Learning from Supervised Graphs

Joseph Potts, Diane J. Cook and Lawrence B. Holder

Summary. We describe an approach to learning patterns in relational data represented as a graph. The approach, implemented in the Subdue system, searches for patterns that maximally compress the input graph. Subdue can be used for supervised learning, as well as unsupervised pattern discovery and clustering.

Mining graph-based data raises challenges not found in linear attribute-value data. However, additional requirements can further complicate the problem. In particular, we describe how concepts can be learned from training examples which are embedded into a single connected graph, or *supervised graph*. We demonstrate the technique using data from a NASA SST domain as well as a homeland security domain.

1 Introduction

Much of current data mining research focuses on algorithms to discover sets of attributes that can discriminate data entities into classes, such as shopping or banking trends for a particular demographic group. In contrast, we are developing data mining techniques to discover patterns consisting of complex relationships between entities. The field of relational data mining, of which graph-based relational learning is a part, is a new area investigating approaches to mining relational information by finding associations involving multiple tables in a relational database.

Two main approaches have been developed for mining relational information: logic-based approaches and graph-based approaches. Logic-based approaches fall under the area of inductive logic programming (ILP) [1]. ILP embodies a number of techniques for inducing a logical theory to describe the data, and many techniques have been adapted to relational data mining [2]. Graph-based approaches differ from logic-based approaches to relational mining in several ways, the most obvious of which is the underlying representation. Furthermore, logic-based approaches rely on the prior identification of the predicate or predicates to be mined, while graph-based approaches are more data-driven, identifying any portion of the graph that has high support. However, logic-based approaches allow the expression of more complicated patterns involving, e.g., recursion, variables, and constraints among variables. These representational limitations of graphs can be overcome, but at a computational cost.

J. Potts et al.: *Learning from Supervised Graphs*, Studies in Computational Intelligence (SCI) **52**, 183–201 (2007)
www.springerlink.com

Our research is particularly applicable to domains in which the data is event driven, such as counter-terrorism intelligence analysis, and domains where distinguishing characteristics can be object attributes or relational attributes. This ability has also become a crucial challenge in many security-related domains. For example, the US House and Senate Intelligence Committees' report on their inquiry into the activities of the intelligence community before and after the September 11, 2001 terrorist attacks revealed the necessity for "connecting the dots" [3]; that is, focusing on the relationships between entities in the data, rather than merely on an entity's attributes. A natural representation for this information is a graph, and the ability to discover previously unknown patterns in such information could lead to significant improvement in our ability to identify potential threats. Similarly, identifying characteristic patterns in spatial or temporal data can be a critical component in acquiring a foundational understanding of important research in many of the basic sciences.

Learning systems capable of utilizing graph-based data typically require training examples to be represented using disjoint graphs, one for each example. In a highly relational domain, however, the positive and negative examples of a concept are not easily separated. We call such a graph a *supervised graph*, because the graph as a whole contains embedded class information which may not be easily separated into individual labeled components. In this chapter we describe a method of learning concepts from examples in supervised graphs that builds upon the capabilities of the Subdue graph-based data mining system.

2 Related Work

Graph-based data mining (GDM) is the task of finding novel, useful, and understandable graph-theoretic patterns in a graph representation of data. Several approaches to GDM exist based on the task of identifying frequently occurring subgraphs in graph transactions, i.e., those subgraphs meeting a minimum level of support. Kuramochi and Karypis [4] developed the FSG system for finding all frequent subgraphs in large graph databases. FSG starts by finding all frequent single and double edge subgraphs. Then, in each iteration, it generates candidate subgraphs by expanding the subgraphs found in the previous iteration by one edge. In each iteration the algorithm checks how many times the candidate subgraph occurs within an entire graph. The candidates, whose frequency is below a user-defined level, are pruned. The algorithm returns all subgraphs occurring more frequently than the given level.

Yan and Han [5] introduced gSpan, which combines depth-first search and lexicographic ordering to find frequent subgraphs. Their algorithm starts from all frequent one-edge graphs. The labels on these edges together with labels on incident vertices define a code for every such graph. Expansion of these one-edge graphs maps them to longer codes. The codes are stored in a tree structure such that if $\alpha = (a_0, a_1, ..., a_m)$ and $\beta = (a_0, a_1, ..., a_m, b)$, the β code is a child of the α code. Since every graph can map to many codes, the codes in the tree structure are not

unique. If there are two codes in the code tree that map to the same graph and one is smaller then the other, the branch with the smaller code is pruned during the depth-first search traversal of the code tree. Only the minimum code uniquely defines the graph. Code ordering and pruning reduces the cost of matching frequent subgraphs in gSpan.

Inokuchi et al. [6] developed the apriori-based graph mining (AGM) system, which uses an approach similar to Agrawal and Srikant's [7] apriori algorithm for discovering frequent itemsets. AGM searches the space of frequent subgraphs in a bottom-up fashion, beginning with a single vertex, and then continually expanding by a single vertex and one or more edges. AGM also employs a canonical coding of graphs in order to support fast subgraph matching. AGM returns association rules satisfying user-specified levels of support and confidence.

