

# Graph-Based Data Mining

Diane J. Cook and Lawrence B. Holder, University of Texas at Arlington

**T**HE LARGE AMOUNT OF DATA collected today is quickly overwhelming researchers' abilities to interpret the data and discover interesting patterns in it. In response to this problem, researchers have developed techniques and systems for discovering concepts in databases.<sup>1-3</sup> Much of the collected data, however, has an explicit or implicit structural component (spatial or temporal), which few discovery systems are designed to handle.<sup>4</sup> So, in addition to the need to accelerate data mining of large databases, there is an urgent need to develop scalable tools for discovering concepts in structural databases.

One method for discovering knowledge in structural data is the identification of common substructures within the data. Substructure discovery is the process of identifying concepts describing interesting and repetitive substructures within structural data. The discovered substructure concepts allow abstraction from the detailed data structure and provide relevant attributes for interpreting the data.

The substructure discovery method is the basis of Subdue, which performs data mining on databases represented as graphs. The system performs two key data-mining techniques: unsupervised pattern discovery and supervised concept learning from examples. Our test applications have demonstrated the scalability and effectiveness of these techniques on a variety of structural databases.

*USING DATABASES REPRESENTED AS GRAPHS, THE SUBDUE SYSTEM PERFORMS TWO KEY DATA-MINING TECHNIQUES: UNSUPERVISED PATTERN DISCOVERY AND SUPERVISED CONCEPT LEARNING FROM EXAMPLES. APPLICATIONS TO LARGE STRUCTURAL DATABASES DEMONSTRATE SUBDUE'S SCALABILITY AND EFFECTIVENESS.*

## Unsupervised concept discovery

Subdue discovers substructures that compress the original data and represent structural concepts in the data. The substructure discovery system represents structural data as a labeled graph. Objects in the data map to vertices or small subgraphs in the graph, and relationships between objects map to directed or undirected edges in the graph. A *substructure* is a connected subgraph in the graphical representation. An *instance* of a substructure is a set of vertices and edges from the input graph that match—graph theoretically—the substructure's graphical representation. This graphical representation serves as input to the substructure discovery system. Figure 1 shows a geometric example of a database. It also shows the graph representation of the discovered substructure and highlights one of the four instances of the substructure.

Subdue's substructure discovery algorithm is a beam search. Figure 2 shows the algorithm. The first step initializes ParentList (substructures to be expanded), ChildList (substructures that have been expanded), and BestList (the highest-valued substructures found so far) as empty. It also sets Processed-Subs (the number of substructures expanded so far) to 0. Each list is a linked list of substructures, sorted in nonincreasing order by substructure value. For each unique vertex label, Subdue assembles a substructure whose definition is a vertex with that label and whose instances are all the vertices in input graph  $G$  with that label. Each substructure is inserted in ParentList.

The inner `while` loop is the algorithm's core. It removes each substructure in turn from the head of ParentList, and extends each of the substructure's instances in all possible ways. It does this either by adding a new edge and vertex in  $G$  to the instance or by adding a new edge between two vertices if both ver-



## Data Mining: A Long-Term Dream Continues

In this continuation of our special issue on data mining, we present four articles that deal with mining nontraditional forms of data. Two articles deal with text mining. Sholom Weiss, Brian White, Chidanand Apte, and Fredrick Damerau show that simple and fast document matching can effectively assist help-desk applications in matching problem descriptions to relevant stored solution descriptions. Kurt Bollaker, Steve Lawrence, and C. Lee Giles present their Web-based CiteSeer system that finds user-specific scientific documents in the documents that are preprocessed and stored in a local database. The system learns the user profile from the interactions and uses it as an agent to monitor new documents that might interest the user.

Two articles describe new techniques. Neal Lesh, Mohammed Zaki, and Mitsunori Ogihara find frequent subsequence patterns that help classify a sequence, such as DNA, into different classes of sequences. Their Feature-Mine system efficiently examines subsequences to select a drastically pruned feature set. The article by Diane Cook and Lawrence Holder deals with data mining graph-structured data, such as CAD diagrams and chemical structures. Their Subdue system uses a beam search to discover substructures (features) that efficiently describe the given set of structures based on the MDL principle. Both of these new techniques discover useful features from the source data that are not the flat tables typically found in popular databases, which discriminate or describe such data.

### The earlier issue

In the first part of this two-part special issue on data mining that ran in our November–December 1999 issue, Gregory Piatetsky-Shapiro described the growing community of data mining in his “Expert Opinion” column. Data mining is becoming an indispensable decision-making tool in the ever more competitive business world. And challenging applications inspire new techniques and affirm their utility.

Two articles by Simon Kasif and Steven Salzberg discussed data mining

in the exciting field of computational biology, where frequent pattern discovery, clustering, and classification all play crucial roles in understanding protein structures and their functions.

Several articles represented innovative data-mining applications in more traditional domains with conceptually flat data tables. Chidanand Apte, Edna Grossman, Edwin Pednault, Barry Rosen, Fateh Tipu, and Brian White have developed a specialized probabilistic model for auto insurance pure premium—that is, the expected claim amount for each policy holder that meets the strict actuarial requirement. Special attention is given to missing values and the model’s scalability. Sylvain Létourneau, Fazel Famili, and Stan Matwin described the entire process of modeling failing aircraft components, from gathering data and generating serious models to evaluating them using a domain-specific scoring function. Extensive experiments seem to show that the nearest-neighbor method does best for most parts. Philip Chan, Wei Fan, Andreas Prodromidis, and Salvatore Stolfo apply a specialized boosting technique to credit card fraud detection for scalability and enhanced utility of the model. Cost savings thus achieved demonstrate the improved accuracy of multiple models while providing a scalable fast model generation on a very large data set.

—David Waltz and Se June Hong, Guest Editors

**David Waltz** has been vice president of the NEC Research Institute’s Computer Science Research Division and adjunct professor of computer science at Brandeis University since 1993, and will become president of the NEC Research Institute in April 2000. Contact him at NEC Research Institute, 4 Independence Way, Princeton, NJ 08540; waltz@research.nj.nec.com.

**Se June Hong** is a research staff member working on data-mining technology at the IBM T.J. Watson Research Center in Yorktown Heights, New York. His research interests have included error-correcting code, fault-tolerant computing, design automation, and knowledge-based systems. Contact him at IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598; hong@us.ibm.com.



articles are already part of the instance. The first instance of each unique expansion becomes a definition for a new child substructure. All the child instances that were expanded in the same way (by adding the same new edge or new edge with new vertex to the same old vertex) become instances of that child substructure. In addition, child instances generated by different expansions that match the child substructure definition within the match-cost threshold (described later) become instances of the child substructure.

