Generalization Lattices

Howard J. Hamilton, Robert J. Hilderman,
Liangchun Li, and Dee Jay Randall

Department of Computer Science
University of Regina
Regina, Saskatchewan, Canada S4S 0A2
{hamilton,hilder,lil,randal}@cs.uregina.ca

Abstract. Generalization lattices encode domain knowledge relevant to generalization. They provide a convenient framework for data visualization during user-guided exploration and for automated guidance during independent exploration. To reduce the size of a generalization lattice for an individual attribute, we define six types of pruning. Then we consider the generalization space defined by the cross product of lattices for several attributes. To increase the relevance of the data exploration results, we define five additional types of pruning. An interactive, web-based system for visualizing the generalization space allows the user to interactively guide the data exploration process.

1 Introduction

Generalization is fundamental to knowledge discovery and data mining. To provide a high-level view of the generalization operations permitted on data, various researchers have proposed similar structures, here called *generalization lattices* (*GLs*), but variously called *type hierarchies* [12, 9], *concept generalization graphs* [10], *domain generalization graphs* [6], *dependency lattices* [7], and *attribute lattices* [3]. A generalization lattice is an effective form of knowledge representation for encoding prior domain knowledge relevant to generalization.

In simple form, a generalization lattice shows the relationships between domains of values. Each generalization operation can be regarded as a mapping from one domain to a smaller domain. If a lattice is defined on the domains $D_1 = \{A, B, C\}, D_2 = \{A, BorC\}, D_3 = \{AorB, C\}, D_4 = \{AorBorC\}$, then D_1 is the most specific domain, with 3 possible values, D_2 and D_3 are domains of intermediate generality with two possible values each and no ordering defined between them, and D_4 is the most general domain, with one possible value. In the GL, an arc from node D_i to node D_j indicates that any value in domain D_i maps to a value in D_j. Mitchell's version space method is an example of a learning method based on a GL. Godin, Missaoui, and Alaoui's work on incremental concept formation is based on Galois lattices, a type of GL used in discrete mathematics [5]. Bournaud and Ganascia investigated the automatic creation of a GL from a set of objects described by conceptual graphs[1].

In data mining, GLs are used to conceptualize the process of generalizing data as a transformation of values from one domain to values of another, smaller

domain. The original data, as retrieved from a database or other source, is considered the most specific representation of the data. Applying an operation to the data that maps any of several values in one domain to a single value in a smaller domain corresponds to traversing an arc in the GL from one node to a higher node. Many generalizations are possible, but for data mining, it is efficient and effective to limit the nodes to those representing generalized encodings of the domain that are of interest to the users of the knowledge discovery system. A GL is appropriate for this task because the possible generalizations form a partial order rather than a strict hierarchy (e.g., days can be generalized to weeks or months, but weeks cannot be generalized to months). A GL also allows a user to guide the generalization process by defining the domains of values to be considered in the data exploration process. Placing a node for a domain in a GL documents an inductive bias, namely that the partition of the original data values represented by that domain is at a level of granularity that the user finds interesting.

This work was motivated by the need to automate the data exploration process for cases where many ways of generalization may be be appropriate. For example, given a database with a time-related attribute, summaries can be created according to a GL containing the *hour of day*, *part of day*, *day*, *day of week*, *day of month*, *week*, *week in month*, *week in quarter*, *month*, *year*, and many others. Our system not only creates all these summaries, but also ranks them to help identify any anomalies, such as a disproportionate percentage of sales activity in the first week of a month. Furthermore, all attributes of interest can have arbitrarily complex GLs and our system will consider all resulting combinations. These features enable a database analyst to analyze the database from many different perspectives.

The remainder of this paper is organized as follows. In Section 2, we formally define GLs, and present an example GL for calendar attributes. In Section 3, we describe a semi-automated method for data exploration which uses pruning to identify the nodes of a GL that are distinct for a particular set of data. Six types of pruning are defined: reachability, preliminary manual, data-range, previous-discard, pregeneralization manual, and post-generalization. In Section 4, we consider the generalization space defined by the cross product of GLs associated with a set of attributes. We define five additional types of pruning and several measures for ranking the interestingness of the nodes in the generalization space. We also describe an interactive, web-based system for visualizing the generalization space that allows the user to interactively guide the exploration process and view the results. In § 5, we present conclusions.