We distinguish graph-based relational learning (GBRL) from graph-based data mining in that GBRL focuses on identifying novel, but not necessarily most frequent, patterns in a graph representation of data [8]. Only a few GBRL approaches have been developed to date. Two specific approaches, Subdue [9] and GBI [10], take a greedy approach to finding subgraphs maximizing an information theoretic measure. Subdue searches the space of subgraphs by extending candidate subgraphs by one edge. Each candidate is evaluated using a minimum description length metric [11], which measures how well the subgraph compresses the input graph if each instance of the subgraph were replaced by a single vertex. GBI continually compresses the input graph by identifying frequent triples of vertices, some of which may represent previously compressed portions of the input graph. Candidate triples are evaluated using a measure similar to information gain. Kernel-based methods have also been used for supervised GBRL [12].

3 Graph-based Relational Learning in Subdue

The Subdue graph-based relational learning system[1] [9, 13] encompasses several approaches to graph-based learning, including discovery, clustering, and supervised learning, which will be described in this section. Subdue uses a labeled graph $G = (V, E, L)$ as both input and output, where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of vertices, $E = \{(v_i, v_j) | v_i, v_j \in V\}$ is a set of edges, and L is a set of labels that can appear on vertices and edges. The graph G can contain directed edges, undirected edges, self-edges (i.e., $(v_i, v_i) \in E$), and multi-edges (i.e., more than one edge between vertices v_i and v_j). The input graph need not be connected, but the learned patterns must be connected subgraphs (called substructures) of the input graph. The input to Subdue can consist of one large graph or several individual graph transactions, and in the case of supervised learning, the individual graphs are classified as positive or negative examples.

[1] Subdue source code, sample datasets, and publications are available at http://ailab.uta.edu/subdue.

3.1 Substructure Discovery

SUBDUE's discovery algorithm is shown in Fig. 1 and is given as the input graph, the beam length, and a limit on the total number of substructures considered by the algorithm.

Subdue searches for a substructure that best compresses the input graph. Subdue uses a variant of beam search for its main search algorithm. A substructure in Subdue consists of a subgraph definition and all its occurrences throughout the graph. The initial state of the search is the set of substructures consisting of all uniquely labeled vertices. The only operator of the search is the *ExtendSubstructure* operator. As its name suggests, it extends a substructure in all possible ways by a single edge and a vertex, or by only a single edge if both vertices are already in the subgraph.

The search progresses by applying the *ExtendSubstructure* operator to each substructure in the current state. The resulting state, however, does not contain all the substructures generated by the *ExtendSubstructure* operator. The substructures are kept on a queue and are ordered based on their description length (or sometimes referred to as value) as calculated using the MDL principle described later.

The search terminates upon reaching a user-specified limit on the number of substructures extended, or upon exhaustion of the search space. Once the search

```
Subdue  (Graph, Beam, Limit)
    queue Q = {v | v is a vertex in Graph having a unique label}
    bestSub = first substructure in Q
    repeat
        newQ = { }
        for each substructure S ∈ Q
            newSubs = Extend-Substructure (S, Graph)
                      in all possible ways
            Evaluate (newSubs)
            newQ = newQ ∪ newSubs mod Beam
            Limit = Limit − 1
        if best substructure in newQ better than bestSub
        then bestSub = best substructure in Q
        Q = newQ
    until Q is empty or Limit ≤ 0
    return bestSub
```

Fig. 1. SUBDUE's discovery algorithm

terminates and Subdue returns the list of best substructures found, the graph can be compressed using the best substructure. The compression procedure replaces all instances of the substructure in the input graph by single vertices, which represent the substructure definition. Incoming and outgoing edges to and from the replaced instances will point to, or originate in the new vertex that represents the instance. The Subdue algorithm can be invoked again on this compressed graph. This procedure can be repeated a user-specified number of times, and is referred to as an iteration.

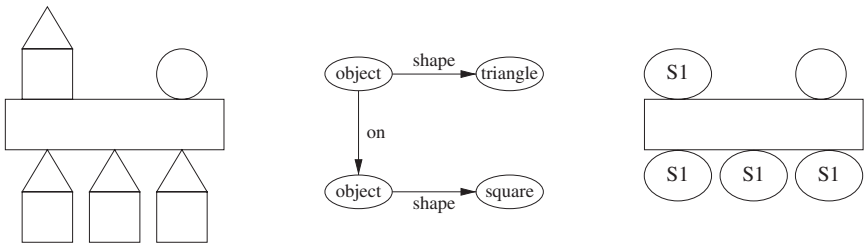
Subdue's search is guided by the minimum description length (MDL) [11] principle, which seeks to minimize the description length of the entire data set. The evaluation heuristic based on the MDL principle assumes that the best substructure is the one that minimizes the description length of the input graph when compressed by the substructure [9]. The description length of the substructure S given the input graph G is calculated as $DL(S) + DL(G|S)$, where $DL(S)$ is the description length of the substructure and $DL(G|S)$ is the description length of the input graph compressed by the substructure. Description length $DL()$ is calculated as the number of bits in a minimal encoding of the graph. Subdue seeks a substructure S that maximizes compression as calculated in (1).

$$Compression = \frac{DL(G)}{DL(S) + DL(G|S)} \tag{1}$$

As an example, Fig. 2a shows a collection of geometric objects described by their shapes and their "ontop" relationship to one another. Figure 2b shows the graph representation of a portion ("triangle on square") of the input graph for this example and also represents the substructure minimizing the description length of the compressed graph. Figure 2c shows the input example after being compressed by the substructure.