Subdue then evaluates each child, using the minimum description length (MDL) heuristic, and inserts each child in ChildList in order of its heuristic value. The algorithm enforces the search’s beam width by controlling ChildList’s length. After inserting a

new child in ChildList, if ChildList’s length exceeds BeamWidth, the system destroys the substructure at the end of the list. The parent substructure is inserted in BestList, and the

same pruning mechanism limits BestList to be no longer than MaxBest. When ParentList is empty, the algorithm switches ParentList and ChildList, so that ParentList now holds

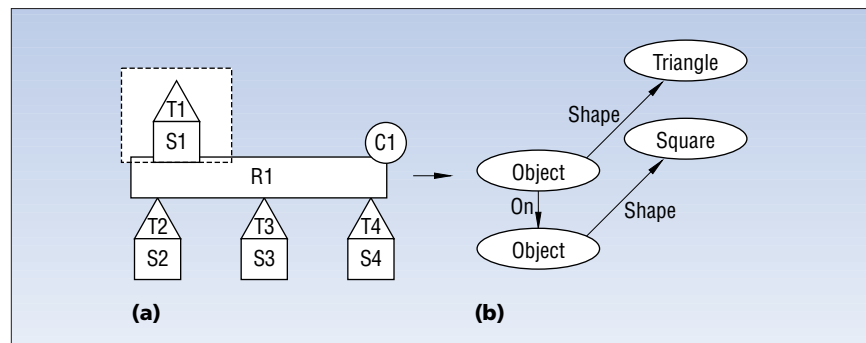


Figure 1. Example substructure in graph form: (a) input graph; (b) substructure.

```

Subdue(Graph, BeamWidth, MaxBest, MaxSubSize, Limit)
  ParentList = {}
  ChildList = {}
  BestList = {}
  ProcessedSubs = 0
  Create a substructure from each unique vertex label and
  its single-vertex instances; insert the resulting
  substructures in ParentList
  while ProcessedSubs <= Limit and ParentList is not empty do
    while ParentList is not empty do
      Parent = RemoveHead(ParentList)
      Extend each instance of Parent in all possible ways
      Group the extended instances into Child substructures
      foreach Child do
        if SizeOf(Child) <= MaxSubSize then
          Evaluate the Child
          Insert Child in ChildList in order by value
          if Length(ChildList) > BeamWidth then
            Destroy the substructure at the end of ChildList
        ProcessedSubs = ProcessedSubs + 1
        Insert Parent in BestList in order by value
        if Length(BestList) > MaxBest then
          Destroy the substructure at the end of BestList
          Switch ParentList and ChildList
  return BestList

```

Figure 2. Subdue's discovery algorithm.

the next generation of substructures to be expanded.

The BeamWidth and Limit (a user-defined limit on the number of substructures to process) parameters, along with computational constraints on the inexact graph match algorithm, constrain Subdue to a polynomial running time.

Another user-specified parameter, MaxSubSize, determines whether the algorithm prunes the discovery search space by discarding a child substructure whose heuristic value is not greater than its parent's heuristic value. Early in the discovery process, the number of a given substructure's instances is very large, and it dominates the value of any heuristic that uses the number of instances as a parameter. As the substructure grows, its number of instances decreases quickly because the substructure is becoming more specific. This means that the heuristic value also usually decreases early in the discovery process. If it decreases for a sufficiently long period, all the child substructures will be discarded, ParentList will empty, and discovery will halt. Disabling pruning during discovery keeps ChildList full, even for substructure values that do not increase monotonically as the substructure definition grows.

Because ParentList never empties, another method for halting the program is needed. If the user specifies a maximum substructure size (MaxSubSize), larger child substructures will not be inserted in ChildList. As a result, ParentList and ChildList will eventually empty, and the discovery algorithm will halt.

Once a substructure is discovered, Subdue uses it to simplify the data by replacing instances of the substructure with a pointer to the newly discovered substructure. Discovered substructures allow abstraction from detailed structures in the original data. Through iteration of the substructure discovery and replacement process, Subdue constructs a hierarchical description of the structural data in terms of the discovered substructures. This hierarchy provides varying interpretation levels that can be accessed on the basis of the data analysis's specific goals.

**The MDL heuristic.** Subdue's heuristic for evaluating substructures is based on the MDL principle: The best theory for describing a data set is the theory that minimizes the data set's description length.<sup>5</sup> The model for description length calculation is a local computer sending the description of a concept to a remote computer. The local computer

encodes the concept as a bit string, and the remote computer decodes the string to restore the original concept. The concept's description length is the number of bits in the string.

Subdue implements the MDL principle in the context of graph compression using a substructure. Thus, the best substructure in a graph is one that minimizes  $DL(S) + DL(G|S)$ . Here  $S$  is the discovered substructure,  $G$  is the input graph, and  $DL(S)$  is the number of bits required to encode the discovered substructure.  $DL(G|S)$  is the number of bits required to encode  $G$  after it has been compressed using  $S$ . An earlier article describes the exact graph-description-length computation used in Subdue.<sup>6</sup>

**Inexact graph match.** Because substructure instances can appear in different forms throughout the database, Subdue uses an inexact graph match to identify them.<sup>7</sup> In this approach, the user assigns a cost to each distortion of a graph. A distortion consists of basic transformations such as deletion, insertion, and substitution of vertices and edges. In determining the distortion costs, the user can bias the match for or against particular types of distortions.

Given graphs  $g_1$  with  $n$  vertices and  $g_2$  with  $m$  vertices,  $m$  being greater than or equal to  $n$ , the complexity of the full inexact graph match is on the order of  $n^{m+1}$ . Because the discovery process uses this routine heavily, its complexity can significantly degrade system performance. To improve the algorithm's performance, we have it search through the space of possible partial mappings using a uniform cost search. The cost from the root of the tree to a given node is calculated as the cost of all distortions corresponding to that node's partial mapping. The algorithm considers vertices from the matched graphs in order, from the most heavily connected to the least connected. Because the uniform cost search guarantees an optimal solution, it ends as soon as it finds the first complete mapping.

