Parallel Knowledge Discovery Using Domain Generalization Graphs

Robert J. Hilderman, Howard J. Hamilton,
Robert J. Kowalchuk, and Nick Cercone

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

Abstract. Multi-Attribute Generalization is an algorithm for attribute-oriented induction in relational databases using domain generalization graphs. Each node in a domain generalization graph represents a different way of summarizing the domain values associated with an attribute. When generalizing a set of attributes, we show how a serial implementation of the algorithm generates all possible combinations of nodes from the domain generalization graphs associated with the attributes, resulting in the presentation of all possible generalized relations for the set. We then show how the inherent parallelism in domain generalization graphs is exploited by a parallel implementation of the algorithm. Significant speedups were obtained using our approach when large discovery tasks were partitioned across multiple processors. The results of our work enable a database analyst to quickly and efficiently analyze the contents of a relational database from many different perspectives.

1 Introduction

Knowledge discovery from database (KDD) algorithms can be broadly classified into two general areas: summarization and anomaly detection. *Summarization algorithms* find concise descriptions of data, such as partitioning the data into disjoint groups. *Anomaly detection algorithms* identify unusual features of data, such as combinations that occur with greater or lesser frequency than expected.

Attribute-oriented induction (AOI) [7, 8, 9] is a summarization algorithm that has been effective for KDD. AOI summarizes the information in a relational database by repeatedly replacing specific attribute values with more general concepts according to user-defined concept hierarchies (CHs). A *concept hierarchy* associated with an attribute in a database is represented as a tree where leaf nodes correspond to actual domain values in the database, intermediate nodes correspond to a more general representation of the domain values, and the root node corresponds to the most general representation of the domain values. For example, a CH for the Location attribute in a sales database is shown in Figure 1(a). Knowledge about the higher level concepts can be learned through generalization of the sales data at each node.

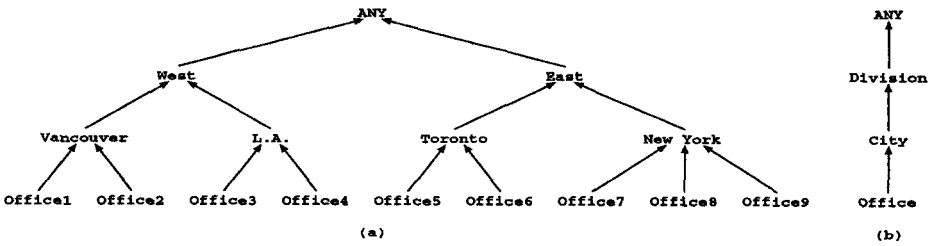

Fig. 1. A concept hierarchy (a) and a domain generalization graph (b)

As the result of recent research, AOI methods are considered among the most efficient of KDD methods for knowledge discovery from databases [1, 2, 3, 4, 7, 10]. In particular, algorithms for generalizing relational databases are presented in [1] that run in $O(n)$ time, where n is the number of tuples in the input relation, and require $O(p)$ space, where p is the number of tuples in the generalized relation (typically $p << n$). In [1], it is also proven that an AOI algorithm which runs in $O(n)$ time is optimal for generalizing a relation.

The complexity of the CHs is a primary factor determining the interestingness of the results [6]. If several CHs are available for the same attribute, which means knowledge about the attribute can be expressed in different ways, current AOI methods require the user to select one. Thus, a fundamental problem is that AOI methods present only one possible generalization to the user without evaluating the relative merits of other possible generalizations consistent with the CHs.

To facilitate other possible generalizations, *domain generalization graphs* (DGGs) were proposed to enable the data in a relational database to be represented in different ways [5, 11]. Informally, a DGG defines a partial order which represents a set of generalization relations for an attribute. A DGG always includes a single source (the node at the lowest level corresponding to the domain of the attribute) and a single sink (the node at the highest level corresponding to the most general representation of the domain and which contains the value ANY). For example, the levels of the CH in Figure 1(a) correspond to the nodes in the more general representation of the DGG in Figure 1(b). Any CH corresponds to a single-path DGG.

