# A Serial Partitioning Approach to Scaling Graph-Based Knowledge Discovery

**Runu Rathi, Diane J. Cook, Lawrence B. Holder**

Department of Computer Science and Engineering
The University of Texas at Arlington
Box 19015, Arlington, TX 76019, USA
{rathi, cook, holder}@cse.uta.edu

**Abstract**

One of the main challenges for knowledge discovery and data mining systems is to scale up their data interpretation abilities to discover interesting patterns in large datasets. This research addresses the scalability of graph-based discovery to monolithic datasets, which are prevalent in many real-world domains like bioinformatics, where vast amounts of data must be examined to find meaningful structures. We introduce a technique by which these datasets can be automatically partitioned and mined serially with minimal impact on the result quality. We present applications of our work in both artificially-generated databases and a bioinformatics domain.

## Introduction

Several approaches have been proposed for the analysis and discovery of concepts in graphs in the context where graphs are used to model datasets. Modeling objects using graphs in Subdue [1] allows us to represent arbitrary relations among entities and capture the structural information. Although the subgraph isomorphism procedure needed to deal with these datasets has been polynomially constrained within Subdue, the system still spends a considerable amount of computation performing this task. The utilization of richer and more elaborate data representations for improved discovery leads to even larger graphs. The graphs are often so large that they can not fit into the dynamic memory of conventional computer systems. Even if the data fits into dynamic memory, the amount of memory left for use during execution of the discovery algorithm may be insufficient, resulting in an increased number of page swaps and ultimately performance degradation.

The goal of this research is to demonstrate that graph-based knowledge discovery systems can be made scalable through the use of sequential discovery over static partitions. To accomplish this goal, we have developed a serial graph partitioning algorithm to facilitate scaling, both in terms of speedup and memory usage, without the need for any distributed or parallel resources. Our work describes how substructures discovered locally on data partitions can be evaluated to determine the globally-optimal substructures. This approach requires the data to be partitioned. Some information is lost in the form of edges that are cut at the partition boundaries. We illustrate a method to recover this lost information. A full discussion of the experiments that demonstrate scalability of the serial partitioned version of Subdue in both artificially-generated datasets and a bioinformatics domain can be found in [2]. This paper describes a learning model we built to predict the amount of memory used during the execution of Subdue discovery algorithm for graphs from the protein database to illustrate that we can automatically deduce the ideal number of partitions into which a graph must be divided to ensure that each partition is small enough to fit in main memory.



## Overview Of Subdue

The Subdue system is a structural discovery tool that finds substructures in a graph-based representation of structural databases. Subdue operates by evaluating potential substructures for their ability to compress the entire graph. Once a substructure is discovered, the discovery is used to simplify the data by replacing instances of the substructure with a pointer to the newly discovered substructure definition. Repeated iterations of the substructure discovery and replacement process construct a hierarchical description of the structural data in terms of the discovered substructures. This hierarchy provides varying levels of interpretation that can be accessed based on the specific goals of the data analysis [1].

Subdue uses the Minimum Description Length Principle [3] as the metric by which graph compression is evaluated. Subdue is also capable of using an inexact graph match parameter to evaluate substructure matches, so that slight deviations between two patterns can be considered as the same pattern.

$$Compression = \frac{DL(S) + DL(G \mid S)}{DL(G)} \quad (1)$$

Equation 1 illustrates the compression equation used to evaluate substructures, where *DL(S)* is the description length of the substructure being evaluated, *DL(G|S)* is the description length of the graph as compressed by the substructure, and *DL(G)* is the description length of the original graph. The better a substructure performs, the smaller the compression ratio will be.

