ITERATIVE STRUCTURE DISCOVERY IN GRAPH-BASED DATA

JEFFREY A. COBLE, 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
{coble, rathi, cook, holder}@cse.uta.edu

Received
Revised
Accepted

Much of current data mining research is focused on discovering sets of attributes that discriminate data entities into classes, such as shopping trends for a particular demographic group. In contrast, we are working to develop data mining techniques to discover patterns consisting of complex relationships between entities. Our research is particularly applicable to domains in which the data is event-driven or relationally structured. In this paper we present approaches to address two related challenges; the need to assimilate incremental data updates and the need to mine monolithic datasets. Many realistic problems are continuous in nature and therefore require a data mining approach that can evolve discovered knowledge over time. Similarly, many problems present data sets that are too large to fit into dynamic memory on conventional computer systems. We address incremental data mining by introducing a mechanism for summarizing discoveries from previous data increments so that the globally-best patterns can be computed by mining only the new data increment. To address monolithic datasets we introduce a technique by which these datasets can be partitioned and mined serially with minimal impact on the result quality. We present applications of our work in both the counter-terrorism and bioinformatics domains.

Keywords: Structural Data Mining; Incremental Discovery; Graph Partitioning; Machine Learning.

1. Introduction

Much of current data mining research is focused on algorithms that can discover sets of attributes that discriminate data entities into classes, such as shopping or banking trends for a particular demographic group. In contrast, we are working to develop data mining techniques to discover patterns consisting of complex relationships between entities. Our research is particularly applicable to domains in which the data is event driven, such as counter-terrorism intelligence analysis, and domains where the only distinguishing characteristics are relational, like molecular structures. Analytical tasks require discovery of relational patterns between events and actors so that these patterns can be exploited for the purposes of prediction and action. Similarly, identifying characteristic molecular structures is necessary to acquire a foundational understanding of important research problems in many of the basic sciences. Problems of such complexity often present two

related challenges; the need to assimilate incremental data updates and the need to mine monolithic datasets.

Many challenging problems, including those in the counter-terrorism domain, require processing and assimilation of periodic increments of new data, which provides new information in addition to that which was previously processed. Our approach provides a mechanism for summarizing discoveries from previous data increments so that the globally best patterns can be computed by examining only the new data increment.

The second challenge we are addressing is the scalability of graph-based discovery to monolithic datasets, which are prevalent in domains like bioinformatics, where vast amounts of data must be examined to find meaningful structures. The algorithms used for data interpretation in graph-based knowledge discovery and data mining systems are generally computationally expensive. 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. In this paper we describe a technique by which large datasets can be segmented and processed serially with minimal impact on the result quality.

Fortunately, these two challenges rely on many of the same mathematical concepts and algorithm techniques.

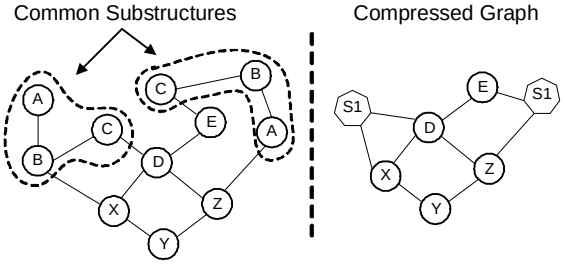


Figure 1. Subdue discovers common substructures within relational data by evaluating their ability to compress the graph.

2. Structure Discovery

The work we describe in this paper is based on Subdue,[1] which is a graph-based data mining system designed to discover common structures from relational data. Subdue represents data in graph form and can support either directed or undirected edges. Subdue operates by evaluating potential substructures for their ability to compress the entire graph, as illustrated in Figure 1. The better a particular substructure describes a graph, the more the graph will be compressed by replacing that substructure with a placeholder vertex. Repeated iterations will discover additional substructures, potentially those that are hierarchical, containing previously compressed substructures.

Subdue uses the Minimum Description Length Principle[2] 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.

Subdue provides two algorithms for calculating the description length of a graph. The first takes a more comprehensive view of the physical representation of the graph components and consists of the number of bits needed to encode the vertex labels, the adjacency matrix, the number of edges between vertices, and the edge labels. The second is more simplistic, relying on the number of vertices and edges as a measure of size. For the work in this paper, we have used this latter encoding. We refer the reader to [Cook and Holder 1994][3] for a full discussion of the MDL computation used by Subdue to encode graphs.