In addition, the user can limit the number of search nodes (defined as a function of the number of vertices in the first graph) considered by the branch-and-bound procedure. When the number of nodes expanded in the search tree reaches the defined limit, the search resorts to hill climbing, using the mapping cost so far as the measure for choosing the best node at a given level. Our earlier article provides a complete description of Subdue's polynomial inexact graph match.<sup>6</sup>

Bounds on the number of substructures

considered ( $L$ ) and the number of partial mappings considered during an inexact graph match ( $g$ ) constrain Subdue to run in polynomial time. The system's worst-case runtime is the product of the number of generated substructures, the number of instances of each substructure, and the number of partial mappings considered during the graph match. This is expressed

$$\left(\sum_{i=1}^L i * ((v-1) - (i-1))\right) * (v(L-1)) * g$$

where  $v$  represents the number of vertices in the input graph. An earlier article provides the derivation of this expression.<sup>7</sup>

## Discovery system applications

We have successfully applied Subdue, with and without domain knowledge, to databases in domains including image analysis, CAD circuit analysis, Chinese characters, program source code, and chemical reaction chains.<sup>7,8</sup>

**CAD circuit analysis.** Figure 3 shows the substructures Subdue discovered in a CAD circuit representing a sixth-order bandpass leapfrog ladder (boxed substructures were discovered in previous iterations). The figure also tabulates an evaluation based on compression obtained with the substructure, time required to process the database, a human rating of the discovered concept's interestingness, and the number of substructure instances found in the database. For the interestingness rating, eight domain experts rated each substructure on a scale from 1 to 5, with 1 meaning the substructure does not represent useful CAD information and 5 meaning the substructure is very useful.

In this experiment, Subdue generated substructures in three ways: using no background knowledge, using background knowledge in the form of graph match rules customized for this domain, and using both graph match rules and information about specific models likely to occur in this domain. As the figure shows, Subdue discovered substructures that perform well at compressing the database and represent functional CAD concepts. Using general background knowledge improved the concept's functional rating. But the MDL principal alone was effective in discovering the operational amplifier substructure, which the experts determined is highly interesting and functional in this domain.

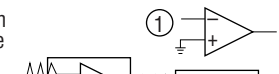
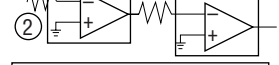
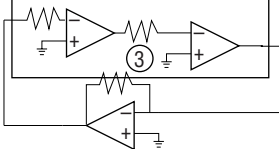
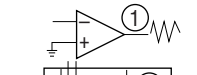
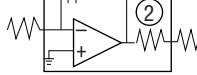
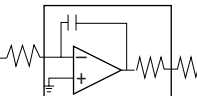
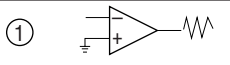
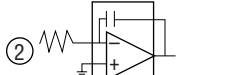
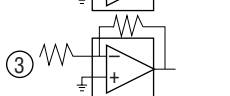
Usage of domain knowledge	Discovered substructures	Compression	Nodes expanded	Human rating [std. dev.]	Instances
No domain knowledge		0.63	571,370	4.2 [1.2]	9
		0.68	46,422	2.7 [1.2]	3
		0.72	59,886	2.7 [1.0]	2
Graph match rules		0.43	145,175	2.7 [1.7]	6
		0.51	39,881	2.7 [1.0]	3
		0.62	425,772	1.5 [0.8]	2
Model knowledge and graph match rules		0.53	24,273	4.3 [1.2]	9
		0.66	12,960	4.5 [0.8]	4
		0.72	7,432	4.5 [0.8]	2

Figure 3. Discovered substructures in a leapfrog circuit (boxed substructures were discovered in previous iterations).

**Protein structure analysis.** More recently, we applied Subdue to several large databases containing data requiring scientific interpretation. For example, we applied the unsupervised-discovery system to the July 1997 release of the Protein Data Bank (PDB). Our goal was to identify structural patterns in the primary, secondary, and tertiary structures of three protein categories: hemoglobin, myoglobin, and ribonuclease A. These patterns would act as signatures distinguishing proteins in the category from other types of proteins and providing a classification mechanism.

Using Subdue, we converted primary structure information from the primary DNA sequence specified in each PDB file by representing each amino acid in the sequence as a graph vertex. The vertex numbers increase in the sequence order from N-terminus to C-terminus, and the vertex label is the name of the amino acid. We added an edge labeled "bond" between adjacent amino acids in a sequence.

We extracted secondary structure by listing occurrences of helices and strands along the primary sequence. A graph vertex labeled "h"

followed by the helix type and length represents each helix. A graph vertex labeled "s" followed by the strand's orientation and length represents each strand. Edges between two consecutive vertices are labeled "sh" if they belong to the same PDB file. To represent the protein's 3D features, we used the  $x$ ,  $y$ , and  $z$  coordinates of each atom in the protein. We represented each amino acid  $\alpha$ -carbon as a graph vertex. If the distance between two  $\alpha$ -carbons was greater than 6 Å, we discarded the information. Otherwise, we created edges between two  $\alpha$ -carbons and labeled them "vs" (very short, distance  $\leq 4$  Å) or "s" (short).

Subdue indeed found a structural pattern for each protein category. Using primary structure information, it identified patterns unique to each class of protein but occurring in 63 of the 65 hemoglobin cases, 67 of the 103 myoglobin cases, and 59 of the 68 ribonuclease A cases.

Figure 4 summarizes one of the findings for hemoglobin secondary structure, presenting an overall view of the protein, the portion where the discovered pattern exists, and schematic views of the best pattern. The

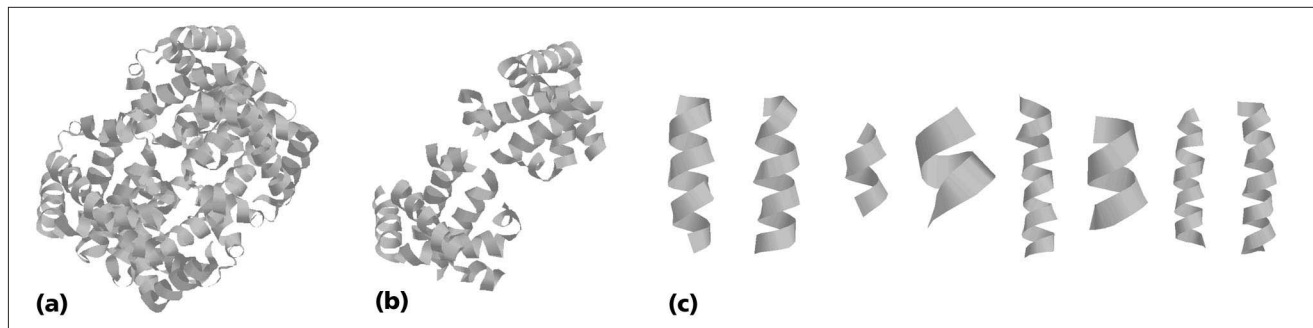


Figure 4. Hemoglobin secondary structure: (a) overall view, (b) discovered pattern, and (c) schematic views of the pattern.

patterns discovered for each sample category covered a majority of proteins in that category. That is, 33 of the 50 analyzed hemo-

globin proteins, 67 of the 89 myoglobin proteins, and 35 of the 52 ribonuclease A proteins contained the discovered patterns.

