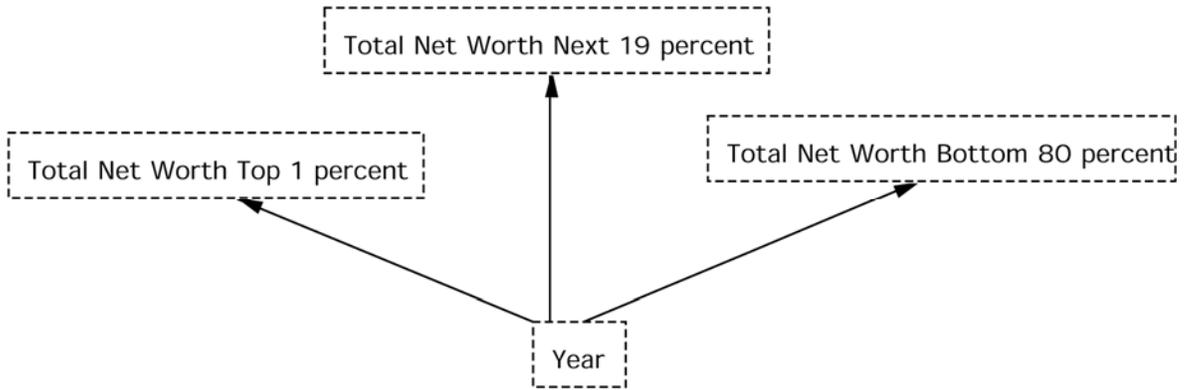
Table 2: Wealth distribution by type of asset, 2001

Investment Assets

	Top 1 percent	Next 9 percent	Bottom 90 percent
Business equity	57.3%	32.3%	10.4%
Financial securities	58.0%	30.6%	11.3%
Trusts	46.3%	40.4%	13.3%
Stocks and mutual funds	44.1%	40.4%	15.5%
Non-home real estate	34.9%	43.6%	21.5%
TOTAL	47.8%	37.7%	14.5%

Investment Assets Bottom 90 percent

Investment Assets Next 9 percent

Investment Assets Top 1 percent

osmx26
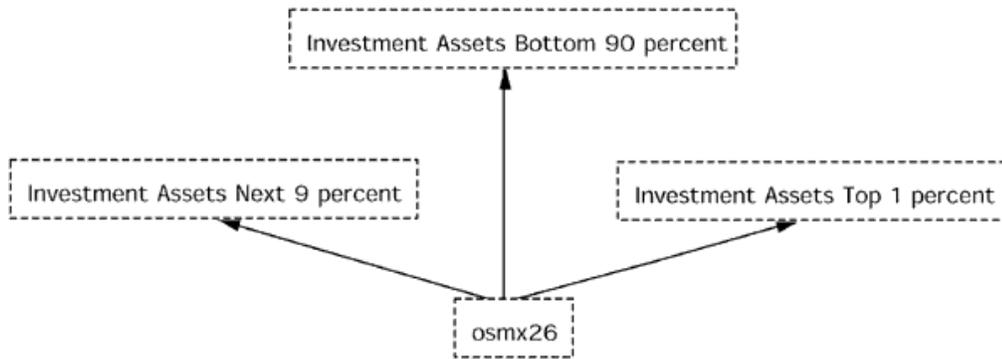
Table 4: Percentage of wealth held by the Top 10% of the adult population in various Western countries

country	wealth owned by top 10%
Switzerland	71.3%
United States	69.8%
Denmark	65.0%
France	61.0%
Sweden	58.6%
UK	56.0%
Canada	53.0%
Norway	50.5%
Germany	44.4%
Finland	42.3%