2 Generalization Lattices

Given the domain of an attribute represented by a set $S = \{s_1, s_2, \ldots, s_n\}$, S can be partitioned in many different ways. For example, $D_1 = \{\{s_1\}, \{s_2\}, \ldots, \{s_n\}\}$, $D_2 = \{\{s_1\}, \{s_2, \ldots, s_n\}\}$, etc. Let D be the set of partitions of set S, and $\preceq$ be a nonempty binary relation (called a *generalization relation*) defined on D, such

that $D_i \preceq D_j$ if for every $d_i \in D_i$ there exists $d_j \in D_j$ such that $d_i \subseteq d_j$. The generalization relation $\preceq$ is a partial order relation and $\langle D, \preceq \rangle$ defines a partial order set from which we can construct a lattice called a *generalization lattice* $\langle D, E \rangle$ as follows. First, the nodes of the graph are elements of D. And second, there is a directed arc from D_i to D_j (denoted by $E(D_i, D_j)$) iff $D_i \neq D_j$, $D_i \preceq D_j$, and there is no $D_k \in D$ such that $D_i \preceq D_k$ and $D_k \preceq D_j$. The partial order set $\langle D, \preceq \rangle$ is transitively closed and is a lattice.

Let $D_g = \{S\}$ and $D_d = \{\{s_1\}, \{s_2\}, \ldots, \{s_n\}\}$. For any $D_i \in D$ we have $D_d \preceq D_i$ and $D_i \preceq D_g$, where D_d and D_g are called the bottom and top of D, respectively. We call the nodes (elements of D) *domains*, where the bottom is the *most specific level of generality* and the top is the *most general level*. There is a trivial GL where the bottom is mapped directly to the top (i.e., D_d is mapped to D_g). For each node D_i in $\langle D, E \rangle$, we define $descendants(D_i)$ to be all nodes D_j such that $D_i \preceq D_j$ and $ancestors(D_i)$ to be all nodes D_k such that $D_k \preceq D_i$.