3.2 Graph-based Clustering

Given the ability to find a prevalent subgraph pattern in a larger graph and then compress the graph with this pattern, iterating over this process until the graph can no longer be compressed will produce a hierarchical, conceptual clustering of the input



(a) Input (b) Substructure (c) Compressed

Fig. 2. Example of Subdue's substructure discovery capability

data. On the ith iteration, the best subgraph S_i is used to compress the input graph, introducing new vertices labeled Si in the graph input to the next iteration. Therefore, any subsequently discovered subgraph S_j can be defined in terms of one or more S_i, where $i < j$. The result is a lattice, where each cluster can be defined in terms of more than one parent subgraph. For example, Fig. 3 shows such a clustering done on a DNA molecule. See [14] for more information on graph-based clustering. The idea of clustering graphs has been explored by others such as Günter and Bunke [15, 16] who also determine the optimal number of clusters automatically, and by Giugno and Shasha [17], who provide graph querying and clustering tools for a wide variety of graph types. Our approach is unique in employing a discovery algorithm to perform the clustering, and in yielding a hierarchical lattice of graph clusters from the original graph data.

3.3 Supervised Learning from Graphs

Extending a graph-based discovery approach to perform supervised learning involves, of course, the need to handle negative examples (focusing on the two-class scenario). In the case of a graph the negative information can come in two forms. First, the data may be in the form of numerous small graphs, or graph transactions, each labeled either positive or negative. Second, data may be composed of two large graphs: one positive and one negative.

The first scenario is closest to the standard supervised learning problem in that we have a set of clearly defined examples. Figure 4a depicts a simple set of positive and negative examples. Let G^+ represent the set of positive graphs, and G^- represent the set of negative graphs. Then, one approach to supervised learning is to find a subgraph that appears often in the positive graphs, but not in the negative graphs. This amounts to replacing the information-theoretic measure with simply an error-based measure. For example, we would find a subgraph S that minimizes

$$\frac{|\{g \in G^+ | S \not\subseteq g\}| + |g \in G^- | S \subseteq g\}|}{|G^+| + |G^-|},$$

where $S \subseteq g$ means S is isomorphic to a subgraph of g. The first term of the numerator is the number of false negatives, and the second term is the number of false positives.

This approach will lead the search toward a small subgraph that discriminates well, e.g., the subgraph in Fig. 4b. However, such a subgraph does not necessarily compress well, nor represent a characteristic description of the target concept. We can bias the search toward a more characteristic description by using the information-theoretic measure to look for a subgraph that compresses the positive examples, but not the negative examples. If $I(G)$ represents the description length (in bits) of the graph G, and $I(G|S)$ represents the description length of graph G compressed by subgraph S, then we can look for an S that minimizes $I(G^+|S) + I(S) + I(G^-) - I(G^-|S)$, where the last two terms represent the portion of the negative graph incorrectly compressed by the subgraph. This approach