wealth owned by top 10%

↑

country
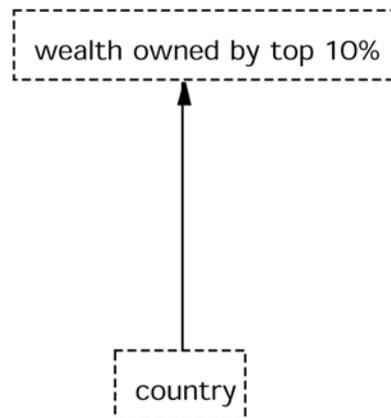
Region	Percent of world population	Percent of world net worth (PPP)	Percent of world net worth (exchange rates)	Percent of world GDP (PPP)	Percent of world GDP (exchange rates)
North America	5.17	27.1	34.39	23.88	33.67
Central/South America	8.52	6.51	4.34	8.49	6.44
Europe	9.62	26.42	29.19	22.8	27.06
Africa	10.66	1.52	0.54	2.36	1.01
Middle East	9.88	5.07	3.13	5.69	4.1
Asia	52.18	29.4	25.61	31.07	24.1
Other	3.14	3.7	2.56	5.4	3.38
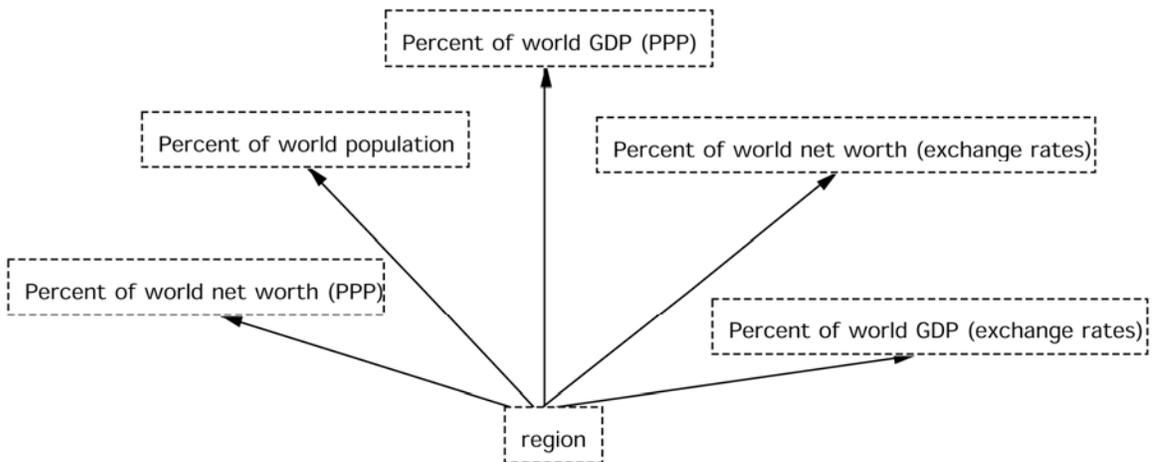
Five most populous incorporated places in the United States (2006)[141][142]					
Rank	**City**	**Population** within city limits	**Metropolitan Area**		**Region**[143]
			population	rank	
1	New York City	**8,214,426**	18,818,536	1	Northeast
2	Los Angeles	**3,849,378**	12,950,129	2	West
3	Chicago	**2,833,321**	9,505,748	3	Midwest
4	Houston	**2,144,491**	5,539,949	6	South
5	Phoenix	**1,512,986**	4,039,182	13	West
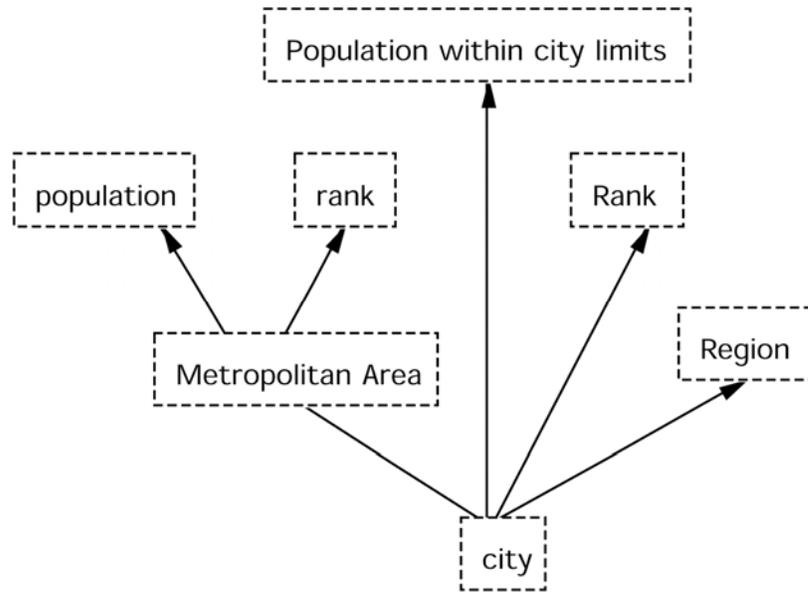