When there are multiple single-path DGGs associated with an attribute, a multi-path DGG can be constructed. For example, Figure 2(c) shows how a multi-path DGG can be constructed from the single-path DGGs in Figures 2(a) and 2(b). Here we assume if a common name is used in multiple DGGs, then the name represents the same partition of the domain in the underlying CHs.

In [11], we introduced the Path-Based Generalization (PBG) and Bias-Based Generalization (BBG) algorithms for generalization using DGGs. Although PBG and BBG avoid unnecessary re-generalization by determining which intermediate generalized relations to store for possible future use, the DGG associated with an attribute is considered independently of the DGGs for other attributes. To resolve this problem, we introduced the Serial Multi-Attribute Generalization

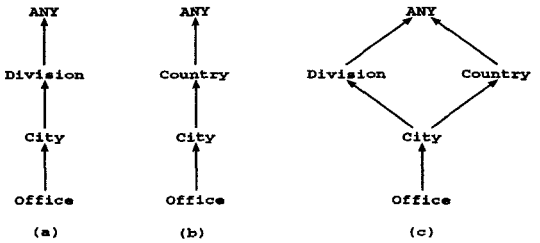

Fig. 2. Constructing a multi-path DGG

algorithm in [5] for generalizing a set of attributes using DGGs. There we show that a set of attributes can be considered a single attribute whose domain is the cross product of the individual attribute domains. A generalization from this domain is described as all possible combinations of nodes from the set of attributes, with one node from the DGG associated with each attribute.

In this paper, we introduce the Parallel Multi-Attribute Generalization algorithm. When generalizing a set of attributes in parallel, a distinct combination of paths from the DGGs associated with the set can be assigned to separate processors, where the generalization along those paths can be done independently of the others. This data parallel algorithm enables us to perform intensive investigation of databases where a few attributes have been determined to be relevant and for which considerable domain knowledge is available (represented as CHs and DGGs). This strategy reflects our experience in applying data mining techniques to a variety of sponsors' commercial databases in the areas of health care, education, and home entertainment.

The remainder of this paper is organized as follows. In the following section, we restate the formal definition of a DGG from [5]. In Section 3, we review the Serial Multi-Attribute Generalization algorithm and introduce the Parallel Multi-Attribute Generalization algorithm. In Section 4, we present experimental results. In Section 5, we summarize our results and suggest future research.

2 Definitions

Given a set $S = \{s_1, s_2, \ldots, s_n\}$ (the domain of an attribute), 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 *domain generalization graph* $\langle D, E \rangle$ 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 $D_k \preceq D_j$.

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 *least* and *greatest elements* of D, respectively. We call the nodes (elements of D) *domains*, where the least element is the *most specific level of generality* and the greatest element is the *most general level.* There is a trivial DGG where the least element is mapped directly to the greatest element (i.e., D_d is mapped to D_g).

For example, given $S = \{$*Vancouver, Toronto, Montreal, Los Angeles, New York, St. Louis*$\}$, let $D = \{$*Office, City, Division, Country, ANY*$\}$, where $D_g = \{S\} = \{\{$*Vancouver, Toronto, Montreal, Los Angeles, New York, St. Louis*$\}\}$, $D_3 = \{\{$*Vancouver, Toronto, Montreal*$\}, \{$*Los Angeles, New York, St. Louis*$\}\}$, $D_2 = \{\{$*Vancouver, Toronto*$\}, \{$*Montreal, Los Angeles*$\}, \{$*New York, St.Louis*$\}\}$, $D_1 = \{\{$*Vancouver, Toronto*$\}, \{$*Montreal*$\}, \{$*Los Angeles*$\}, \{$*New York, St.Louis*$\}\}$, and $D_d = \{\{$*Vancouver*$\}, \{$*Toronto*$\}, \{$*Montreal*$\}, \{$*Los Angeles*$\}, \{$*New York*$\}, \{$*St. Louis*$\}\}$, then these partitions are described by the DGG shown in Figure 2.