There are many possible reasons that some proteins did not show a pattern. Many factors can affect a protein's structure: sample quality, experimental conditions, and human error. Discrepancies can also result from physiological and biochemical factors. The same protein molecule's structure might differ from one species to another. The protein might be genetically defective. For example, sickle-cell anemia is the classic example of a genetic hemoglobin disease in which the protein lacks the structure necessary to perform its normal function.

We mapped the secondary structural patterns of the hemoglobin, myoglobin, and ribonuclease A proteins back into the PDB files. After mapping, we found that one discovered hemoglobin pattern belongs to the  $\alpha$  chains and the other to the  $\beta$  chains of a hemoglobin molecule (a hemoglobin molecule contains two  $\alpha$  and two  $\beta$  chains). The discovered myoglobin pattern appears in a majority of the myoglobin proteins in the data set. Finally, upon mapping the discovered ribonuclease A patterns back to the PDB files, we observed that several ribonuclease S proteins have the same patterns as those in ribonuclease A proteins. This is consistent with the fact that ribonuclease S is a complex consisting of two fragments (S-peptide and S-protein) of the ribonuclease A proteins. The pattern in the ribonuclease S comes from the S-protein fragment.

The secondary-structure patterns discovered are also distinct to each protein category. Subdue searched the global data set to identify the possible existence of the discovered pattern from each protein category. The results indicated that there is no exact match of the best patterns of one category in other categories.

Steve Sprang, a molecular biologist at the University of Texas Southwestern Medical Center, evaluated the patterns discovered by the Subdue system. He reviewed the original database and the discovered substructures to determine whether the discovered concepts represented the data accurately and pointed

## Related work

Researchers have proposed a variety of unsupervised-discovery approaches for structural data.<sup>1,2</sup> One approach is to use a knowledge base of concepts to classify the structural data. Systems using this approach learn concepts from examples and then categorize observed data. Such systems represent examples as distinct objects and process individual objects one at a time. In contrast, Subdue stores the entire database (with embedded objects) as one graph and processes the graph as a whole.

Scientific discovery systems that use domain knowledge have also been developed, but they target a single application domain. An example is Mechem,<sup>3</sup> which relies on domain knowledge to discover chemistry hypotheses. In contrast, Subdue performs general-purpose, automated discovery with or without domain knowledge and hence can be applied to many structural domains.

Logic-based systems have dominated relational concept learning, especially inductive logic programming (ILP) systems. However, first-order logic can also be represented as a graph and, in fact, is a subset of what graphs can represent. Therefore, learning systems using graphical representations potentially can learn richer concepts if they can handle the larger hypothesis space.

FOIL,<sup>4</sup> the ILP system discussed in this article, executes a top-down approach to learning relational concepts (theories) represented as an ordered sequence of function-free definite clauses. Given extensional background knowledge including relations and examples of the target concept relation, FOIL begins with the most general theory. Then it follows a set-covering approach, repeatedly adding a clause that covers some positive examples and few negative examples. Then, FOIL removes the positive examples covered by the clause and iterates the process on the reduced set of positive examples and all negative examples until the theory covers all the positive examples. To avoid overcomplex clauses, FOIL ensures that a clause's description length does not exceed the description length of the examples the clause covers. In addition to the applications discussed here, as well as applications in numerous recursive and nonrecursive logical domains, FOIL has been applied to learning search-control rules and patterns in hypertext.

## References

1. D. Conklin, "Machine Discovery of Protein Motifs," *Machine Learning*, Vol. 21, Nos. 1/2, 1995, pp. 125–150.
2. K. Thompson and P. Langley, "Concept Formation in Structured Domains," *Concept Formation: Knowledge and Experience in Unsupervised Learning*, D.H. Fisher and M. Paz-zani, eds., Morgan Kaufmann, San Mateo, Calif., 1991.
3. R.E. Valdes-Perez, "Conjecturing Hidden Entities by Means of Simplicity and Conservation Laws: Machine Discovery in Chemistry," *Artificial Intelligence*, Vol. 65, No. 2, 1994, pp. 247–280.
4. R.M. Cameron-Jones and J.R. Quinlan, "Efficient Top-Down Induction of Logic Programs," *SIGART Bulletin*, Vol. 5, No. 1, 1994, pp. 33–42.

to interesting discoveries. Sprang found that Subdue discovered an interesting, previously unknown pattern that suggests new information about the microevolution of such proteins in mammals.

We are continuing experimental applications of Subdue's unsupervised-discovery capabilities in the domains of biochemistry, geology, program source code, and aviation data.

## Scalability

A barrier to integrating scientific discovery into practical data-mining approaches is that discovery systems lack scalability. Many system developers evaluate a discovery method's correctness without regard to its scalability. Another barrier is that some scientific-discovery systems deal with rich data representations that degrade scalability. For example, Subdue's discovery relies on computationally expensive procedures such as subgraph isomorphism. Although Subdue's algorithm is polynomially constrained, the system still spends a considerable amount of computation on this task.

We are researching the use of distributed hardware to improve Subdue's scalability. In our approach, we partition the data among  $n$  individual processors and process each partition in parallel.<sup>9</sup> Each processor performs the sequential version of Subdue on its local graph partition and broadcasts its best substructures to the other processors. The processors then evaluate the communicated substructures on their local partitions. Once all evaluations are complete, a master processor gathers the results and determines the global best discoveries.

Sequential Subdue's runtime is nonlinear with respect to the graph's size. So decreasing the input's size by partitioning the graph among multiple processors sometimes results in a speedup greater than the number of processors. However, the serial algorithm analyzes the entire graph and therefore does not overlook important relationships of its parts. Partitioning the graph among processors might remove essential information (in our case, edges along boundary lines), and neighboring information can no longer be used to discover concepts.