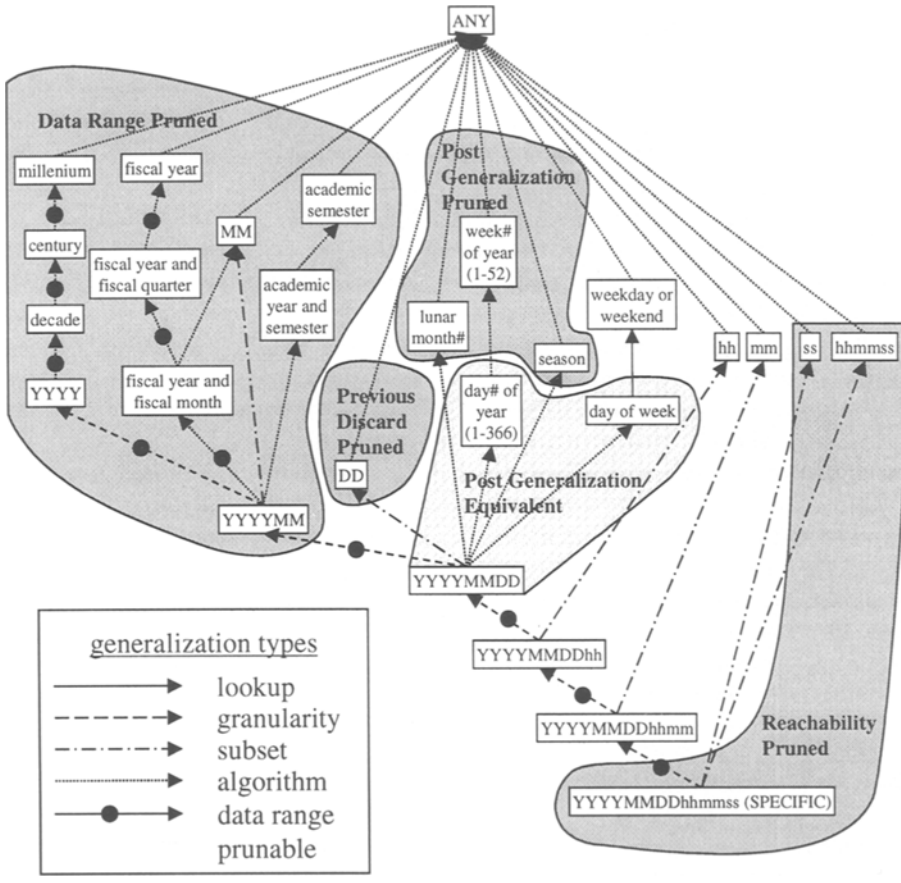

Fig. 1. Calendar GL with shading indicating pruning

We now describe an example GL for a calendar attribute, as shown in Figure 1, adapted from [11]. (The shading is explained in Sec. 3.) All attributes related to the time of an event's occurrence are combined into a *calendar attribute*, which contains subattributes such as year, month, etc. This GL is larger than the example GLs given in previous reports ([10], [3], [6], [7]), but the additional complexity is required to illustrate our method. In Figure 1, the node labelled *YYYYMMDDhhmmss* represents the most specific domain considered (i.e., the finest granularity of our calendar domain is one second). Every other node represents a generalization of this domain, and the arcs connecting the nodes represent generalization relations. To handle data containing calendar values specified to finer granularity (e.g., microseconds), more specific nodes could be added to the GL. A GL is specified for each attribute to be generalized.

Specification of a generalization relation is done using one of four techniques [11]: (1) granularity generalization for dropping in sequence the least significant subattribute, e.g., first drop *ss* and then *mm*; (2) subset generalization for dropping any combination of subattributes; (3) lookup generalization for explicitly specifying the mapping of values between the more specific and more general domains; and (4) algorithmic specification for all other cases.

3 Adaptation of Generalization Lattices

To guide the user quickly to the most interesting results, a GL can be manually and automatically pruned during the knowledge discovery process. Two automatic pruning techniques are: (1) based on a superficial examination of the data, the GL is pruned prior to generalization according to three heuristics, and (2) during generalization, if a step results in either no reduction in the number of values, or a complete reduction to one value, special processing is used. After all pruning is complete, the resulting GL can be displayed to guide the user to the generalizations of interest. In tasks involving multiple attributes, the method's first five steps can be applied independently to each attribute with a GL (see Sec. 4). In such cases, pruning would be particularly advantageous.

A GL for an attribute can be pruned in six steps, as follows.

Reachability Pruning: Once the user has specified how to map the data to a node in the GL, all nodes not reachable from this node are pruned.

Preliminary Manual Pruning: The user can hide any interior node regarded as uninteresting, and it is not subsequently displayed. To preserve the integrity of the GL for generalization, some hidden nodes are retained and used in the generalization process. The GL must not become disconnected. Any hidden node adjacent to "ANY" can be pruned and its incoming arcs can be directed to "ANY". Nodes with children may be pruned only if those children are still reachable afterwards.

Data-Range Pruning: Any node that does not correspond to a distinction in the data is removed. Some arcs in the GL represent monotonic functions mapping their domain (parent node data) to their range (child node data). Given a set of values A and some monotonic arc $E : A \rightarrow A_g$, we let $m = min(A)$,

$M = max(A)$, $E : m \rightarrow g$, and $E : M \rightarrow G$. Because E is monotonic, $\forall a \in A, E : a \rightarrow a_g, g \leq a_g \leq G$. For example, the range of data for a calendar attribute is determined by identifying its minimum and maximum values. These two values are generalized along all monotonic arcs, i.e., those permitting data range pruning. If these two values generalize to the same value at any node in the GL, then all occurring values for the calendar attribute generalize to this value, as described above. This node and all its descendants can be pruned, i.e., conceptually joined with the "ANY" node. Granularity generalizations are inherently monotonic, while algorithmic generalizations may or may not be.

Previous-Discard Pruning: Any node that is indistinguishable from another node except for information that data-range pruning has shown to be irrelevant is removed. This method is convenient for granularity and subset generalizations. If we are considering pruning node B, which is a generalization of node A, we look at what information is discarded when data is generalized from node A to node B. If we have already chosen to prune a node C that contains either exactly the subattributes or a superset of the subattributes that we are discarding when generalizing from node A to node B, then we should prune node B. Previous analysis has shown that at node C, the data will contain only a single value; thus, the information in node C does not distinguish any values. We do not automatically prune children of node B.

Pregeneralization-Manual Pruning: Again, the user is allowed to prune nodes from the GL. At this point, the time-consuming work of actually generalizing the data has not yet been done. Pruning nodes at this point may substantially reduce the time and space required to generalize the values.

Post-Generalization Pruning: The original data is now generalized step by step according to the GL, with each node corresponding to the data at that specified level of granularity and each arc corresponding to one transformation of the data. After each generalization step, we consider the number of values in the generalized data. If only one distinct value remains after the generalization step, then the corresponding node and any other interior node reachable from it can be pruned. Otherwise, if the number of values is the same before and after the step, then the data have been transformed by a one-to-one mapping rather than by a true generalizing, and we prune by conceptually joining the two nodes. This conceptual joining of nodes is transitive.