no.	name	calculation 2008	latitude	longitude	country	name variants	metropolitan area
1	Abidjan	3 900 546	5.33°N	4.03°W	Ivory Coast		
2	Accra	2 280 216	5.56°N	0.20°W	Ghana		
3	Āddīs Ābebā	3 144 918	9.03°N	38.74°E	Ethiopia	Addis Abeba, Adis Abeba, Addis Ababa, Finfinnee, Shaggar	
4	Āgra	1 590 073	27.19°N	78.01°E	India	Agara, Akbarabad	
5	Ahmadābād	3 867 336	23.03°N	72.58°E	India	Amdabad, Ahmedabad, Ahmadabad	
6	al-Baṣrah	1 861 523	30.50°N	47.83°E	Iraq	al-Basrah, Basra, Bassora, Bassorah, Bussora, Bussorah, Busra, Busrah	
7	Aleppo	1 671 673	36.23°N	37.17°E	Syria	Ḥalab, Halab, Haleb, Alep, Heleb	Halab
8	Alexandria	4 247 414	31.22°N	29.95°E	Egypt	al-Iskandarīyah, al-Iskandariyah, El Iskandariya, Alexandrie	
9	Algiers	2 159 051	36.77°N	3.04°E	Algeria	al-Jazā'ir, Alger, Algier, Al-Jazair, Al-Jaza'ir, El Djazaïr, El-Jazaïr, El-Djezaïr, Tamurt n Ldzayer	
10	al-Ḩarṭūm Baḩrī	1 594 833	15.64°N	32.52°E	Sudan	Šarj an-Nīl, Kharṭum Bahrī, Khartoum Bahri, Khartum North	al-Khartum
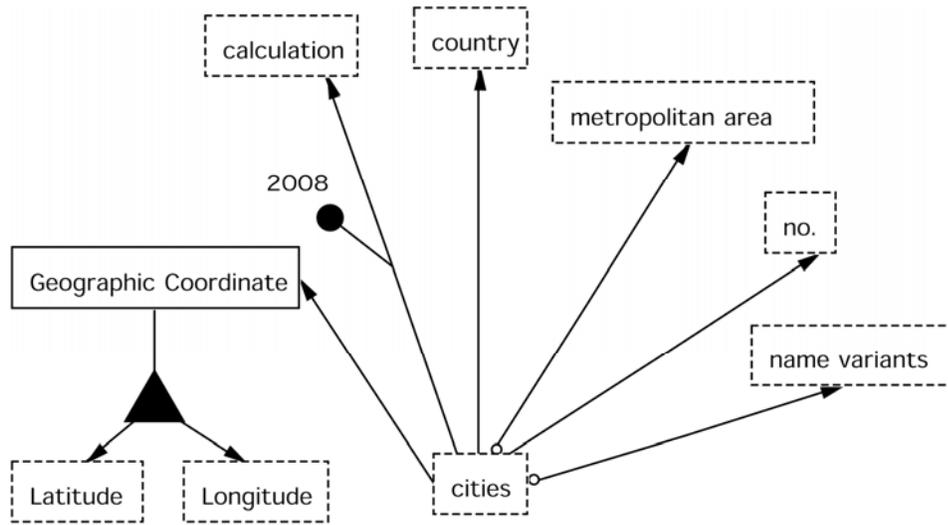
Gross domestic product, income-based