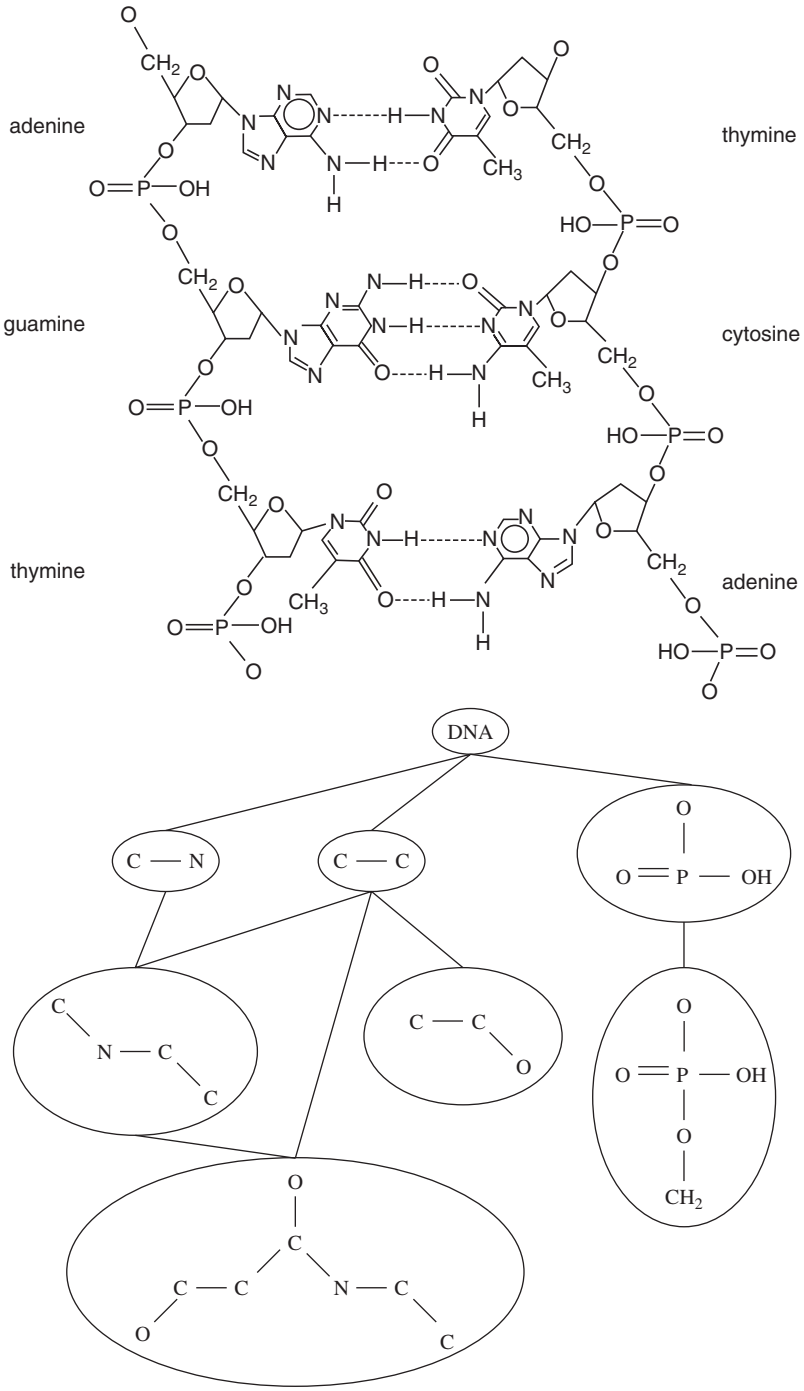


Fig. 3. Example of Subdue's clustering (bottom) on a portion of DNA (top)

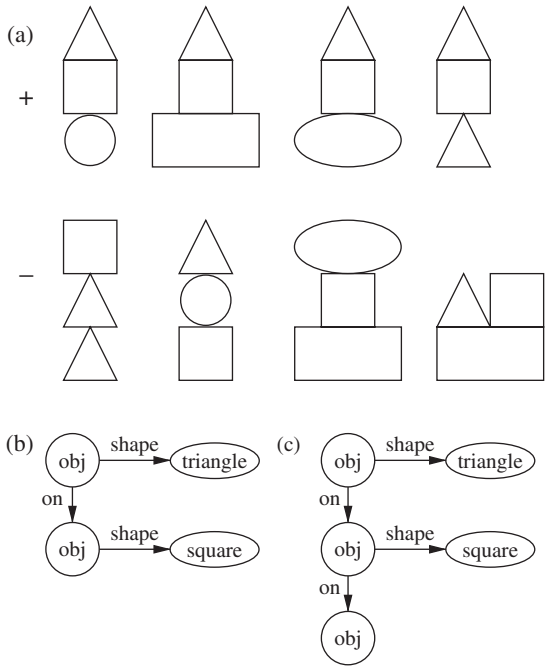


Fig. 4. Graph-based supervised learning example with (**a**) four positive and four negative examples, (**b**) one possible graph concept, and (**c**) another possible graph concept

will lead the search toward a larger subgraph that characterizes the positive examples, but not the negative examples, e.g., the subgraph in Fig. 4c.

Finally, this process can be iterated in a set-covering approach to learn a disjunctive hypothesis. If using the error measure, then any positive example containing the learned subgraph would be removed from subsequent iterations. If using the information-theoretic measure, then instances of the learned subgraph in both the positive and negative examples (even multiple instances per example) are compressed to a single vertex. See [18] for more information on graph-based supervised learning.

4 Learning from Supervised Graphs

Learning systems capable of utilizing graph-based input have typically required the training examples to be represented as disjoint graphs. Input for these systems consists of training examples represented as individual graphs, each of which is an example of one of n classes. The goal is to learn a concept which can be used to determine to which class a previously unseen graph belongs.

In a domain where training examples are naturally embedded (and possibly overlap) in a single graph, efficiently transforming the data for input to systems such as

these can be quite difficult. If a system requires individual graphs for each example, then it is necessary to excise each example along with some amount of surrounding graph structure to create a disconnected graph containing that example. If the examples are close enough to each other in the original graph, then this surrounding data may overlap with the surrounding data of another example. In fact, the training example graph may even have to include all or part of another example. This overlap will result in some data appearing in more than one example graph.

Determining just how much structure to include in an example is tricky. Taking too large a region around the example causes extra data to be handled. Taking too small a region may result in the loss of potentially vital information. Since processing graph-based data is very resource intensive, any redundant information can have a drastic effect on performance. Failure to include enough data may result in the inability of the system to learn.

We hypothesize that a compression-based graph mining algorithm can be used to learn class information embedded in a single, connected graph. We develop a learner that allows the input graph, containing all the training examples for all classes to be input with a minimum of preprocessing and a minimum of added or redundant information. In a highly complex relational domain, positive and negative examples of a concept are not easily separated into nonoverlapping graphs. We call such a graph with embedded, possibly overlapping examples a *supervised graph*, or a graph that contains embedded class information which may not be easily separated into individual labeled components.

For example, consider a social network in which we look for patterns distinguishing various income levels. Individuals of a particular income level can appear anywhere in the graph and may be interact with or be related to individuals at other income levels, so we cannot easily partition the graph into separate training cases without potentially severing the target relationships.

To validate our hypothesis, we propose a representation requiring the addition to the input graph of one vertex for each example. We also propose an enhancement of the Subdue algorithm which will construct substructures capable of identifying the examples of each class guided by a new performance metric called *classification compression*. Finally, we propose a representation for these learned substructures called a *classification sequence* which facilitates the determination of class membership for new observations.

4.1 Problem Statement

Our approach to learning concepts from supervised graphs is embodied in the Subdue-EC algorithm. In addition to the labeled graph defined earlier as $G = (V, E, L)$, Subdue-EC also expects as input a set of examples, X, where each $x \in X$ is a set of one or more vertices in V and a set C of class designations, one for each example from the set of n classes. Subdue-EC the learns a concept which is expressed as a set of subgraphs, S, which can be used to assign a class to designated sets of vertices in graphs.

For simplicity in the following discussion, we will consider the two-class learning problem. However, the algorithm, the performance measures, and the classification concepts are applicable to problems with any number of classes. Furthermore, the examples are represented as sets of vertices. Again, for simplicity, we will use single-vertex examples, but any number of vertices may be part of a training example.