The Metis graph-partitioning algorithm (<http://www-users.cs.umn.edu/~karypis/metis>) lets us divide the input graph into  $n$  partitions in a way that minimizes the number of edges shared by partitions and thus reduces

information loss. We modified this algorithm to allow a small amount of overlap between partitions, which recovers some of the information lost at the partition boundaries.

Figure 5 graphs the runtime of distributed Subdue on two classes of graphs as the number of processors increases. The two classes are

- graphs representing CAD circuits (each labeled " $n$ CAD," where  $n$  represents the size factor, and 1CAD contains 8,441 ver-

tices and 19,206 edges) and

- graphs generated artificially (each labeled " $n$ ART," where  $n$  represents the size factor, and 1ART contains 2,000 vertices and 5,000 edges).

As predicted, the speedup is usually close to linear and sometimes greater than the number of processors. Increasing the number of partitions results in improved speedup until the number of partitions approaches the number of vertices in the graph.

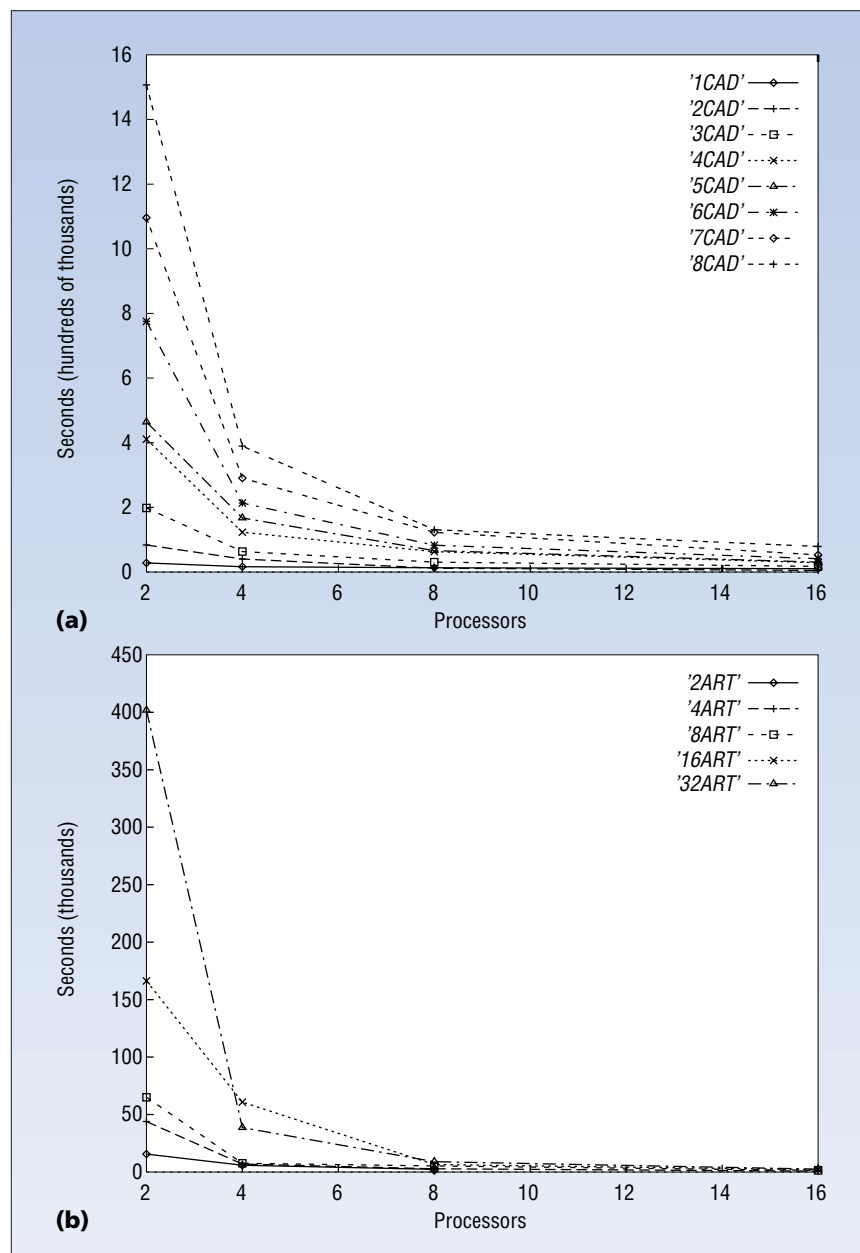
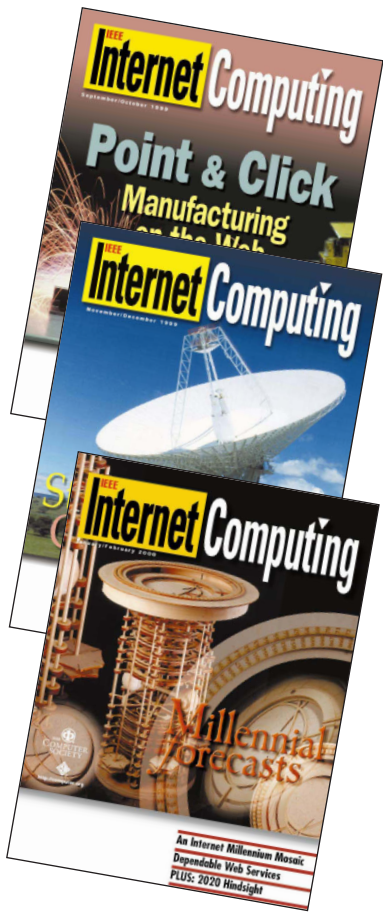


Figure 5. Distributed Subdue runtime on (a) CAD and (b) ART graphs.



IC and IC Online publish the latest developments in Internet-based applications and supporting technologies, and address the Internet's widening impact on engineering practice and society.

In 2000, we'll look at:

- Agent technologies
- Internet-based workflow
- Internet QoS
- Knowledge networking
- Widely deployed security solutions ... and more

Check us out!