	2003	2004	2005	2006	2007
	\$ millions				
Gross domestic product at market prices	**1,213,175**	**1,290,828**	**1,375,080**	**1,446,307**	**1,531,427**
Wages, salaries and supplementary labour income	621,003	654,957	694,041	737,382	782,290
Corporation profits before taxes	144,501	169,151	189,357	198,859	210,426
Government business enterprise profits before taxes	12,604	12,923	14,578	13,823	15,455
Interest and miscellaneous investment income	49,989	54,109	61,070	65,310	68,684
Accrued net income of farm operators from farm production	1,439	3,106	1,321	344	582
Net income of non-farm unincorporated business, including rent	77,181	81,037	83,636	85,980	89,777
Inventory valuation adjustment	4,262	-1,747	-933	-1,775	2,968
Taxes less subsidies on factors of production	56,072	58,998	61,847	64,421	66,949
Net domestic product at basic prices	967,051	1,032,534	1,104,917	1,164,344	1,237,131
Taxes less subsidies on products	84,380	89,838	94,334	97,161	100,133
Capital consumption allowances	161,817	168,274	176,338	184,750	193,814
Statistical discrepancy	-73	182	-509	52	349

Source: Statistics Canada, CANSIM, table (for fee) 380-0001 and Catalogue no. 13-001-X.
Last modified: 2008-03-03.

```
        +----------+
        | osmx12   |
        +----------+
             |
        +----------+
        | osmx10   |
        +----------+
             |
        +----------+
        | osmx25   |
        +----------+
             |
             |------------>  +----------+
             |               | osmx26   |
             |               +----------+
        +----------+
        | Year     |
        +----------+
```
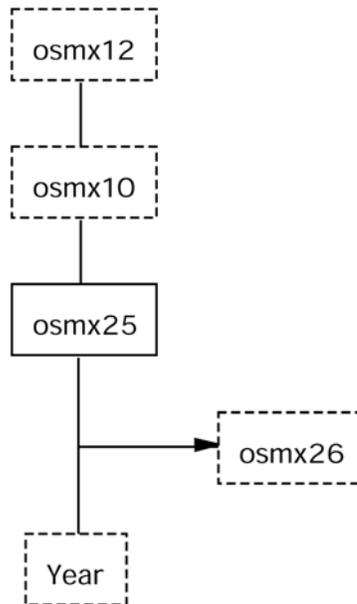
Land and freshwater area, by province and territory

	Total area	Land	Freshwater	% of total area
	km²			
Canada	9,984,670	9,093,507	891,163	100.0
Newfoundland and Labrador	405,212	373,872	31,340	4.1
Prince Edward Island	5,660	5,660	0	0.1
Nova Scotia	55,284	53,338	1,946	0.6
New Brunswick	72,908	71,450	1,458	0.7
Quebec	1,542,056	1,365,128	176,928	15.4
Ontario	1,076,395	917,741	158,654	10.8
Manitoba	647,797	553,556	94,241	6.5
Saskatchewan	651,036	591,670	59,366	6.5
Alberta	661,848	642,317	19,531	6.6
British Columbia	944,735	925,186	19,549	9.5
Yukon Territory	482,443	474,391	8,052	4.8
Northwest Territories	1,346,106	1,183,085	163,021	13.5
Nunavut	2,093,190	1,936,113	157,077	21.0