## Related Work

Related partitioning and sampling approaches have been proposed for scaling other types of data mining algorithms to large databases. The partition algorithm [4] makes two passes over an input transaction database to generate association rules. The database is divided into non-overlapping partitions and each of the partitions is mined individually to generate local frequent itemsets. We adapt some of the ideas of the partition algorithm to graph-based data mining. However, generally the graph cannot be divided into non-overlapping partitions as in the partition algorithm for generating association rules. The edges cut at the partition boundaries pose a challenge to the quality of discovery. The turbo-charging vertical mining algorithm [5] incorporates the concept of data compression to boost the performance of the mining algorithm. The FP-Tree algorithm [6] builds a special tree structure in main memory to avoid multiple passes over database. In an alternative approach, the sampling algorithm [7] picks a random sample to find all association rules that with high probability apply to the entire database, and then verifies the results with the rest of the database.

```
//Invoke serial Subdue on each partition Gj, which
//returns top b substructures for the jth partition
    for each partition Gj
        localBest[] = Subdue(Gj);
//Store local best substructures for global evaluation
    bestSubstructures[] =
        Union(bestSubstructures[],localBest[]);
-------------------------------------------------------------------------
//Reevaluate each locally-best substructure on all
//partitions
    sizeOfGraph = 0;
    for each substructure Si in bestSubstructures[]
        sizeOfSubSi =  MDL(Si);
        sizeCompressedGraph = 0;  //initialize
        for each partition Gj
         //size of graph (in bits) is the sum of sizes of
         //individual partitions
            sizeCompressedGraph =
                sizeCompressedGraph + MDL(Gj|Si);
            sizeOfGraph = sizeOfGraph + MDL(Gj);
//Calculate global value of substructure
    subValueSi = sizeOfGraph / (sizeOfSubSi +
                    sizeCompressedGraph);
    bestSubstructures[i].globalValue = subValueSi;
//Return the top b substructures in bestSubstructures[]
//as the top b global best substructures
```

Figure 1. SSP-Subdue Algorithm

In earlier work, a static partitioning algorithm was introduced [8] to scale the Subdue graph-based data mining algorithm using distributed processing. This type of parallelism is appealing in terms of memory usage and speedup. The input graph is partitioned into *n* partitions for *n* processors. Each processor performs Subdue on its local graph partition and broadcasts its best substructures to the other processors. A master processor gathers the results and determines the global best discoveries. However, this approach requires a network of workstations using communication software such as PVM or MPI. The knowledge discovered by each processor needs to be communicated to other processors. Our serial partitioning approach, implemented in the SSP-Subdue system, is unaffected by the communication problems of a distributed cluster as the partitions are mined one after the other on a single machine with the same processor playing the roles of slave and master processors in the static partitioning approach.

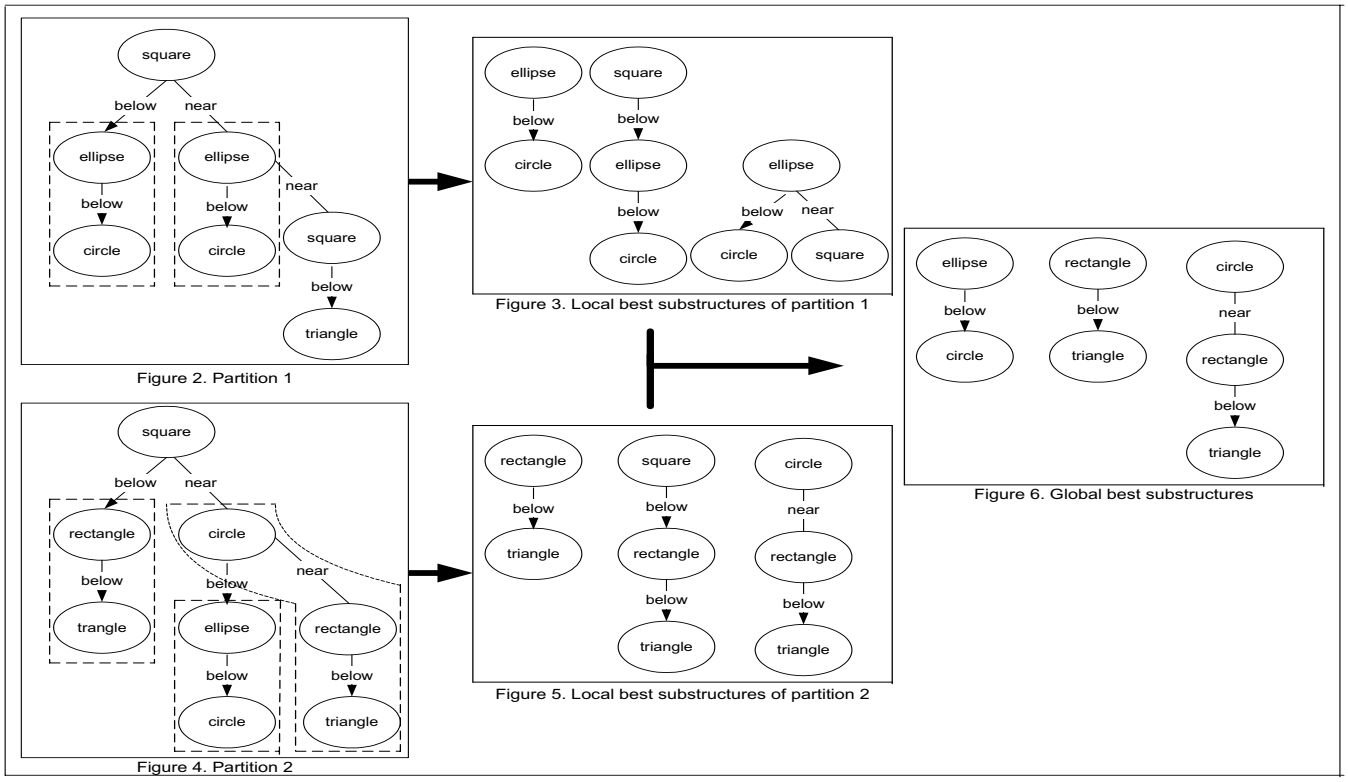
## Serial Static Partitioning Using SSP-Subdue