4.2 Evaluating Concepts

To be able to perform inductive learning on a single graph with both positive and negative examples, compression of the input graph becomes a less desirable evaluation metric because the graph contains examples of all classes. To allow the MDL principle to guide us in classification, we have to look not at the graph, but at the classification itself. That is, we assume that our receiving agent already has the graph and all of the examples it contains. What we need to send is the classification of those examples. The straightforward way to do that is simply send the class number for each example. Since the examples are in the same order in the receiver's copy of the graph as they are in the sender's, we can just send the class number for examples $1 \ldots n$ and the agent will be able to classify each example. The description length of this naive classification, C_{naive}, is simply the number of bits required to provide a class number for each of the examples. Thus $DL(C_{naive})$ is $nlog_2k$, where n is the number of classes and k is the number of examples.

An alternative to just telling the receiver the class of each example, is to provide the concept as a sequence of substructures $s_1, s_2, \ldots, s_j$, each with an associated class. If an instance of one of the substructures is found in the new graph, then the class associated with that substructure is assigned to all vertices in the substructure instance. The description length of this encoding is thus the description length of the substructure sequence with classes, or classification sequence CS. This is computed as the sum of the description length of the substructures in the sequence, $DL(CS) = \Sigma_i DL(s_i)$.

Of course, this approach may misclassify or fail to classify some examples. In this case, we must inform the agent of the correct classification for those examples. Thus the descriptive length of our alternative message is the sum of the descriptive lengths of each substructure, the class number for each substructure, and encoded exceptions for each class. The description length of this exception list, $DL(EL)$, will require $(k+m+u)log_2(n+1)$ bits, where m is the number of misclassified examples and u is the number of examples left unclassified by the substructure sequence.

If the description length of CS together with ES is smaller than the description length of the naive classification, $DL(CS) + DL(EL) < DL(C_{naive})$, then we will have reduced the message size required to convey the classification to the receiver. We will thus have compressed the classification using our concept, or classification sequence CS. In the same way that Subdue searches for a subgraph that best compresses the input graph, Subdue-EC searches for a classification sequence which provides the best compression of the naive classification. We can now calculate compression as

$$Compression = \frac{DL(CS) + DL(EL)}{DL(C_{naive})} = \frac{\Sigma_i DL(s_i) + (k+m+u)log_2(n+1)}{nlog_2 k}.$$

As before, we take the reciprocal of the compression and use the resulting value as the evaluation measure for potential concepts (classification sequences). The classification sequence that yields the largest value is selected by Subdue-EC as the best concept.

4.3 Identifying Examples

Now that we have a metric for evaluating potential concepts, the remaining issue is how to identify the embedded examples and their associated classes. This is accomplished by the addition of a vertex to the graph for each training example. The vertex is labeled with the class name of the example and is connected by an edge to each vertex in the graph that is part of the training example. We do not need to mark the edges of the example since Subdue-EC will include them in the classifying substructure if they are needed for classification purposes. This vertex is relabeled by Subdue-EC to "EXAMPLE."

Observe that with this representation, vertices and edges in the original graph can be members of more than one training example, possibly from different classes. This is the type of representational freedom that we desire. An individual may interact with one group that is represents a terrorist threat and at the same time do business with other groups that are not under suspicion. In fact, these types of overlaps are sometimes critical to finding the desired concept.

In addition, note that now we can make the initial state of the search algorithm much smaller by starting with only one substructure. All we need are the instances of the single vertex subgraph "EXAMPLE," since all classifying substructures must start there. This "focuses" the search immediately on the right place. Subdue-EC is constrained to never add an "EXAMPLE" node during substructure extension since no classifying substructure can have more than one such vertex.

When the example in Fig. 5 is processed by Subdue-EC, the following five classifying substructures are discovered:

- D (negative)
- B→A→C (positive)
- C (negative)
- B→A→B (positive)
- B (negative)

In this description, the underlined vertices are the ones being classified (the vertices to which an "EXAMPLE" vertex is connected). Thus the first vertex labeled "B" in the B→A→B subgraph is being classified as positive, not the second one. The second "B" vertex is later classified as negative.

Two points should be addressed here. The order that the classifying substructures are applied must be the same as the order in which they were discovered. This facilitates the discover of smaller substructures. For example, the substructure B→A→B

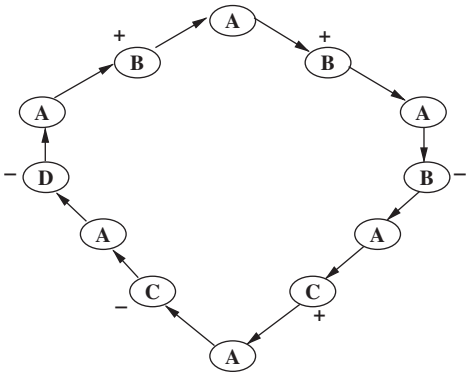


Fig. 5. Embedded examples. Positive-labeled vertices are connected to vertices in the figure with a "+," and negative-labeled vertices are connected to vertices the figures with a "−"

compresses away both positive B vertices. Any remaining B vertices are thus part of negative examples.