Example: We illustrate pruning for a calendar attribute. The input data are 8132 login times, collected over a one week period in January 1998: Jan 18 1998 00:26, Jan 18 1998 00:55, Jan 18 1998 01:21, ..., Jan 24 1998 23:48}. Times are not recorded to seconds.

Given this data and the (unshaded) GL shown in Figure 1, generalization and pruning proceeds as follows. First, the user identifies the initial node as *YYYYMMDDhhmm*. Reachability pruning (step 1) removes nodes *YYYYMMDDhhmmss*, *hhmmss*, and *ss*. We assume no preliminary manual pruning (step 2). For data range pruning (step 3), the minimum and maximum date values in the data are found to be `Jan 18 1998 00:26` and `Jan 25 1998 23:48`, respectively. These two dates are generalized by following all arcs allowing data range

pruning (most arcs on the lower left of Figure 1). Both values generalize to the same value at *YYYYMM*. Node *YYYYMM* and all its children are pruned (the nodes in the upper left side of Figure 1). Previous-discard pruning (step 4), *DD* is pruned because *YYYYMM* has been pruned and generalization from *YYYYMMDD* to *DD* is based on discarding *YYYYMM*. We assume no pregeneralization manual pruning (step 5). Finally, the data are generalized, guided by the pruned GL. After each generalization step, the result is checked for further pruning. For example, when the data are generalized from *YYYYMMDD* to *day# of year*, the number of values remains constant, indicating that results corresponding to only one of these nodes should be shown to the user. Thus, these nodes can be composed. Similarly, *YYYYMMDD* is also composed with *day of week*, and *weekday name*. When the data is generalized to *season*, *lunar month*, or *week# of year*, only one value remains; thus, all of these nodes are pruned.

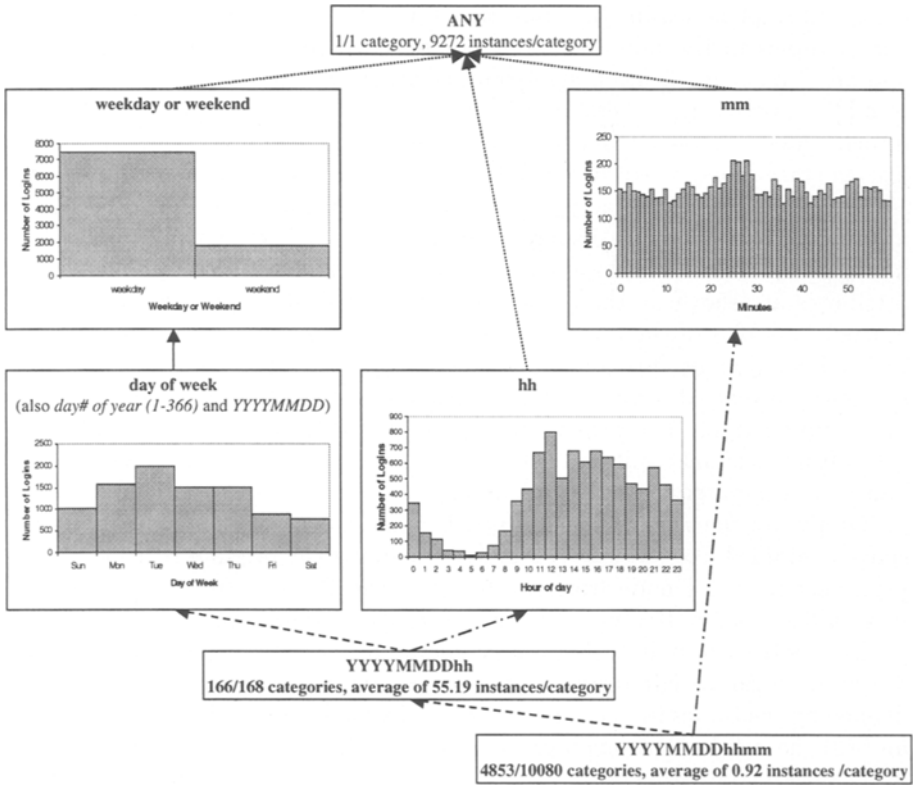

Fig. 2. Final Calendar GL

In Figure 2, we show the nodes remaining after pruning, enhanced where feasible with 2-D plots of the results. Each node gives a summary at a distinct level of temporal generality, e.g., the plot for node *hh* shows the number-of-logins

vs hour-of-the-day. In our current implementation, each node is simply shown as a colored sphere, and the user must select it to obtain the detailed summary information shown in Figure 2.