3 Multi-Attribute Generalization

3.1 Basic Idea

Given the simple, single-path DGGs for attributes A, B, and C shown in Figure 3, Figure 4 shows the complete generalization state space for all possible combinations of nodes from the set of attributes. The objective of the Multi-Attribute Generalization algorithm is to visit each node in the generalization state space once, generating all possible summaries consistent with the DGGs for the set of attributes being generalized. The number of nodes in the generalization state space is $O(\prod_{i=1}^{m} |D_i|)$, where m is the number of attributes and $|D_i|$ is the number of nodes in the DGG for attribute i. Node $\langle A_d, B_d, C_d \rangle$, containing the least element from each of the individual DGGs in Figure 3, is called an *input* or *ungeneralized relation.* Its domain corresponds to the cross-product of the values contained in the individual attribute domains. All other nodes, called *generalized relations*, correspond to a different possible generalization of node $\langle A_d, B_d, C_d \rangle$. For example, assume that a, b_1 and b_2, and c describe the generalization relations for attributes A, B, and C, respectively. Applying a, b_1, b_2, and c, in the order specified, we obtain generalizations of $\langle A_d, B_d, C_d \rangle$ corresponding to nodes $\langle A_g, B_d, C_d \rangle$, $\langle A_g, B_1, C_d \rangle$, $\langle A_g, B_g, C_d \rangle$, and $\langle A_g, B_g, C_g \rangle$, respectively. Node $\langle A_g, B_g, C_g \rangle$, containing the greatest element from each of the individual DGGs in Figure 3, corresponds to the most general case where all attributes are generalized to ANY. Other generalizations can be obtained by applying different combinations of the generalization relations in a similar manner.

3.2 The Serial Algorithm

Given a relation R, a set of m DGGs, and a set of m attributes, where one DGG is associated with each attribute, the All_Gen algorithm, shown in Figure 5, generates all possible generalized relations consistent with the DGGs for

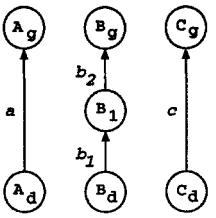

Fig. 3. A set of DGGs for attributes A, B, and C

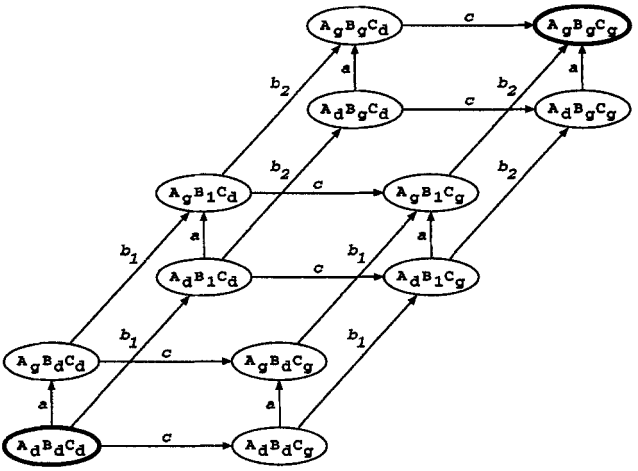

Fig. 4. Generalization state space for attributes A, B, and C

the set of attributes. In All_Gen, the function Node_Count (line 4) determines the number of nodes in DGG D_i. The function Generalize (line 9) returns a generalized relation where attribute i in the target relation has been generalized to the level of node D_{i_k} (that is, D_{i_k} is the k-th node of DGG D_i). Any of the generalization algorithms presented in [1, 2, 3, 4, 7] may be used to implement the Generalize function. The procedure Output (line 10) saves the generalized relation and combination of nodes from which the generalized relation was generated. The computational complexity of the serial algorithm is $O(n \prod_{i=1}^{m} |D_i|)$, where n is the number of tuples, m is the number of attributes, and $|D_i|$ is the number of nodes in the DGG for attribute i.