We have developed an algorithm that operates serially on smaller partitions of the graph and then compares the local results to acquire a measure of the overall best substructures for the entire graph.

The input graph is partitioned into *x* partitions. We perform Subdue on each partition and collect the *b* best substructures local to each partition in a list, where *b* is the beam used to constrain the number of best substructures reported. We take care that for each partition, Subdue reports only the substructures that have not already been reported as locally-best on any of the previously-processed partitions. By doing so, we implicitly increase the beam dynamically. At the end of this pass, there are *xb* substructures in the list. Then we evaluate these *xb* locally-best substructures on all partitions in a second pass over the static partitions, similar to the partition approach applied to association rule mining [3]. Once all evaluations are complete, we gather the results and determine the global best discoveries. This is a serial approach and does not rely on parallel hardware. Figure 1 summarizes the basic algorithm.

$$Compressio\,nRatio_{\,j}(S) = \frac{DL(S) + DL(G_j \mid S)}{DL(G_j)} \qquad (2)$$

As a part of this research, we have generated a variant of the MDL measure, which is used to rank discoveries globally. SSP-Subdue measures graph compression using our measure variant given in Equation 2, where DL(S) is the description length of the substructure S being evaluated, $DL(G_j|S)$ is the description length of the graph corresponding to the jth partition as compressed by substructure S, and $DL(G_j)$ is the description length of the uncompressed jth partition. The substructure that minimizes the sum of DL(S) and $DL(G_j|S)$ is the most descriptive substructure, and thus is locally the best. The smaller the value of the compression ratio of a




substructure, the higher will Subdue rank that substructure locally for the jth partition.

The global best substructures are found by reevaluating the locally best substructures using Equation 3 on the other partitions. Here, S is a substructure in the common list. The common list represents a collection of all local best substructures. The variable $x$ represents the number of partitions, DL(S) is the description length of the substructure S under consideration, $\sum^{x}_{j=1}DL(G_j|S)$ is the sum of description lengths of all the partitions after being compressed by the substructure S, and $\sum^{x}_{j=1}DL(G_j)$ is the description length of the entire graph. The substructure with the minimum value of the compression ratio obtained from Equation 3 is ranked as globally the best substructure.

$$Compression(S) = \frac{DL(S) + \sum_{j=1}^{x} DL(G_j \mid S)}{\sum_{j=1}^{x} DL(G_j)} \qquad (3)$$

The following example illustrates the SSP-Subdue algorithm concepts. For this example input graph is split into two partitions. Subdue is run on partition 1 shown in Figure 2 and the best substructures local to this partition, shown in Figure 3, are stored for global evaluation. Next, Subdue is run on partition 2 shown in Figure 4 and the best substructures local to this partition, shown in Figure 5, are stored for global evaluation. In a second pass over both of the static partitions, all of the locally-best substructures are evaluated using Equation 3 to produce the globally-best substructures shown in Figure 6. The instances of these globally best substructures are highlighted in the two partitions.