Subdue's evaluation algorithm ranks the best substructures by measuring the inverse of the compression value in Equation 1. Favoring larger values serves to pick a substructure that minimizes *DL(S) + DL(G|S)*, which means we have found the most descriptive substructure.

3. Research

In this paper we introduce two new algorithms, Incremental-Subdue (I-Subdue) and Serial-Static-Partitioning-Subdue (SSP-Subdue). For our work on I-Subdue, we assume that data is received in incremental blocks as is the case for many long-term analytical tasks. Continuously reprocessing the accumulated graph after each increment would be intractable, so instead we wish to develop methods to iteratively refine the substructure discoveries with a minimal amount of reexamination of old data so that the globally-best patterns can be identified based on previous local discoveries.

For our work on SSP-Subdue, 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. The baseline Subdue algorithm discovers prevalent patterns that best represent the structure of a graph-based dataset, but requires a substantial amount of processing time and dynamic memory for the types of datasets generally found in realistic problem domains. This work describes how substructures discovered locally on data partitions can be evaluated to determine the globally-optimal substructures.

3.1 Related work

3.1.1 Online learning

The online sequential learning[4] problem presents many similarities to the incremental discovery problem with which we are concerned. Since the learner is receiving its training data sequentially, it must repeatedly apply the learning algorithm until it is satisfied that it has converged to a good model of the world. This means that it has to store all of the previously encountered training vectors in some usable form, rather than reapply the learner to the aggregate dataset after each new data increment is received. Due to the enormity of the datasets, some summarization technique must be used that does not sacrifice valuable information. This summarization technique must also be selected so that it does not inappropriately bias the next learning iteration toward a previously learned model. Friedman and Goldszmidt[5] address this issue in their work on sequentially updating the structure of Bayesian Networks.

3.1.2 Mining sequential patterns

Research in mining sequential patterns[6] is still largely about finding patterns in item-set data, unlike the structural datasets that we are investigating in this research. However, there are similarities in that the objective of mining sequential patterns is to look for time-sequenced transaction patterns, such as a series of movie rentals or consumer purchases. This differs from traditional item-set data mining, which is generally concerned with finding intra-transaction patterns. Our goal for our pattern discovery work includes the additional constraint that transactional and event relationships are emerging over time and so we must be able to evolve the fundamental structure of our discovered patterns. The existing work on item-set data considers all of the transaction data in its entirety and looks for sequential patterns within it.

3.1.3 Online data mining

There is a body of emerging research related to online data mining.[7,8,9] However, much of this work is related to online machine learning research in that the focus is more on identifying the point of stability in the discovered trend or concept and in dealing with changing systems in the form of concept drift or shift. This is an important problem for our work as well and is the focus of continuing research. However, by restricting the research to item-set data, which is assumed to arrive in complete and independent units, current approaches are able to largely ignore issues related to event and transaction relationships that emerge over time. Although we make a similar assumption in the work presented here for the purposes of illustrating effective summarization techniques, our ongoing work is addressing the issue of sequentially connected data, where relationships extend across temporal data boundaries. This is a critical issue for applications of structural data mining.

3.1.4 Parallel graph-based knowledge discovery

Several related partitioning and sampling approaches have been proposed in existing association rule mining research[10,11,12,13] but generally a graph cannot be divided into non-overlapping partitions as is the case for association rules. The edges cut at the partition boundaries pose a challenge to the quality of discovery. In earlier work, a static partitioning algorithm[14] was introduced 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. 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 globally best discoveries. However, this approach requires a network of computers using communication software such as PVM or MPI. Our serial partitioning approach, implemented in the SSP-Subdue system, does not require a system of specialized distributed or parallel hardware. Instead, 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.

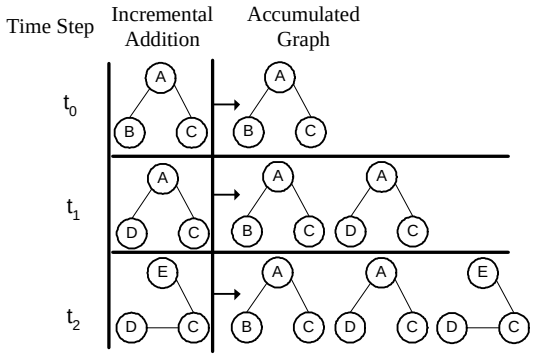


Figure 2. Independent data received incrementally can be viewed as a unique extension to the accumulated graph.