The initial call to All_Gen is All_Gen $(R, 1, m, D, D_{nodes})$, where R is the input relation for this discovery task, 1 is an identifier corresponding to the first attribute, m is an identifier corresponding to the last attribute, D is the set of m DGGs associated with the m attributes, and D_{nodes} is a vector in which the i-th element is initialized to D_{i_1} (we assume the first node in each D_i corresponds to the domain of D_i). D_{nodes} is used to store the combination of nodes from which each generalized relation is generated.

```
1.   procedure All_Gen (relation, i, m, D, D_nodes)
2.   begin
3.       work_relation ← relation
4.       for k = 1 to Node_Count (D_i) − 1 do begin
5.           if i < m then
6.               All_Gen (work_relation, i + 1, m, D, D_nodes)
7.           end
8.           D_nodes[i] ← D_{i_{k+1}}
9.           work_relation ← Generalize (relation, i, D_{i_{k+1}})
10.          Output (work_relation, D_nodes)
11.      end
12.  end
```

Fig. 5. Serial multi-attribute generalization algorithm

The algorithm is described as follows. In the i-th call to All_Gen (corresponding to the i-th attribute), one pass is made through the *for* loop (lines 4 to 11) for each non-domain node in D_i (i.e., the DGG associated with attribute i). If the i-th call to All_Gen is not also the m-th call (that is, corresponding to the last attribute) (line 5), then the $i+1$-th call to All_Gen is made (line 6). The $i+1$-th call to All_Gen is All_Gen ($work\_relation, i+1, m, D, D_{nodes}$), where the values of m, D, and D_{nodes} do not change from the i-th call. The first parameter, *work_relation*, was previously set to the value of *relation* prior to entering the *for* loop (line 3). The second parameter, i, is incremented by one (corresponding to the $i+1$-th attribute). In the first pass through the *for* loop (i.e., $k = 1$) for the i-th call, the value of *work_relation* is R (i.e., the original input relation).

In the m-th call to All_Gen, or when the $i+1$-th call returns control to the i-th call (line 6), the i-th call determines the next level of generalization for attribute i (i.e., $D_{i_{k+1}}$) and saves it in the i-th element of the vector D_{nodes} (line 8). The relation used as input to the i-th call to All_Gen is generalized to the level of node $D_{i_{k+1}}$ (line 9), and the resulting generalized relation is saved along with the combination of nodes from which the generalized relation was generated (line 10). In all passes through the *for* loop, other than the first (i.e., $k > 1$), the value of *work_relation* passed by the i-th call to the $i+1$-th call is *relation* generalized to the level of $D_{i_{k+1}}$.

3.3 The Parallel Algorithm

The size of the generalization state space depends only on the number of nodes in the DGGs; it is not dependent upon the number of tuples in the input relation. When the number of attributes to be generalized is large or the DGGs associated with a set of attributes is complex, we can improve the performance of the serial algorithm through parallel generalization. Our parallel algorithm does not simply assign one node in the generalization state space to each processor, because the startup cost for each processor was considered too great in comparison to the actual work performed. Through experimentation, we adopted a more coarse-grained approach, where a unique combinations of paths, including one

```
1.   procedure Par_All_Gen (relation, i, m, D, D_paths, D_nodes)
2.   begin
3.       for k = 1 to Path_Count (D_i) do begin
4.           D_paths[k] ← D_i^k
5.           if i < m then
6.               Par_All_Gen (relation, i + 1, m, D, D_paths, D_nodes)
7.           else
8.               fork All_Gen (relation, 1, m, D_paths, D_nodes)
9.           end
10.      end
11.  end
```

Fig. 6. Parallel multi-attribute generalization algorithm

path through the DGG for each attribute, was assigned to each processor. For example, given attribute A with three possible paths through its DGG, attribute B with 4, and attribute C with 2, our approach creates $3 \times 4 \times 2 = 24$ processes. The Par_All_Gen algorithm, shown in Figure 6, creates parallel All_Gen child processes on multiple processors (line 8). In Par_All_Gen, the function Path_Count (line 3) determines the number of paths in DGG D_i.

The initial call to Par_All_Gen is Par_All_Gen $(R, 1, m, D, \emptyset, D_{nodes})$, where R, 1, m, D, and D_{nodes} have the same meaning as in the serial algorithm, and $\emptyset$ initializes D_{paths}. D_{paths} is a vector in which the i-th element is assigned a unique path from D_i.