**Internet Computing**

<http://computer.org/internet/>

IEEE  
COMPUTER  
SOCIETY

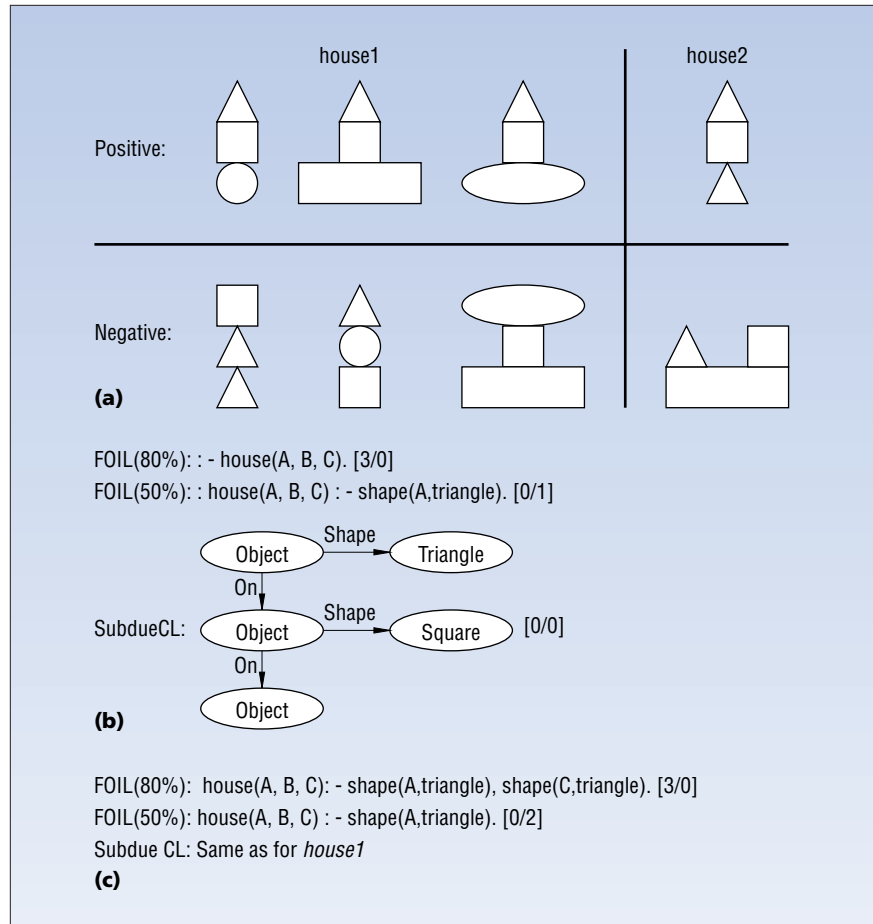


Figure 6. (a) The house domain; (b) SubdueCL and FOIL results for house1; (c) results for house2.

Because each processor handles only a portion of the overall database, some degradation in the quality of discovered substructures might result from this parallel version of Subdue. In practice, we find that as the number of processors increases, quality measured as compression initially improves because a greater number of substructures can be considered in the allotted time. As the number of partitions approaches the number of vertices in the graph, the quality of discovered substructures degrades.

By partitioning the database effectively, distributed Subdue proves to be a highly scalable system. One of our tested databases, representing a satellite image, contains 2 million vertices and 5 million edges. It could not be effectively processed on one machine given the memory constraints. But distributed Subdue processed the database in less than three hours using eight processors. Requiring a minimal amount of communication and using PVM (Parallel Virtual Machine) for communication, distributed Subdue is available for widespread use and runs on a variety of heterogeneous distributed networks.

## Supervised concept learning

We have extended Subdue to act not only as an unsupervised-discovery system, but also to perform supervised, graph-based, relational concept learning. Few general-purpose learning methods use a formal graph representation of knowledge, perhaps because of a graph's arbitrary expressiveness. Another reason is the inherent NP-hardness of typical learning routines, such as covers and least-general generalization, which both require subgraph isomorphism tests in the general case. Pattern-recognition-learning methods in chemical domains have been successful because of the natural graphical description of chemical compounds, but no domain-independent concept-learning systems use a graph representation.

Our main challenge in adding a concept-learning capability to the graph-based discovery system was including a negative graph in the process. Substructures that occur often in the positive graph but infrequently in the negative graph are likely to represent the target concept. Therefore, the Subdue concept learner (SubdueCL) accepts both a

positive and a negative graph and evaluates substructures as to their compression of the positive and lack of compression of the negative. SubdueCL searches for the substructure  $S$  that minimizes the cost of describing substructure  $S$  in positive graph  $G_p$  and negative graph  $G_n$ . This cost is expressed

$$value(G_p, G_n, S) = DL(G_p, S) + DL(S) + DL(G_n) - DL(G_n, S).$$

$DL(G, S)$  is the description length, according to the MDL encoding, of graph  $G$  after being compressed using substructure  $S$ , and  $DL(G)$  is the description length of graph  $G$ . This cost represents the information needed to represent  $G_p$  using substructure  $S$ , plus the information needed to represent the portion of  $G_n$  that was compressed using  $S$ . Thus, SubdueCL prefers substructures that compress the positive graph but not the negative graph.

Another challenge was the discovery system's bias toward finding only one good substructure in the entire input graph. Inductive logic programming (ILP) systems have a distinct advantage over Subdue; they typically find theories composed of many rules, whereas Subdue finds essentially one rule. Subdue's iterative, hierarchical capabilities somewhat address this problem. But the substructures found in later iterations are typically defined in terms of previously discovered substructures and are therefore only specializations of the earlier, more general rule. To avoid this tendency, SubdueCL discards any substructure that contains substructures discovered during previous iterations. SubdueCL iterates until it can find no substructure that compresses the positive graph more than the negative graph.

## Comparing learning systems

We compared SubdueCL to the ILP system FOIL (First-Order Inductive Learning)<sup>10</sup> and to the decision-tree induction system C4.5.<sup>3</sup> First, we compared the relational learners SubdueCL and FOIL on a simple artificial domain to examine qualitative differences. Then we compared all three systems on relational domains.

**An artificial domain.** Figure 6a depicts the two data sets of the *house* domain, *house1* and *house2*, whose target concept is "triangle on square." *House1* contains the six examples on the left, and *house2* contains the two additional examples on the right. Each object has a shape and is related to other objects by the binary relation *on*. Figure 6b shows the results of running SubdueCL and FOIL on *house1*.

The bracketed numbers in Figure 6b are the number of false negatives and false positives. We ran FOIL twice on each data set, once with the default minimum individual-clause-accuracy of 80%, and once with a minimum of 50%. At 80%, FOIL returned the concept that nothing was a house, misclassifying the three positive examples. At 50%, FOIL described houses as any example with a triangle on top. However, to relate the triangle and square objects in a connected substructure, SubdueCL includes the *on* relation. Therefore, SubdueCL recognized that the triangle should be on top of the square.