## Edge-loss Recovery Approach

The partitions are compressed using the globally best substructures found by running SSP-Subdue and then combined in pairs. Then the edges that were lost due to the original partitioning are reinserted between the combined partitions.

Since merging all possible combinations of two partitions that have edges cut between them could lead to a total of x(x-1)/2 combinations, each partition is constrained to be combined at most once with another partition. The pair of partitions that have the maximum number of edges cut between them are merged. Then the pair of partitions that have the second maximum number of edges cut between them are combined, and so on. This guarantees that two partitions are not combined unless they had any edges cut between them. However, this might sometimes lead to a matching such that some partitions are left that cannot be combined with any of the remaining unpaired partitions due to no edges cut at the boundaries. Here we are assuming that the compression and combining of partitions will not lead to a partition with a size too large to fit in dynamic memory. Finally, SSP-Subdue is executed on the combined partitions to get the globally-best substructures. The following example illustrates our approach. The input graph, showsn in Figure 7, is divided into two parts. As a result of this partitioning, all the instances of one of the

most frequently occurring substructures, "rectangle below triangle", are lost. After running SSP-Subdue on the partitions shown in Figure 8, the substructures illustrated in Figure 9 are reported as the global best substructures. The two partitions are compressed using the above substructures and combined to form the graph shown in Figure 10. After running SSP-Subdue on the compressed graph shown in Figure 10, the substructures in Figure 11 were reported as the best substructures. Clearly this set includes larger substructures encompassing the frequently-occurring substructure, "rectangle below triangle," which was initially lost due to the original partitioning. Thus, this approach proves useful in recovering the instances of those interesting substructures that are lost due to the original partitioning. However, a problem can occur when the best substructure is broken across partition boundaries, and subgraphs within this substructure are discovered in local partitions in different combinations with other subgraphs. The local discoveries would be used to compress the partitions and the original substructure will not be reformed and discovered in the second iteration. To remain consistent with the original Subdue algorithm, the compression could be performed using only the single best substructure found as opposed to the beam number of best substructures. Then the compressed subgraph would still appear as part of the original substructure and the best could be found.

## Learning Model to Deduce Number of Partitions

$N_p$, the ideal number of partitions for a given graph, is an important parameter for SSP-Subdue that affects the quality of discovery as well as the run time. As a result, the user would benefit from receiving a suggested value of $N_p$ from the system.

**Motivation for Employing a Learning Model**

The mechanism to find $N_p$ should be independent of the different versions and implementations of Subdue. Since Subdue is a main memory algorithm, $N_p$ depends on the amount of memory available for it to use during execution of the discovery algorithm after loading the input graph in memory. The amount of memory used during the execution of the discovery algorithm ($M_{used}$) is not a straightforward function of the size of the graph. It depends on several other factors directly related to the structure of the graph and those specific to Subdue's parameters used to constrain the search for interesting patterns in the input graph. Thus, the mechanism to deduce $M_{used}$, and in turn $N_p$, should be powerful enough to deduce the values based on all of the above factors.