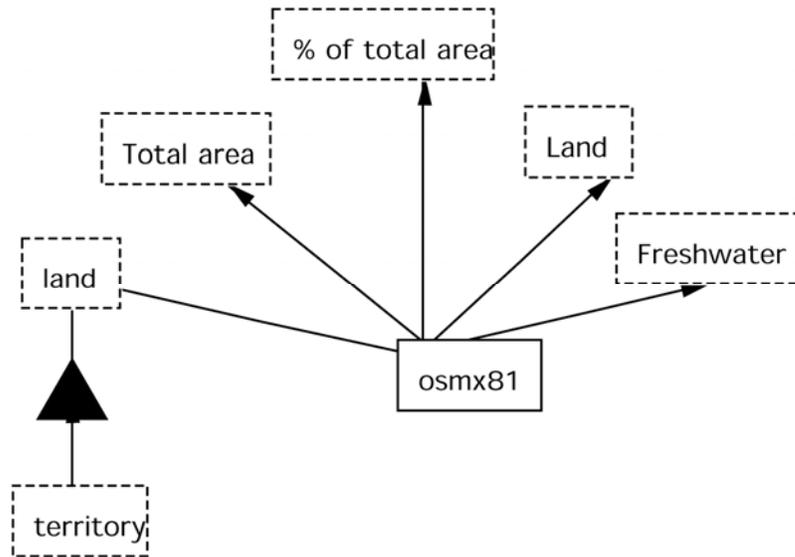**Source:** Natural Resources Canada, GeoAccess Division.
Last modified: 2005-02-01.

Table 2.2 Current AAC on Crown Lands

	Units	NL[a]	PE[b]	NS	NB	QC	ON	MB[c]	SK	AB	BC
Start of Current Period	year	2006	2000	2001	2002	1995	2001	2006	2001	2005	2003
End of Current Period	year	2010	2010	2005	2007	1999	2005	2010	2005	2006	2003
Total Crown Land	000 ha	37 856	263	1 100	3 362	74 286	74 103[d]	25 942[g]	27 881	32 974[m]	87 553
Productive forest Crown Land	000 ha	8 506	256	1 030	3 050	45 907	28 552[e]	12 299[g]	11 975	22 464[m]	49 145
Area available for harvest[*]	000 ha	2 353	216	690	2 901	N/A	23 986[f]	N/A[g]	N/A[i]	N/A	22 934[k]
Current AAC softwood	000 m^3	2 312	300	900	3 494	32 070	20 653	5 673[h]	4 834	15 052[i]	71 116[l]
Current AAC hardwood	000 m^3	180	160	350	1 870	12 598	11 078	2 740[h]	3 388	11 172[j]	3 218
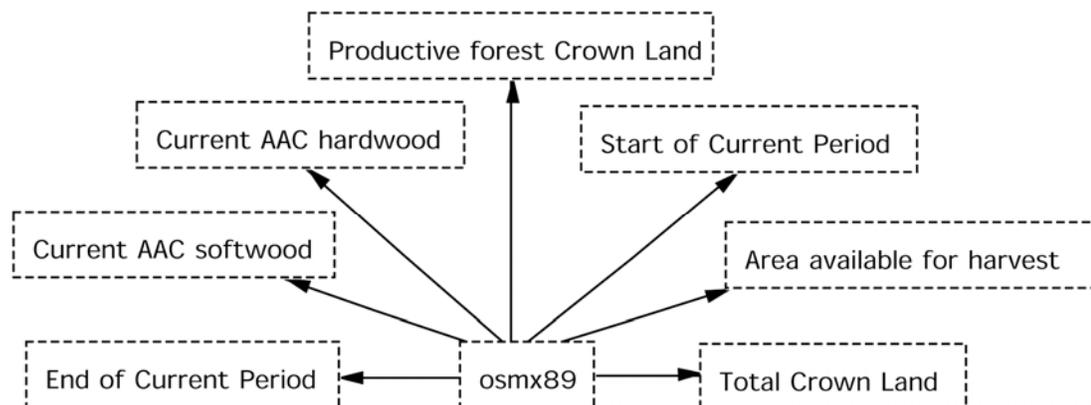
Newfoundland – Island Only