5 Experimental Results Using NASA Data

To validate the effectiveness of Subdue-EC to discover concepts from supervised graphs, we chose a simple classification task on a large set of data. We obtained sea surface temperature (SST) data from NASA [19]. This data is averaged over a five-day period and placed on a one degree global grid. The data contains a fill value for grid points for which the SST is not available such as land or due to missing information. We first determined for each grid point whether the temperature *increased*, *decreased*, or stayed the *same* from January 8, 1990 to February 7, 1990. We then placed the nonfill temperature values into one of 9 equal width bins.

We represent this data as a graph, as shown in Fig. 6. Vertices are used to represent each month, discretized latitude and longitude values, hemisphere, and change in temperature from one month to the next. Vertices labeled with "increase" represent regions with increasing temperatures and "decrease" vertices represent regions with decreasing temperatures. Each vertex is connected to its northern and western neighbors, continued in a circle around the globe. This results in a graph that looks like a mesh cylinder, containing 259,200 vertices and 323,640 edges. Each grid point also was connected to a unique vertex containing its temperature binand to another unique vertex labeled N or S, indicating the position's hemisphere.

Note that this is an example dataset where there may exist overlap between instances of the same or different classes. While instances could be extracted from the graph and presented as separate subgraphs for training, the amount of information surrounding each region node that is critical for learning the concept is not known. As a result, the instances cannot be effectively extracted without jeopardizing the accuracy of the result and greatly affecting the runtime of the system due to redundancy in the instances.

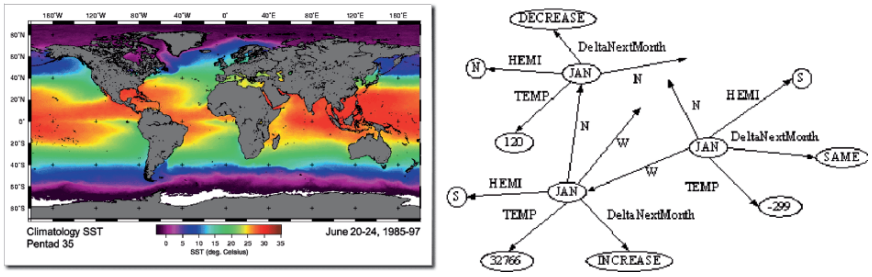


Fig. 6. NASA's SST data (*left*) and Subdue graph representation (*right*)

Table 1. Accuracy results on NASA SST data

run	#substructures generated	time (seconds)	accuracy on training data	accuracy on test data
0	106	52822	86.07%	85.31%
1	104	49669	85.81%	85.26%
2	109	76336	85.57%	85.25%
3	100	71679	85.81%	85.32%
4	104	78874	85.66%	85.69%
5	111	80388	85.84%	86.22%
6	108	73174	85.84%	85.00%
7	112	77236	85.81%	85.39%
8	99	75392	85.64%	84.57%
9	108	80497	85.76%	84.10%
Min	99	49669	85.57%	84.10%
Max	112	80497	86.07%	86.22%
Avg	106.1	71607	85.78%	85.21%

Table 1 shows the results of tenfold cross-validation testing applied to the NASA SST data. For each fold 90% of the grid nodes were randomly selected as training data and the remaining 10% were assigned to a second copy of the graph and was used for testing. The accuracy was good, and accuracy for the test data was fairly consistent with the accuracy on the training data.

We also conducted some tests varying Subdue parameters such as beam size and limit (the number of substructures extended and evaluated). These tests were conducted on 100% of the examples. That is, class vertices were attached to all 64,800 grid points. Surprisingly, the accuracy did not change a lot even when the number of substructures decreased substantially. This is due to the tradeoff in the numerator of classification compression between substructure size and misclassifications. For the NASA data, adding one more vertex adds about 16 bits to the size of the substructure. Since there are about 64,000 examples, the penalty for a misclassification is about 16 bits (that is how many bits it takes to tell the sender the example number of the misclassified example). Thus eliminating two misclassifications more than pays for making the substructure one vertex bigger. This tends to drive substructure growth larger and larger until terminated by the limit parameter. On the other hand,

Table 2. Accuracy with increasing limit

limit	substructures generated	time	accuracy
6	16	17345	83.70%
7	11	10603	84.41%
8	28	28099	85.17%
9	28	26768	85.17%
10	26	26077	85.12%
20	71	53370	85.46%
40	113	77107	85.87%
60	135	82789	85.90%
80	137	92012	86.09%
100	146	101324	86.23%

the unclassified examples are then classified on a subsequent iteration. Thus there are more substructures and larger substructures as the limit is increased, but the accuracy does not significantly improve (see Table 2).

Our final test on the NASA data is to train Subdue-EC with all of the 1990 data and use the learned concept to classify 1991 data. We created a graph using the same representation for data from January 8, 1991 to February 7, 1991, and calculated the accuracy of the learned substructures on this new data. Using a limit of ten substructures, Subdue-EC achieved 81.98% accuracy, showing that the learned substructures have classification value even for subsequent years.

The learned substructures are what one might intuitively expect. The first in the sequence addresses the large number of *same* examples. These are primarily land areas which are still land masses 30 days later and therefore still have fill values for the temperature and receive the same classification. Otherwise, the concepts represent the ideas that the northern hemisphere gets colder in winter and the southern hemisphere gets warmer. Interestingly, temperature bin 0 classifies as *same*. This may be because the coldest areas do not change temperature much throughout the year. In addition, southern hemisphere grid points north of temperature bin 6 decrease. This is consistent with the fact that these areas are on the equator and therefore start to cool off as winter drags on and they get less sun. Finally, it should be noted that none of the tests ever leave any data unclassified. On these data there always seems to be benefit to including a catchall classification substructure at the end that has enough correct classifications to pay for its misclassifications.