4 Multi-Attribute Generalization

Given a set of attributes, each with an associated GL, we consider the generalization space formed by all combinations that include one node from each GL. Each combination represents a separate attribute-oriented induction task, where values for each attribute are independently generalized to the level of generality corresponding to the specified node in that attribute's lattice. In a naive implementation, each combination requires a complete pass over all input data, although by taking advantage of relationships in the GL, smaller intermediate results can be reused [10]. The size of the generalization space depends only on the number of nodes in the associated GLs; it does not depend on the number of tuples in the input relation. For m attributes, a database of n tuples, and an $O(n)$ generalization algorithm, creating all possible summaries requires $O(n \prod_{i=1}^{m} |D^i|)$ time, where $|D^i|$ is the number of nodes in GL D^i. We have implemented practical serial and parallel algorithms for traversing the generalization state space where m is small (≤ 5) and n is large ($> 1{,}000{,}000$) [6, 8].

Our approach to interactive data exploration includes visualizing the generalization space. A sample display from our web-based implementation is shown in Figure 3 for a data exploration task containing three attributes. GLs for three attributes are shown in the lower left, the generalization space is shown in the upper left, a plot of the interestingness versus the number of tuples in the generalized relation is shown in the lower right, and a generalized relation (i.e., summary in textual form) corresponding to one combination of nodes is shown in the upper right. The display of the generalization space is generated from a 3-D VRML (virtual reality modelling language) description, while the two lower panes are generated by Java applets.

Originally, the three GLs contained 4, 8, and 6 nodes; thus, the generalization space contained $4 \times 8 \times 6 = 192$ nodes, including the original relation. Manual pruning removed 1 node from the first GL and 2 nodes from the second GL, leaving $3 \times 6 \times 6 = 108$ nodes in the generalization space shown in Figure 3.

To identify summaries that a user might find most interesting, two measures are used to rank their interestingness: (1) *variance* compares the distribution defined by the tuples in a summary to that of a uniform distribution of the tuples, and (2) the *relative entropy measure* (Kullback-Leibler (KL) distance), which is also used for comparing data distributions in unstructured textual databases [4], compares the distribution defined by the structured tuples in a summary to that of a uniform distribution of the tuples. In Figure 3, more interesting nodes in the generalization space (upper left) are indicated by darker colors, while more interesting nodes in the scatterplot (lower right) are positioned to the right.

To reduce the number of summaries generated during data exploration, it is possible to prune the generalization space based on the interestingness measures.