2.4 Wood Supply Projections

Crown Land Industrial Tenure[a]						
	Units	2000	2010	2050	2100	2150
Annual supply softwood	000 m^3	1 987	1 987	1 987	1 987	1 987
Growing stock softwood	000 m^3	104 650	102 050	95 050	94 400	103 950
Annual supply hardwood[b]	000 m^3	N/A	N/A	N/A	N/A	N/A
Growing stock hardwood[b]	000 m^3	N/A	N/A	N/A	N/A	N/A
Regenerating area	%	23	22	24	26	28
Immature area	%	27	30	35	40	38
Mature area	%	50	48	41	34	24

Growing stock softwood   Immature area   Mature area

Regenerating area   Annual supply hardwood b

Annual supply softwood   Year   Growing stock hardwood b

59
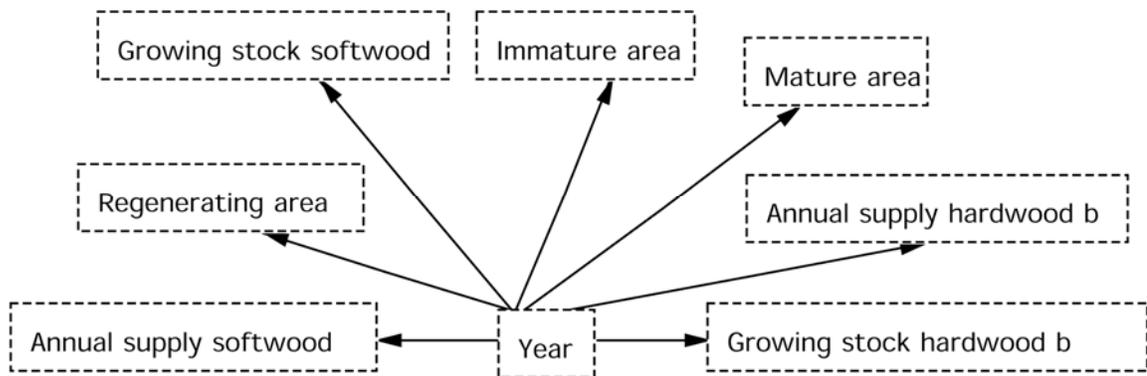
Canadian Island National Parks and Island National Park Reserves

Name of Park/Reserve	Name of Island	Date of Creation	Area (km²)
Arctic Islands			
Aulavik	Banks	1992	12 200.0
Quttinirpaaq	Ellesmere	1988	37 775.0
Sirmilik	Baffin	1992	22 252.0
Auyuittuq	Baffin	1976	19 707.4
Pacific Islands			
Gwai Haanas	Queen Charlotte	1988	1 495.0
Pacific Rim	Vancouver Broken Group	1987	285.8
Gulf Islands	Gulf Islands	2003	33.0