6 Experimental Results Using Security Data

As part of a government-sponsored program, a domain has been built to simulate the evidence available about terrorist groups and their plans prior to execution. This domain is motivated from an understanding of the real problem of intelligence data analysis. The domain consists of a number of concepts, including threat and nonthreat actors, threat, and nonthreat groups, targets, exploitation modes (vulnerability

modes are exploited by threat groups, productivity modes are exploited by threat and nonthreat groups), capabilities, resources, communications, visits to targets, and transfer of resources between actors, groups, and targets.

The domain follows a general plan of starting a group, recruiting members with needed capabilities, acquiring needed resources, visiting a target, and then exploiting the target. The data we use for our experiments represents the activities of terrorist organizations as they attempt to exploit vulnerable targets, represented by the execution of five different event types. They are:

- *Two-way-communication.* Involves one initiating person and one responding person.
- *N-way-communication.* Involves one initiating person and multiple respondents.
- *Generalized-transfer.* One person transfers a resource.
- *Applying-capability.* One person applies a capability to a target.
- *Applying-resource.* One person applies a resource to a target.

All data is generalized so that no specific names are used. The simulator generates evidence related to all of these events, and this evidence is passed through filters varying the degree of observability and noise in the final evidence.

For our experiments, a graph was creating in which vertices are used to represent member agents from threat and nonthreat groups. Anyone which whom these agents communicates is also added to the graph and connected to the agent with an undirected "association" edge. Communication events between associates are similarly represented with "association" edges.

In addition, each person may be described using attribute and capability vertices. In the simulated data, every individual is assigned at least two strong "trust-link" attributes (e.g., school, place of worship, former military unit, extended family) and at least two weaker "culture-link" attributes (e.g., nationality, language, religion) that are commonly applied in social network development. Capabilities refer to unique abilities exhibited by the individual. Figure 7 shows a portion of the graph generated for this dataset.

Our experiments were conducted on a large graph (graph1) consisting of 435,429 vertices and 763,504 edges representing 61,105 people as well as a smaller graph (graph2) consisting of 217,901 vertices and 314,793 edges representing 30,715 people. Class vertices labeled *threat* were attached to members of known threat groups, and *nonthreat* vertices were attached to members of nonthreat groups.

Our goal for the experiments was to see how well Subdue-EC could classify threat and nonthreat individuals, given training examples embedded in a single connected graph. In the original graphs there is a large predominance of nonthreat individuals (58,373, in contrast to the 1,732 threat individuals). To provide a stronger sample to the learning algorithm, we randomly sampled an equal number of threat and nonthreat individuals.

Table 3 summarizes the results for graph1. For the individuals that included one or more of the classifying substructures, Subdue's classification accuracy was 72.98%. However, the computational limitations of the discovery algorithm prevented further substructures from being discovered in a reasonable amount of time,

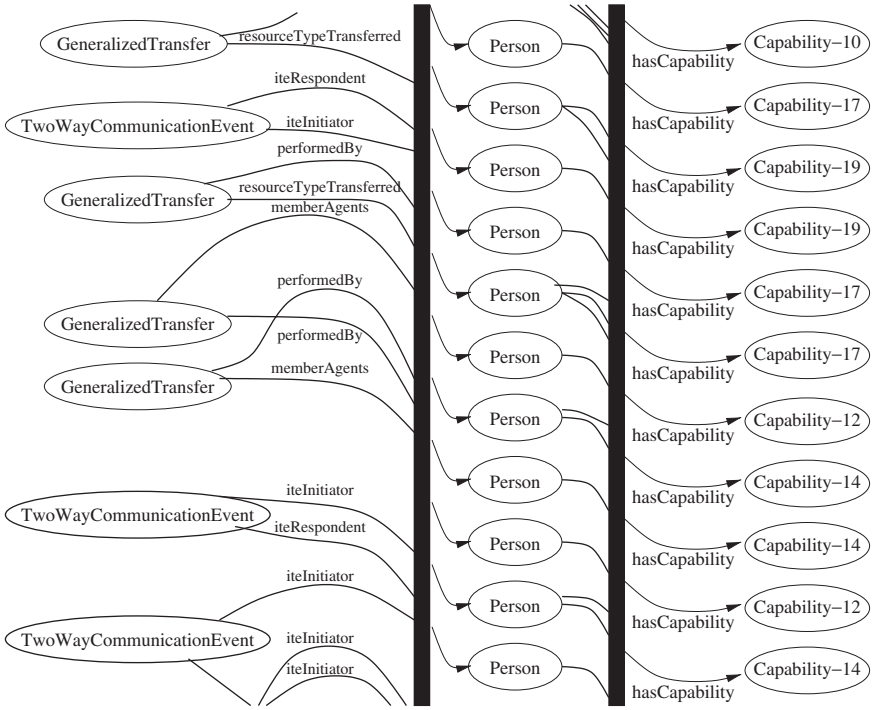


Fig. 7. A section of the graph representation for the counter-terrorism data

Table 3. Classification results on graph1

	total	correct	incorrect	unclassified
threats	1,732	765	35	932
nonthreats	1,732	70	290	1,372

so 2,304 individuals remained unclassified. The greatest number of misclassifications were false positives (classified as threats when the true classification is nonthreat), which is a preferred type of mistake for this problem.