The algorithm is described as follows. In the i-th call to Par_All_Gen, one pass is made through the *for* loop (lines 3 to 10) for each distinct path in D_i. The current path, D_i^k, is determined and saved in the k-th element of D_{paths} (line 4), where D_i^k is the k-th path in D_i. If the i-th call to Par_All_Gen is not also the m-th call (line 5), then the $i+1$-th call to Par_All_Gen is made (line 6). The $i+1$-th call to Par_All_Gen is Par_All_Gen $(relation, i+1, m, D, D_{paths}, D_{nodes})$, where the values for *relation*, m, D, and D_{nodes} do not change from the i-th call. The second parameter is incremented by one. The fifth parameter, D_{paths}, was previously set to D_i^k (line 4). When the $i+1$-th call returns control to the i-th call (line 6), the next pass through the *for* loop begins (line 4).

In the m-th call to Par_All_Gen, an All_Gen child process is created (line 8). The call to All_Gen is All_Gen $(relation, 1, m, D_{paths}, D_{nodes})$, where *relation*, m, and D_{nodes} are unchanged from the values passed as parameters to the m-th call to Par_All_Gen. The second parameter, 1, is an identifier corresponding to the first attribute. The fourth parameter, D_{paths}, is a unique vector containing m paths from D_i (i.e., one from each DGG for the set of attributes). The All_Gen child process then follows the serial algorithm described in the previous section.

4 Experimental Results

We ran all of our experiments on a 64-node Alex AVX Series 2, a MIMD distributed memory parallel computer. Each inside-the-box compute node consists of a T805 processor, with 8 MB of local memory, paired with an i860 processor,

with 32 MB of shared memory (the pair communicates through the shared memory). Each i860 processor runs at 40 MHz and each T805 processor runs at 20 MHz with a bandwidth of 20 Mbits/second of bi-directional data throughput on each of its four links. The compute nodes run version 2.2.3 of the Alex-Trollius operating system. The front-end host computer system is a Sun Sparc 20 with 32 MB of memory, running version 2.4 of the Solaris operating system.

The Parallel Multi-Attribute Generalization algorithm has been implemented in C as an extension to DB-Discover, a software tool for knowledge discovery from databases [1, 3, 4]. The parallel implementation functions as three types of communicating modules: a *slave program* runs on an inside-the-box compute node and executes the discovery tasks that it is assigned, the *master program* assigns discovery tasks to the slave programs, and the *bridge program* coordinates access between the slave programs and the database.

The parallel algorithm may generalize the same combination of nodes in D_{nodes} on multiple processors. This can occur when a node in a DGG resides on more than one path. To prevent this would require prior analysis of the generalization state space or some form of communication and synchronization between processors, introducing additional overhead. For these experiments, we consider this redundant generalization to be tolerable because it only occurs in a small percentage of the total number of states in the generalization state space.

Input data was from a large database supplied by a commercial partner in the telecommunications industry. Queries read approximately 675,000 tuples from three tables which contained a cumulative total of 28 attributes. Our experience in applying data mining techniques to the databases of our commercial partners has shown that domain experts typically perform discovery tasks on a few attributes that have been determined to be relevant. Consequently, we present the results for experiments where two and three attributes were selected for generalization and the DGGs associated with the selected attributes contained from three to seven unique paths. The characteristics of the DGGs associated with each attribute are shown in Table 1, where the *No. of Paths* column describes the number of unique paths, the *No. of Nodes* column describes the number of nodes, and the *Avg. Path Length* column describes the average path length.