canadian province

Name of Island

Date of Creation

osmx56

Area

Name of Park/Reserve
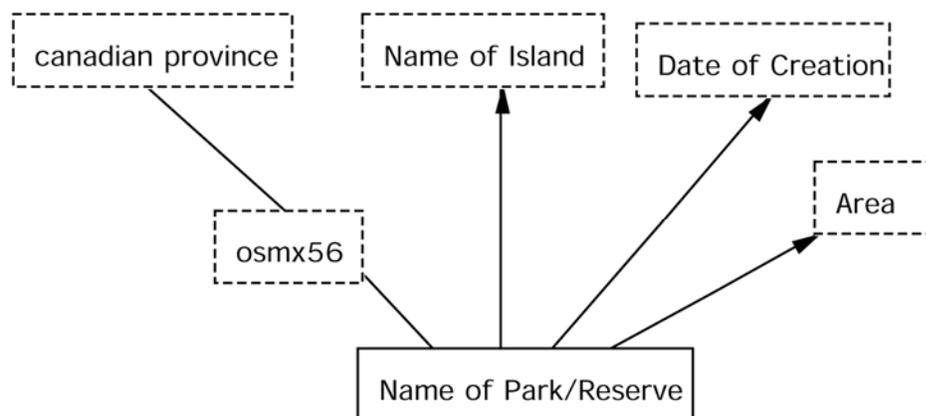
Province or territory	Peak	Range or other region	Height (m)	Height (ft)
Yukon	Mount Logan	St. Elias Mountains	5,959	19,950
British Columbia	Mount Fairweather *	St. Elias Mountains	4,663	15,299
Alberta	Mount Columbia**	Rocky Mountains	3,747	12,293
Northwest Territories	Unnamed Peak, referred to as Mount Nirvana	Backbone Ranges	2,773	9,098
Nunavut	Barbeau Peak	British Empire Range	2,616	8,583
Newfoundland and Labrador	Mount Caubvick ***	Torngat Mountains	1,652	5,420
Quebec	Mont D'Iberville ***	Torngat Mountains	1,651	5,417
Saskatchewan	Unnamed point	Cypress Hills	1,468	4,816
Manitoba	Baldy Mountain	Duck Mountains	832	2,730
New Brunswick	Mount Carleton		817	2,680
Ontario	Ishpatina Ridge ****		693	2,274
Nova Scotia	White Hill	Cape Breton Highlands	532	1,745
Prince Edward Island	Glen Valley (46°20'N, 63°25'W)	Queens County	142	466
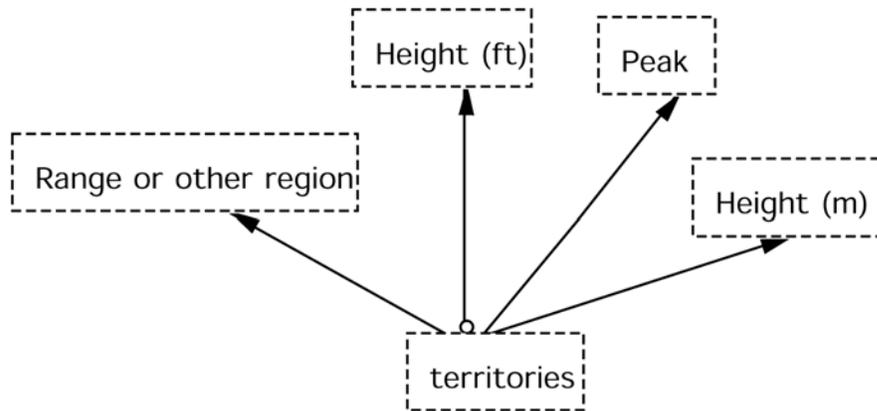
The 100 Highest Major Mountain Peaks of Canada

Rank	Peak	Province	Range	Elevation	Prominence	Isolation
1	Mount Logan[1]	Yukon	Saint Elias Mountains	5,959 m 19,551 ft	5,247 m 17,215 ft	623.4 km 387.4 mi
2	Mount Saint Elias[2]	Yukon Alaska	Saint Elias Mountains	5,489 m 18,008 ft	3,429 m 11,250 ft	41.4 km 26 mi
3	Mount Lucania	Yukon	Saint Elias Mountains	5,226 m 17,146 ft	3,051 m 10,010 ft	43 km 26.7 mi
4	King Peak	Yukon	Saint Elias Mountains	5,173 m 16,972 ft	1,053 m 3,455 ft	4.9 km 3.1 mi
5	Mount Steele	Yukon	Saint Elias Mountains	5,000 m 16,404 ft	720 m 2,362 ft	9.2 km 5.7 mi
6	Mount Wood	Yukon	Saint Elias Mountains	4,840 m 15,879 ft	1,160 m 3,806 ft	18.6 km 11.5 mi
7	Mount Vancouver	Yukon	Saint Elias Mountains	4,812 m 15,787 ft	2,712 m 8,898 ft	43.9 km 27.3 mi
8	Mount Slaggard	Yukon	Saint Elias Mountains	4,742 m 15,558 ft	502 m 1,647 ft	7.2 km 4.5 mi
9	Fairweather Mountain[3]	British Columbia Alaska	Saint Elias Mountains	4,671 m 15,325 ft	3,957 m 12,983 ft	200.7 km 125 mi