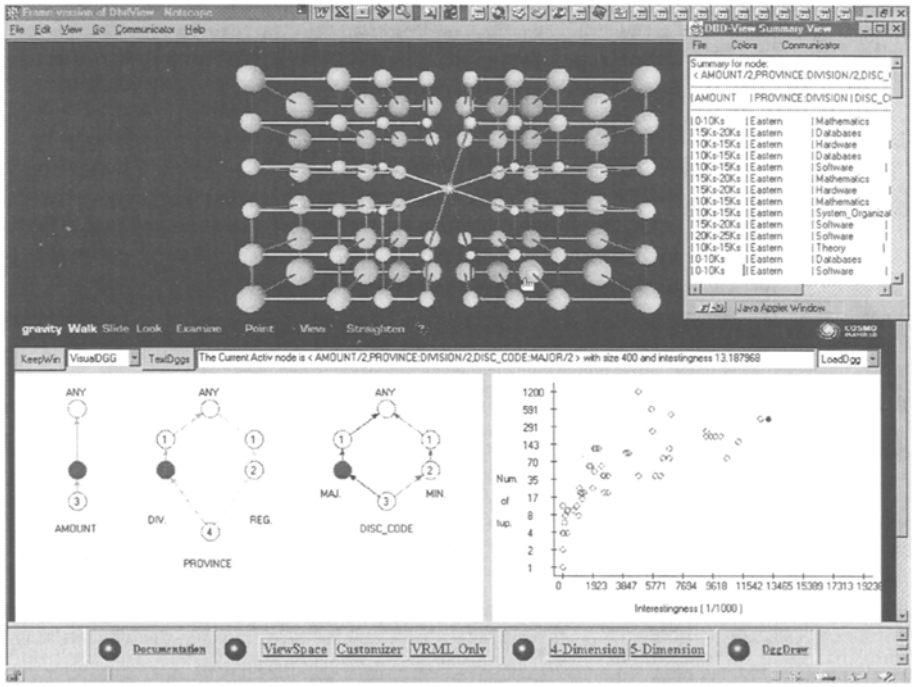

Fig. 3. GSS Display

We define five pruning heuristics as follows.
Ancestor Pruning: If a summary is a direct descendant of some other summary, but has higher interest, then the ancestor can be eliminated.
Descendant Pruning: If a summary is a direct descendant of a summary that has higher interest, then the descendant can be eliminated.
Interestingness Threshold Pruning: All summaries whose degree of interest is less than some user-specified interestingness threshold are deleted.
Table Threshold Pruning: All summaries containing more tuples than some user-specified table threshold are deleted, regardless of their degree of interest. This threshold is commonly used in attribute-oriented induction [2].
Attribute Threshold Pruning: All summaries containing an attribute where the number of distinct values for the attribute is greater than some user-specified attribute threshold, are deleted, regardless of their degree of interest. This threshold is also used extensively in attribute-oriented induction.

5 Conclusion

Generalization lattices allow users to specify the levels of granularity to consider when generalizing a dataset. We showed how pruning heuristics could be used to reduce the size of general-purpose generalization lattices for a specific set of

data. We also showed how the number of combinations in the generalization space could be further pruned by a user, based on a chosen measure of interestingness or other attributes of the generalized relation. Our visual display gives a view of the overall space of possible generalizations. The user can interactively examine specific results and adjust the pruning heuristics.

Acknowledgement: We thank the reviewers for comments. This research was supported by the Natural Sciences and Engineering Research Council of Canada and the Institute for Robotics and Intelligent Systems.

References

1. I. Bournaud and J.-G. Ganascia. Accounting for domain knowledge in the construction of a generalization space. In *Proceedings of the Third International Conference on Conceptual Structures*, pages 446–459. Springer-Verlag, August 1997.
2. Y. Cai, N. Cercone, and J. Han. Attribute-oriented induction in relational databases. In G. Piatetsky-Shapiro and W. Frawley, editors, *Knowledge Discovery in Databases*, pages 213–228, Cambridge, MA, 1991. AAAI/MIT Press.
3. S. Chaudhuri and U. Dayal. OLAP and data warehousing. Technical report, AAAI, Newport Beach, CA, August 1997. Tutorial notes.
4. R. Feldman and I. Dagan. Knowledge discovery in textual databases (KDT). In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pages 112–117, Montreal, August 1995.
5. R. Godin, R. Missaoui, and H. Alaoui. Incremental concept formation algorithms based on Galois (concept) lattices. *Computational Intelligence*, 11(2):246–267, 1995.
6. H.J. Hamilton, R.J. Hilderman, and N. Cercone. Attribute-oriented induction using domain generalization graphs. In *Proceedings of the Eighth IEEE International Conference on Tools with Artificial Intelligence (ICTAI'96)*, pages 246–253, Toulouse, France, November 1996.
7. V. Harinarayan, A. Rajaraman, and J.D. Ullman. Implementing data cubes efficiently. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'96)*, pages 205–216, May 1996.
8. R.J. Hilderman, H.J. Hamilton, R.J. Kowalchuk, and N. Cercone. Parallel knowledge discovery using domain generalization graphs. In J. Komorowski and J. Zytkow, editors, *Proceedings of the First European Conference on the Principles of Data Mining and Knowledge Discovery (PKDD'96)*, pages 25–35, Trondheim, Norway, June 1997.
9. G.M. Mineau and R. Godin. Automatic structuring of knowledge bases by conceptual clustering. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):824–829, October 1995.
10. W. Pang, R.J. Hilderman, H.J. Hamilton, and S.D. Goodwin. Data mining with concept generalization graphs. In *Proceedings of the Ninth Annual Florida AI Research Symposium*, pages 390–394, Key West, Florida, May 1996.
11. D.J. Randall, H.J. Hamilton, and R.J. Hilderman. Generalization for calendar attributes using domain generalization graphs. In *Fifth International Workshop on Temporal Representation and Reasoning (TIME'98)*, pages 177–184, Sanibel Island, Florida, May 1998.
12. J.F. Sowa. *Conceptual Structures.* Addison-Wesley, Reading, MA, 1984.