Also, the actual amount of memory that a process is allowed to use ($M_{max}$) is not necessarily equal to the amount of main memory available on a machine. It is in fact dependent on various other factors like number of

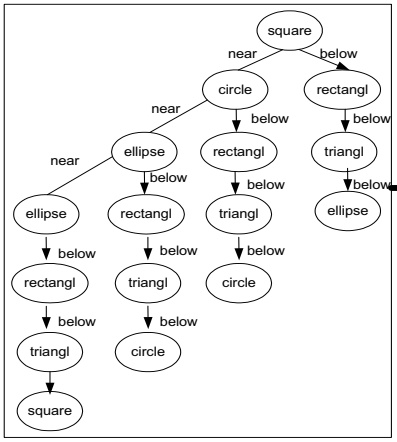


Figure 7. Unpartitioned graph G'

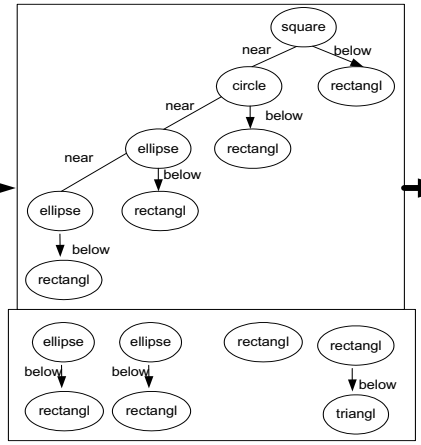


Figure 8. Partition 1 and 2 of graph G'

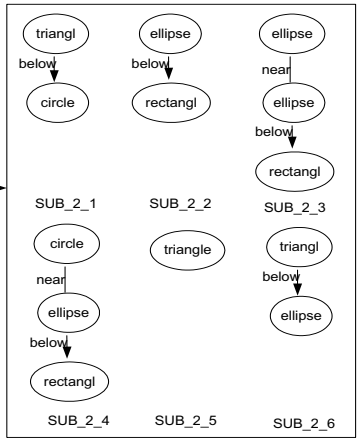


Figure 9. Global best substructures after first pass

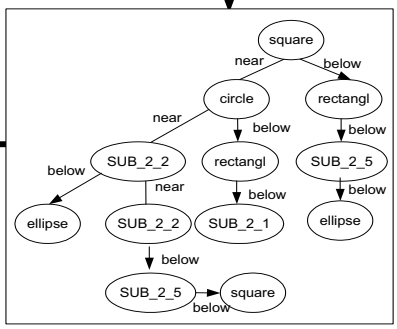


Figure 10. Compressed graph

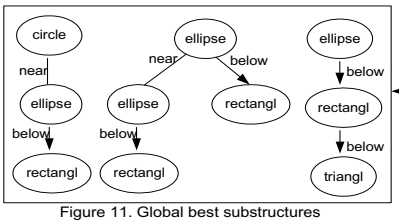


Figure 11. Global best substructures

processors running at a given time, the amount of main memory used by the operating system resident files and other limits and operating system parameters configured at the time of system installation and administration. Thus, it is unreasonable to consider a fixed value for $M_{max}$.

Our goal is to predict the memory usage for any given graph when run with any combination of Subdue-specific parameters to aid in deducing the optimal number of partitions for that graph. We hypothesized that we could build a learning model to predict the amount of memory used ($M_{used}$) during the execution of the Subdue discovery algorithm for a graph provided the learning model is trained with enough training example graphs from a particular domain. The value of $M_{used}$ could then be used to calculate $N_p$, the ideal number of partitions for a given graph. We successfully validated our hypothesis for a constrained case by experimenting with graphs from a particular domain.

## The Approach

Let *param1…paramN* represent the various parameters that govern the amount of memory used by Subdue. Then, we can use training examples with varying values of (*param1…paramN*) to build our learning model. This learning model can be used to predict the value of $M_{used}$ for a new graph. If $M_{used}$ exceeds $M_{max}$, then the graph needs to be partitioned.

With this new graph a set (*param1…paramN,* $M_{max}$) is constructed and input to a similar learning model that can predict $s_g$, the maximum size of graph that can be processed with $M_{max}$, the amount of memory available for use by Subdue discovery algorithm. Now if the original graph to be partitioned is of size $S$, then the value of $N_p$, the number of partitions, can be calculated as $S/s_g$.

We used Weka [9], a data mining software package, to build our learning model. $M_{max}$ can be estimated from the amount of free memory reported by the 'free' command of the Unix operating system.