The *house2* data set adds two examples to *house1* to emphasize the need for the *on* relation. Figure 6c shows the results of running the two systems on *house2*. Again, FOIL has trouble identifying the correct concept.

These examples reveal an advantage of graph representation over logic for such posi-

tion-invariant (and other relational) concepts. However, logic is better suited for some concepts. For example, FOIL can easily represent the concept "top and bottom objects have same shape" as follows:

```
house(A, B, C) :- shape(A, S),
                 shape(C, S). or
house(A, B, C) :- shape(A, S1),
                 shape(C, S2), S1 = S2.
```

In contrast, using the graph representation defined for the *house* domain, SubdueCL would need to learn five different substructures, one for each shape. Of course, if we could foresee such a concept, we could change the graph representation to include an intermediate *shape* vertex connected to another vertex with the actual shape. Then, we could add *equal* edges between these *shape* vertices to represent equivalent shapes independent of the actual shape. Furthermore, although both systems handle numeric values (vertex labels), a similar representational transformation is necessary for learning concepts involving general equalities and inequalities between numbers.

**Relational domains.** The relational domains we used in our tests included illegal chess endgames, tic-tac-toe endgames, and musical excerpts. The chess domain consists of 12,957 row-column positions for a white king, white rook, and black king such that the black king is (positive) or is not (negative) in check. FOIL extensionally defined adjacency relations between chessboard positions and less-than relations between row and column numbers. We provided C4.5 with the same information by adding features that relate each piece's row and col-

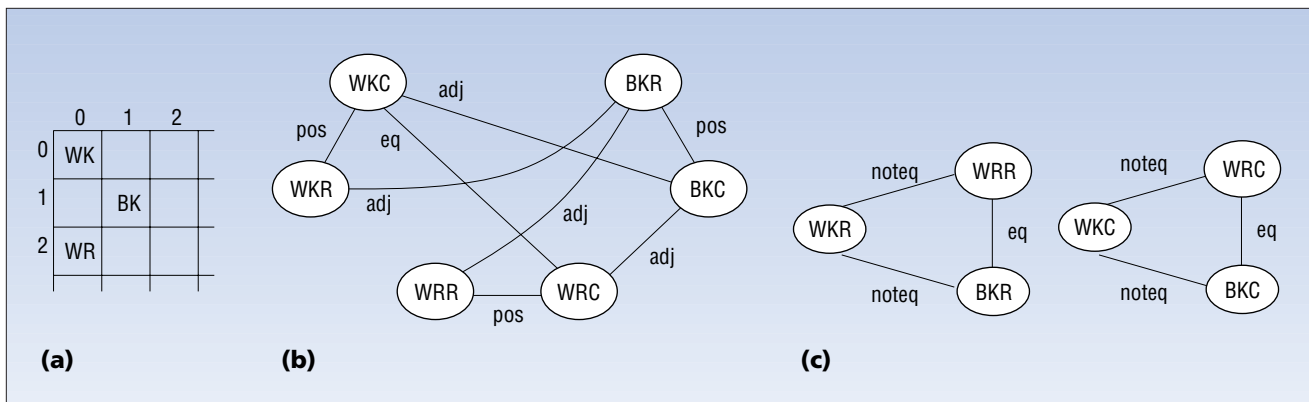


Figure 7. (a) A chess domain example; SubdueCL's (b) graphical representation and (c) discovered substructures for the example.

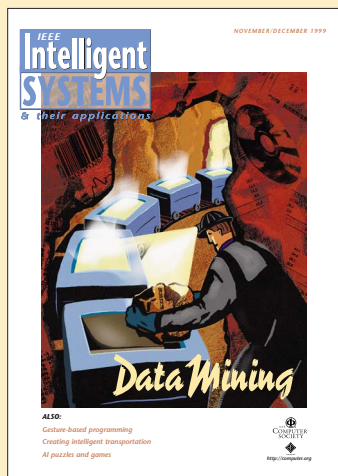


# Coming Next Issue

May/June

## Knowledge Management and the Internet

Guest Editor:  
Rose Dieng



Knowledge management is a key progress factor in organizations. It involves explicit and persistent representation of knowledge of dispersed groups of people in the organization, to improve the organization's activities. When the organization's knowledge is distributed among several experts and documents located all over the world, the Internet or an intranet and World Wide Web techniques can be the means for the acquisition, modeling, and management of this distributed knowledge. This issue will cover issues related to knowledge management, corporate memory, and knowledge distribution via the Internet and intranets.

IEEE  
**Intelligent  
SYSTEMS**

& their applications

[computer.org/intelligent](http://computer.org/intelligent)

umn values as equal, not equal, or adjacent.

Figure 7 shows SubdueCL's representation of a chess domain example. Each piece is represented by two vertices corresponding to a piece's row and column, connected by a *position* relation (for example, WKC stands for white king column). Instead of *less-than* relations, we used *eq* and *noteq* relations between all such rows and columns.


We tested the three systems for predictive accuracy on this domain, using threefold cross validation, with significance values gathered from a paired-student t-test. The accuracy results were 99.8% for FOIL, 99.77% for C4.5, and 99.21% for SubdueCL. The accuracy difference between FOIL and SubdueCL is significant at the 0.19 level (that is, the probability that the difference is insignificant is 0.19), and the difference between C4.5 and SubdueCL is significant at the 0.23 level.

FOIL learned six rules, SubdueCL learned five rules (substructures), and C4.5 learned 43 rules. All three systems discovered four rules that described in each case approximately 2,000 of the 2,118 positive examples. Figure 7c shows two of these rules described as substructures. The remaining rules differed among systems and proved to be more powerful for FOIL and C4.5 because of these systems' ability to learn numeric ranges.

Next, we tested the systems on a complete set of 958 possible board configurations at the end of tic-tac-toe games. The target concept is "a win for x." We supplied the three systems with the values (X, O, and blank) for each of the nine board positions. Unlike the other systems, SubdueCL does not key on individual position values but uses relational information between board positions to learn the three win concepts: three-in-a-row, three-in-a-column, and three-in-a-diagonal. The accuracy results are therefore 100% for SubdueCL, 92.35% for FOIL (the difference is significant at the 0.21 level), and 96.03% for C4.5 (the difference is significant at the 0.03 level).