From these experiments, we draw three main conclusions. First, as the complexity of the DGGs associated with a set of attributes used in a discovery task increases (either by adding more paths or more nodes to paths), the complexity and traversal time of the generalization state space also increases. This was expected based upon the complexity analysis given in Section 3.2. Second, as the number of processors used in a discovery task increases, the time required to

Table 1. Characteristics of the DGGs for three attributes

Attribute	*No. of Paths*	*No. of Nodes*	*Avg. Path Length*
A	7	36	5.1
B	3	17	5.6
C	4	22	5.5

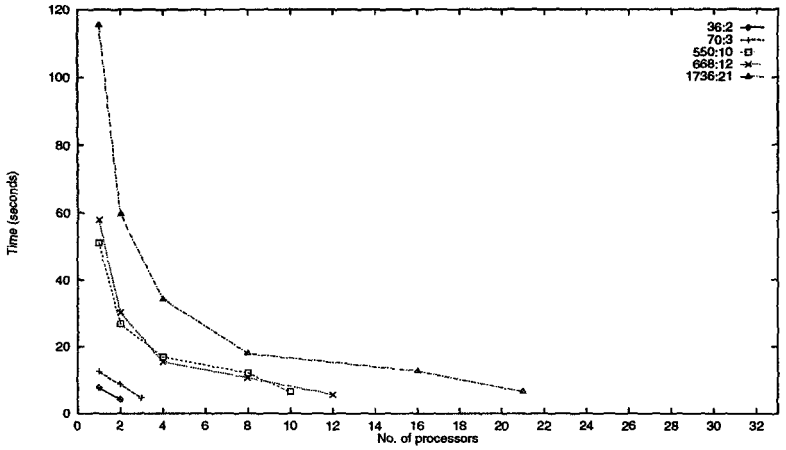

Fig. 7. Relative performance generalizing two attributes

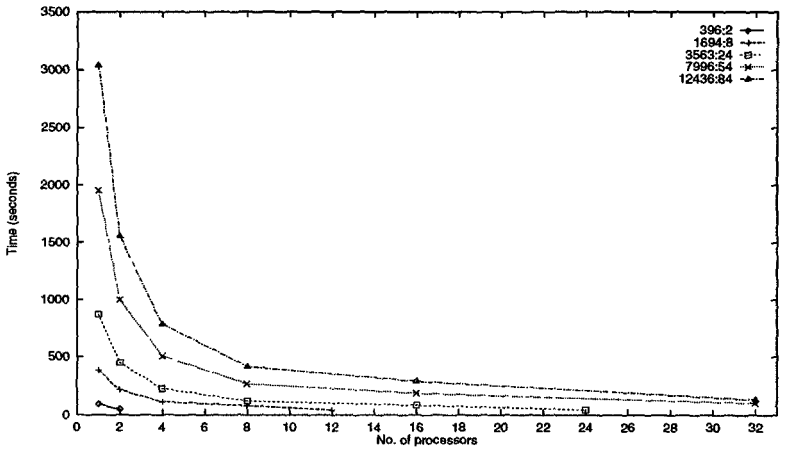

Fig. 8. Relative performance generalizing three attributes

traverse the generalization state space decreases. And third, significant speedups can be obtained using multiple processors. These results are shown in the graphs of Figures 7 and 8, where the number of processors is plotted against execution time. The legend for each curve in these graphs is of the form x:y, where x is the number of nodes in the generalization state space and y is the number of unique path combinations (i.e., the maximum number of processors).

In both experiments, we varied the number of paths through the DGGs for each attribute in the discovery task and the number of processors assigned to the discovery task. A maximum of 32 processors were available. The graphs show that as the complexity of the generalization state space increases, the time required to traverse the generalization state space also increases. For example, in

Table 2. Speedup results obtained using the parallel algorithm

Experiment	*Attributes Generalized*	*No. of Nodes in State Space*	*No. of Sub-Tasks*	*No. of Processors*	*Serial Time*	*Parallel Time*	*Speedup*
1	A,B	36	2	2	7.74	4.16	1.9
		70	3	3	12.48	4.59	2.7
		550	10	10	51.06	6.60	7.7
		668	12	12	57.86	5.72	10.1
		1736	21	21	115.44	6.60	17.5
2	A,B,C	396	2	2	89.32	47.15	1.9
		1694	12	12	380.18	42.13	9.0
		3553	24	24	867.87	44.92	19.3
		7996	54	32	1952.70	101.07	19.3
		12436	84	32	3037.55	131.86	23.0

Figure 7, showing the results of the two-attribute experiment, when running on a single processor, the time to generalize varies from 7.74 seconds for a generalization state space containing 36 nodes to 115.44 seconds for a generalization state space containing 1736 nodes. The less complex discovery tasks could not be partitioned usefully across 32 processors. For example, all the discovery tasks in Figure 7 and three in Figure 8 used fewer than 32 processors.