**Features Influencing Memory Usage**

The features related to the structure of the input graph that influence the memory usage of the discovery algorithm are total number of vertices, total number of edges, total number of directed edges, total number of undirected edges, total number of unique vertex labels, total number of unique edge labels, variance in degree of vertices or connectivity, total number of disconnected graphs making up the input graph and compressive capacity of the best substructures.

The Subdue parameters that influence the amount of memory used by the discovery algorithm are the beam width of Subdue discovery algorithm, the number of different substructures considered in each iteration of the Subdue discovery algorithm, the maximum number of vertices that can exist in a reported substructure, the minimum number of vertices that can exist in a reported substructure, the method used for evaluating candidate substructures, the number of iterations of the Subdue discovery algorithm that will be executed on the input graph, the fraction of the size of an instance by which the instance can be different from the substructure definition.

Varying the above parameters in all different possible combinations would require a large number of experimental tests. Besides, in most practical cases, most of the default Subdue parameters are used or one set of parameter values is used for all graphs from the same domain. Hence, we restricted our tests to some of the features related to the structure of the graph namely number of vertices and edges, number of directed and undirected edges, number of unique vertex and edge labels and variance in degree of vertices while keeping the Subdue-specific parameters set to their default values.

Further, our initial attempt to learn Subdue's memory requirement for a mix of graphs from different domains showed poor performance (about 15% predictive accuracy). This indicated that graphs from different domains are vastly different in their memory requirements and hence pose a very challenging job for the learning model. Thus, we restricted our tests to graphs from a single domain to see if the hypothesis can be validated for a constrained portion of the whole problem.

**Protein Database.** The Protein Data Bank (PDB) is a worldwide repository for processing and distributing 3-D data structures for large molecules of proteins and nucleic acids. We converted the information in the given PDB file to a Subdue-formatted graph file corresponding to the compound described in the PDB file. Since we were mainly concerned with experimenting on graphs of varying sizes, the files from PDB used for our experiments were selected randomly and inclusion of no particular chemical compound was emphasized. We browsed the database to obtain the graphs of the required sizes.

**Use of Decision Trees for Prediction**

Decision trees represent a supervised approach to classification. A decision tree is a simple structure where non-terminal nodes represent tests on one or more attributes and terminal nodes reflect decision outcomes.

We used Weka to apply the *J48* learning method to the PDB dataset and analyze its output. The J48 algorithm is Weka's implementation of the *C4.5* decision tree learner. C4.5 is a landmark system for decision tree induction devised by Ross Quinlan.

We prefer to use decision trees over other classifiers since they have a simple representational form, making the inferred model relatively easy for the user to comprehend. Other classifiers like neural networks, although a powerful modeling tool, are relatively difficult to understand compared to decision trees. Decision trees can classify both categorical and numerical data, but the output attribute must be categorical. There are no prior assumptions made about the nature of the data. However, decision tree algorithms are unstable. Slight variations in the training data can result in different attribute selections at each choice point within the tree. The effect can be significant since attribute choices affect all descendent subtrees. Trees created from numeric data sets can be quite complex since attribute splits for numeric data are binary.

## Experimental Results

We randomly chose a set of 60 PDB datasets and converted them into Subdue-format graph files. All the relevant information required for populating Weka's input file including number of vertices and edges, number of directed and undirected edges, number of unique vertex and edge labels, variance in degree of vertices and memory used for processing was calculated from each of the graphs. This comprised the training set for our learning model. The classes defined in the input file were the memory used for the processing of these graphs by Subdue (i.e., 1MB, 2MB, 3MB, 4MB, 5MB, 6MB, 8MB, 10MB, 12MB, 14MB, 18MB, and 22MB), resulting in 12 possible classes. We used a test set comprising of 30 randomly-selected graphs, with random class distribution, from the training set to evaluate our learning model. On applying the J48 algorithm to the PDB dataset, the J48 pruned tree (in text format), built using the training set, was obtained along with an estimate of the tree's predictive performance. The test data derived the performance statistics. 24 test instances (80%) were classified correctly and 6 (20%) were misclassified out of a total of 30 instances. In addition, the following measurements were derived from the class probabilities assigned by the tree.