In the final experiment, we attempted to differentiate Beethoven works from Bach works, using a set of musical excerpts described by a sequence of pitch values. We obtained the 100 Bach chorales from the UC Irvine repository and randomly selected the Beethoven works from the composer's collected works. To provide examples for the three learning systems, we selected a musical theme from each of 50 Beethoven examples and repeated it 10 times, with a varying pitch offset and surrounded by random pitch values. We tested the learning systems both

by using absolute pitch values and by using the relative difference between one pitch value and the next. All three systems performed better with pitch difference values. Unlike FOIL and C4.5, SubdueCL used the relational information between successive notes to learn the embedded musical sequences for the positive examples. The accuracy values were 100% for SubdueCL, 85.71% for FOIL (the difference is significant at the 0.06 level), and 82% for C4.5 (the difference is significant at the 0.00 level).

 OUR EXPERIMENTAL RESULTS indicate that SubdueCL, the graph-based relational concept learner, is competitive with logic-based relational concept learners on a variety of domains. This comparison has identified a number of avenues for enhancements. SubdueCL would benefit from the ability to identify ranges of numbers. We could accomplish this by utilizing the system's existing capability to find similar but not exact matches of a substructure in the input graph. Numeric values within the instances could be generalized to the encompassing range. A graph-based learner also needs the ability to represent recursion, which plays a central part in many logic-based concepts. More research is needed to identify representational enhancements for describing recursive structures—for example, graph grammars. Our future work will also focus on extending Subdue to handle other forms of learning, such as clustering.

We are continuing our testing of Subdue in real-world applications. In biochemistry, for example, we are applying Subdue to data from the Human Genome Project to find patterns in the DNA sequence that indicate the presence of a gene-transcription-factor site. Unlike other approaches to finding patterns in gene data,<sup>11</sup> Subdue uses a graph to represent structural information in the sequence. We hope that the discovered patterns will point to genes in uncharted areas of the DNA sequence.

In another area of chemistry, we are applying SubdueCL to the Predictive Toxicology Challenge data. This data contains the structural descriptions of more than 300 chemi-

cal compounds that have been analyzed for carcinogenicity. Each compound (except for about 30 held out for future testing) is labeled as either cancer-causing or not. Our goal is to find a pattern in the cancerous compounds that does not occur in the noncancerous compounds. So far, SubdueCL has found several promising patterns, which are currently under evaluation in the University of Texas at Arlington's Department of Chemistry.

In addition, we are applying Subdue to a number of other databases, including the Aviation Safety Reporting System database, US Geological Survey earthquake data, and software call graphs. Subdue has discovered several interesting patterns in the ASRS database. Burke Burkart of UTA's Department of Geology evaluated Subdue's results on the geology data and found that Subdue correctly identified patterns dependent on earthquake depth, often the distinguishing factor among earthquake types. These and other results show that Subdue discovers relevant knowledge in structural data and that it scales to large databases. ■

## Acknowledgments

Our work was supported by NSF grant IRI-9615272 and THECB grant 003656-045.

## References

1. P. Cheeseman and J. Stutz, "Bayesian Classification (AutoClass): Theory and Results," *Advances in Knowledge Discovery and Data Mining*, U.M. Fayyad et al., eds., MIT Press, Cambridge, Mass., 1996, pp. 153–180.
2. D. Fisher, "Knowledge Acquisition via Incremental Conceptual Clustering," *Machine Learning*, Vol. 2, No. 2, 1987, pp. 139–172.
3. J.R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, Calif., 1993.
4. U.M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From Data Mining to Knowledge Discovery: An Overview," *Advances in Knowledge Discovery and Data Mining*, U.M. Fayyad et al., eds., MIT Press, Cambridge, Mass., 1996, pp. 1–34.
5. J. Rissanen, *Stochastic Complexity in Statistical Inquiry*, World Scientific Publishing, Singapore, 1989.
6. D.J. Cook and L.B. Holder, "Substructure Discovery Using Minimum Description Length and Background Knowledge," *J. Artificial Intelligence Research*, Vol. 1, 1994, pp. 231–255.
7. D.J. Cook, L.B. Holder, and S. Djoko, "Scalable Discovery of Informative Structural Concepts Using Domain Knowledge," *IEEE Expert*, Vol. 11, No. 5, Sept./Oct. 1996, pp. 59–68.
8. S. Djoko, D.J. Cook, and L.B. Holder, "An Empirical Study of Domain Knowledge and Its Benefits to Substructure Discovery," *IEEE Trans. Knowledge and Data Engineering*, Vol. 9, No. 4, 1997, pp. 575–586.
9. G. Galal, D.J. Cook, and L.B. Holder, "Improving Scalability in a Scientific Discovery System by Exploiting Parallelism," *Proc. Int'l Conf. Knowledge Discovery and Data Mining*, AAAI Press, Menlo Park, Calif., 1997, pp. 171–174.
10. R.M. Cameron-Jones and J.R. Quinlan, "Efficient Top-Down Induction of Logic Programs," *SIGART Bulletin*, Vol. 5, No. 1, 1994, pp. 33–42.
11. M.W. Craven and J.W. Shavlik, "Machine Learning Approaches to Gene Recognition," *IEEE Expert*, Vol. 9, No. 2, Mar./Apr. 1993, pp. 2–10.

**Diane J. Cook** is an associate professor in the Computer Science and Engineering Department at the University of Texas at Arlington, where she is a director of the Learning and Planning Laboratory. Her research interests include artificial intelligence, machine planning, machine learning, robotics, and parallel algorithms for artificial intelligence. Cook received her BS from Wheaton College and her MS and PhD from the University of Illinois. Contact her at the Dept. of Computer Science and Engineering, Univ. of Texas at Arlington, Box 19015, Arlington, TX 76019-0015; cook@cse.uta.edu; http://www-cse.uta.edu/~cook.

**Lawrence B. Holder** is an associate professor and associate chair in the Department of Computer Science and Engineering at the University of Texas at Arlington, where he is also a codirector of the Learning and Planning Laboratory. His research interests include artificial intelligence and machine learning. Holder received his BS in computer engineering and his MS and PhD in computer science, all from the University of Illinois at Urbana-Champaign. He is a member of the AAAI, the ACM, and the IEEE. Contact him at the Dept. of Computer Science and Engineering, Univ. of Texas at Arlington, Box 19015, Arlington, TX 76019-0015; holder@cse.uta.edu; http://www-cse.uta.edu/~holder.