Of the substructures that were discovered, many consisted of an individual exhibiting a particular capability. However, a few of the substructures, such as the one shown in Fig. 8, highlight an association between two individuals in addition to attributes and capabilities of the individuals.

The fact that Subdue discovered useful substructures that highlight relationships between the individuals to be classified highlights the strength of Subdue-EC. If the individuals have been separated into disjoint examples, this relationship could not have been found. If we tried to extract individuals with a large enough neighborhood of information around them to find these discoveries, several difficulties would arise. First, how much information do we retain? The user cannot always know a priori how much of a neighborhood must be extracted in order to retain all potentially

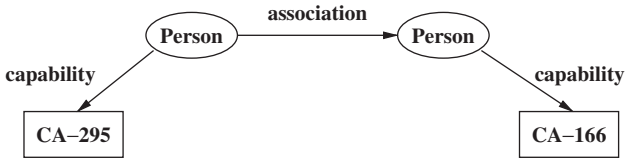


Fig. 8. A sample discovered substructures. This substructure highlights an association between two individuals, each with certain capabilities. The individual on the left is a known threat

Table 4. Results of graph1 testing on all individuals

	total	correct	incorrect	unclassified
threats	1,732	765	35	932
nonthreats	59,373	1,830	12,840	44,703

Table 5. Classification results on graph2

		total	correct	incorrect	unclassified
set 1	threats	1,225	463	28	734
	nonthreats	1,225	38	221	966
set 2	Threats	1,225	463	28	734
	nonthreats	29,490	876	5,596	23,018

useful information. Second, when the neighborhood of information is extracted, it is in essence reproduced for each example object that requires the information. This results in substantial cost increase both in memory and in processing time for the discovery algorithm.

To determine the effect of the sample size on Subdue's classification accuracy, we performed another classification experiment on graph1 in which training and testing were performed on the entire set of input threat and nonthreat individuals. The results are summarized in Table 4. As can be seen, the results did not change for threat individuals. While the number of correctly classified nonthreat individuals did increase, so did the number of misclassifications, resulting in a poorer performance than the earlier experiment.

In a separate experiment, we evaluated the generalizability of Subdue's results by using the substructures discovered in the first two experiments to classify individuals from a separate dataset, graph2. Table 5 shows the results of this experiment. As can be seen, while the percentage of accurate classifications does drop for the new dataset, Subdue still is able to perform fairly well on previously unseen data.

7 Conclusions

The handling of supervised graphs is an important direction for mining structural data. To extend our current work, we would like to handle embedded instances without a single representative instance node (the "increase" and "decrease" nodes in

our NASA example) and instances that may possibly overlap. In addition, improved scalability of graph operations is necessary to learn patterns, evaluate their accuracy on test cases, and ultimately to use the patterns to find matches is future intelligence data. The graph and subgraph isomorphism operations are a significant bottleneck to these capabilities. We are currently designing faster approximate versions of these operations to improve the scalability of graph-based relational learning.

8 Acknowledgments

This research is sponsored by the Air Force Research Laboratory (AFRL) under contract F30602-01-2-0570. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied of AFRL or the United States Government.

References

1. S. Muggleton, editor. *Inductive Logic Programming*. Academic, New York, 1992
2. S. Dzeroski and N. Lavrac, editors. *Relational Data Mining*. Springer, Berlin Heidelberg New York, 2001
3. U. Senate and H. C. on Intelligence. Joint inquiry into intelligence community activities before and after the terrorist attacks of september 11, 2001, December 2002
4. M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proceedings of the First IEEE Conference on Data Mining*, 2001
5. X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *Proceedings of the International Conference on Data Mining*, 2002
6. A. Inokuchi, T. Washio, and H. Motoda. Complete mining of frequent patterns from graphs: Mining graph data. *Machine Learning*, 50(3): 321–354, 2003
7. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the Twentieth Conference on Very Large Databases*, pages 487–499, 1994
8. L. Holder and D. Cook. Graph-based relational learning: Current and future directions. *ACM SIGKDD Explorations*, 5(1): 90–93, 2003
9. D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1: 231–255, 1994
10. K. Yoshida, H. Motoda, and N. Indurkhya. Graph-based induction as a unified learning framework. *Journal of Applied Intelligence*, 4: 297–328, 1994
11. J. Rissanen. *Stochastic Complexity in Statistical Inquiry*. World Scientific, Singapore, 1989
12. H. Kashima and A. Inokuchi. Kernels for graph classification. In *Proceedings of the International Workshop on Active Mining*, 2002
13. D. Cook and L. Holder. Graph-based data mining. *IEEE Intelligent Systems*, 15(2): 32–41, 2000
14. I. Jonyer, D. Cook, and L. Holder. Graph-based hierarchical conceptual clustering. *Journal of Machine Learning Research*, 2: 19–43, 2001
15. S. Gunter and H. Bunke. Self-organizing map for clustering in the graph domain. *Pattern Recognition Letters*, 23: 401–417, 2002

16. S. Gunter and H. Bunke. Validation indices for graph clustering. *Pattern Recognition Letters*, 24(8): 1107–1113, 2003
17. R. Giugno and D. Shasha. Graphgrep: A fast and universal method for querying graphs. In *Proceedings of the IEEE International Conference on Pattern Recognition*, pages 112–115, 2002
18. J. Gonzalez, L. Holder, and D. Cook. Graph-based relational concept learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, 2002
19. JPL. Physical oceanograpy DACC, WOCE global data, v2.0, satellite data, sea surface temperature, 2000