The graphs also show that as the number of processors assigned to a discovery task is increased, the time required to traverse the generalization state space decreases. Increasing the number of processors divides the discovery task into smaller discovery tasks (i.e., sub-tasks). For example, in Figure 8, showing the results of the three-attribute experiment, the time to generalize in a generalization state space containing 12436 nodes varies from 3037.55 seconds on one processor to 131.86 seconds on 32 processors. Both of the largest discovery tasks in Figure 8 used all 32 processors.

Speedups for the discovery tasks run in each experiment are shown in Table 2, where the *No. of Nodes in State Space* column is the number of nodes in the generalization state space, the *No. of Sub-Tasks* column is the number of unique path combinations from the set of DGGs, the *No. of Processors* column is the number of processors used, the *Serial Time* column is the time required to run the discovery task on one processor, the *Parallel Time* column is the time required on the actual number of processors used, and *Speedup* is the serial time divided by the parallel time. Significant speedups were obtained when a discovery task was run on multiple processors. For example, the speedups for the largest generalization state spaces were 17.5 on 21 processors and 23.0 on 32 processors for the first and second experiments, respectively.

5 Conclusion and Future Work

We presented the Parallel Multi-Attribute Generalization algorithm for parallel attribute-oriented induction. The algorithm generates all possible generalized relations from the DGGs associated with a set of attributes by partitioning path combinations from the DGGs across multiple processors. Increasing the

complexity of the DGGs associated with a set of attributes or increasing the number of attributes, increases the complexity of the generalization state space. We showed that increasing the number of processors is effective for significantly reducing the time required to traverse the generalization state space.

Future research will focus on ways to reduce the number of generalized relations generated. Preliminary experiments have shown variance, the most common measure of variability used in statistics, to be a useful measure for comparing the distribution defined by the structured tuples in a generalized relation to that of a uniform distribution of the tuples. Heuristics which use the variance can then be used to prune those of least "interest". For example, pruning all generalized relations except the one with the highest variance from each subtask has been shown to be effective. A complementary heuristic, which measures the complexity of generalized relations, can be used to break ties.

References

1. C. L. Carter and H. J. Hamilton. Efficient attribute-oriented algorithms for knowledge discovery from large databases. To appear in IEEE Trans. on Knowledge and Data Engineering.
2. C. L. Carter and H. J. Hamilton. Fast, incremental generalization and regeneralization for knowledge discovery from databases. In *Proceedings of the 8th Florida Artificial Intelligence Symposium*, pages 319–323, Melbourne, Florida, April 1995.
3. C. L. Carter and H. J. Hamilton. A fast, on-line generalization algorithm for knowledge discovery. *Applied Mathematics Letters*, 8(2):5–11, 1995.
4. C. L. Carter and H. J. Hamilton. Performance evaluation of attribute-oriented algorithms for knowledge discovery from databases. In *Proceedings of the Seventh IEEE International Conference on Tools with Artificial Intelligence (ICTAI'95)*, pages 486–489, Washington, D.C., November 1995.
5. 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.
6. H.J. Hamilton and D.F. Fudger. Measuring the potential for knowledge discovery in databases with DBLearn. *Computational Intelligence*, 11(2):280–296, 1995.
7. J. Han. Towards efficient induction mechanisms in database systems. *Theoretical Computer Science*, 133:361–385, October 1994.
8. J. Han, Y. Cai, and N. Cercone. Knowledge discovery in databases: an attribute-oriented approach. In *Proceedings of the 18th International Conference on Very Large Data Bases*, pages 547–559, Vancouver, August 1992.
9. J. Han, Y. Cai, and N. Cercone. Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. on Knowledge and Data Engineering*, 5(1):29–40, February 1993.
10. H.-Y. Hwang and W.-C. Fu. Efficient algorithms for attribute-oriented induction. In *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining (KDD-95)*, pages 168–173, Montreal, August 1995.
11. 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.