- Kappa statistic 0.765
- Mean absolute error 0.0347
- Root mean squared error 0.1323
- Relative absolute error 23.9333 %
- Root relative squared error 49.3356 %

A kappa statistic of 0.7 or higher is generally regarded as good statistic correlation. In all of these error measurements, a lower value means a more precise model, with a value of 0 depicting the statistically-perfect model.

**Learning Curve.** The learning model was trained on random subsets of varying sizes (5, 10, 15... 55, 60) from the 60 PDB graphs. A learning curve was plotted by recording the learning model's percentage accuracy in prediction of the memory used by the graphs in the test set for each such training set. The learning curve, shown in Figure 12, was found to plateau at 80% predictive accuracy.

## Conclusions

This research proposes an effective solution, in the form of a serial partitioning approach, to one of the main challenges for graph-based knowledge discovery and data mining systems, which is to scale up their data interpretation abilities to discover interesting patterns in large datasets without the need of any distributed or parallel resources. It also illustrates how information lost in the form of edges that are cut at the partition boundaries can be recovered and how the optimal number of partitions into which a graph needs to be divided into can be deduced automatically.

The performance of the learning model on a constrained portion of the complete problem was encouraging enough

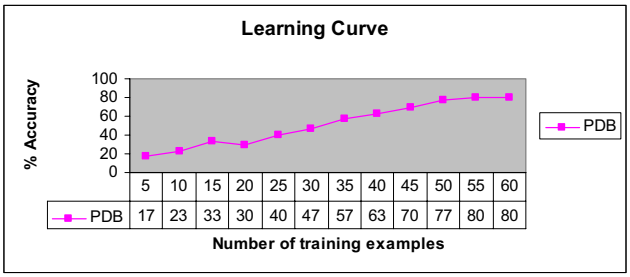



Figure 12. Learning Curve for graphs from PDB with a test set of 30 graphs.

to conclude that the system could be made to learn to predict the memory usage for a fresh graph similar to those it was trained on. One could enhance the model to include graphs from other domains but since graphs from different domains are vastly different in their memory requirements, they pose a very challenging job for the learning model. To learn a truly generic model, an exhaustive training set would be required comprising of all types of graphs from all possible different domains. The learning model will have better prediction accuracy when trained on graphs from a single domain since graphs from the same domain are similar in their memory requirements.

## References


[1] D. J. Cook and L. B. Holder, Graph-Based Data Mining, *IEEE Intelligent Systems*, 15(2), pages 32-41, 2000.

[2] Coble, J., Rathi, R., Cook, D., Holder, L. Iterative Structure Discovery in Graph-Based Data. To appear in the International Journal of Artificial Intelligence Tools, 2005.

[3] Cook, D. and Holder, L. 1994. Substructure Discovery Using Minimum Description Length and Background Knowledge. In *Journal of Artificial Intelligence Research,* Volume 1, pages 231-255.

[4] Savasere, A., E. Omiecinsky, and S. Navathe. An efficient algorithm for mining association rules in large databases. *21st Int'l Cong. on Very Large Databases (VLDB)*. 1995. Zurich, Switzerland.

[5] Shenoy, P., et al. Turbo-charging Vertical Mining of Large Databases. *ACM SIGMOD Int'l Conference on Management of Data*. 2000. Dallas.

[6] Han, J., J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. *ACM SIGMOD Int'l Conference on Management of Data*. 2000. Dallas.

[7] Toivonen, H. Sampling Large Databases for Association Rules. In Proc. 1996 *Int. Conf. Very Large Data Bases*. 1996: Morgan Kaufman.

[8] D. J. Cook, L. B. Holder, G. Galal, and R. Maglothin, Approaches to Parallel Graph-Based Knowledge Discovery, *Journal of Parallel and Distributed Computing*, 61(3), pages 427-446, 2001.

[9] Ian H. Witten and Eibe Frank, Data Mining: Practical Machine Learning Tools with Java Implementations, *Morgan Kaufmann*, San Francisco, 2000.