3.2 Incremental Subdue

For this work, we view each new data increment as a distinct graph structure. Figure 2 illustrates one conceptual approach to mining sequential data, where each new increment received at time step t_i is considered independently of earlier data increments so that the accumulation of these structures is viewed as one large, but disconnected, graph. The original Subdue algorithm would still work equally well if we applied it to the accumulated graph after each new data increment is received. The obstacle is the computational burden required for repeated full batch processing.

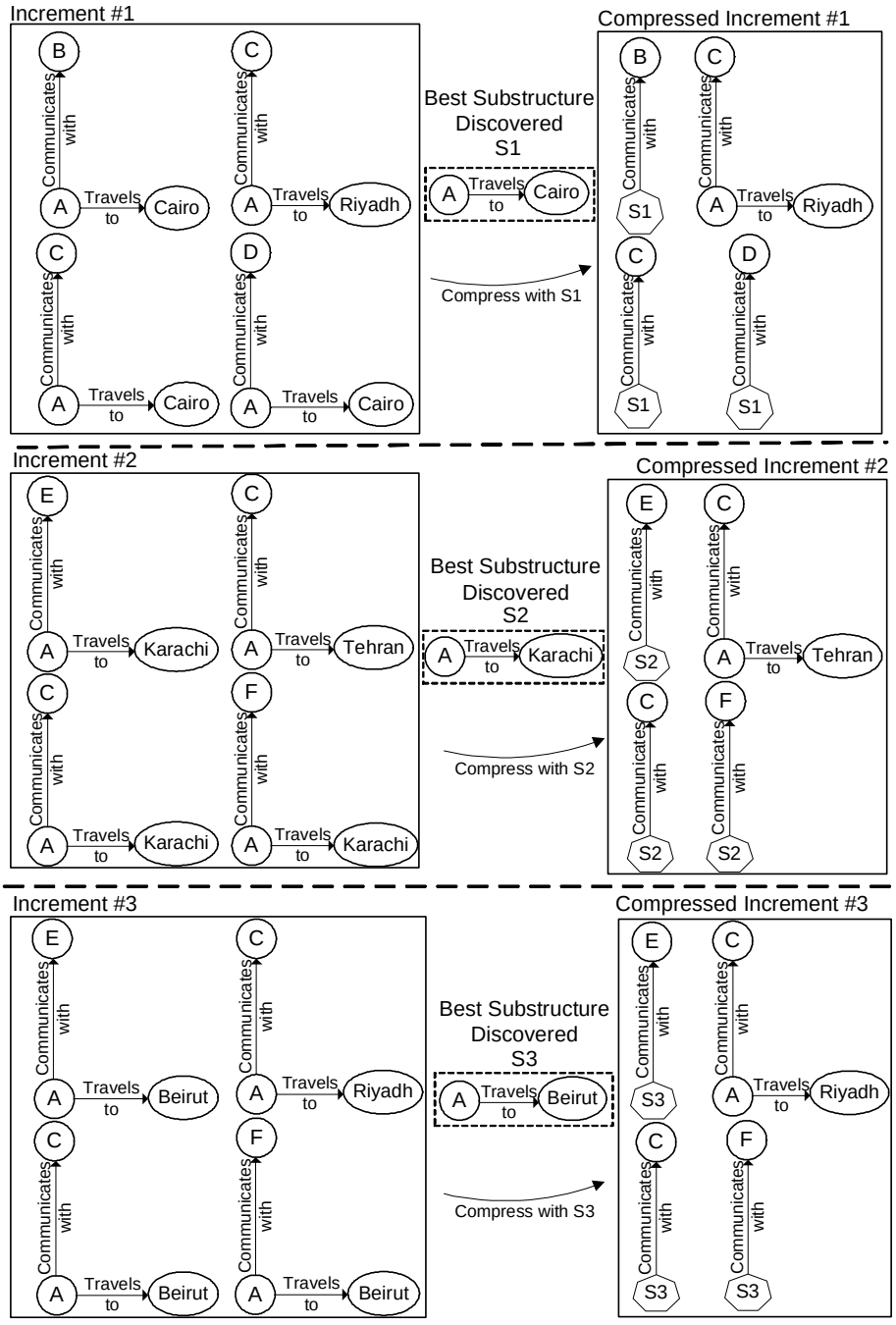


Figure 3. Three data increments received serially and processed individually by Subdue. The best substructure is shown for each local increment.

It is easy to see how the concept depicted in Figure 2 can be applied to real problems. For instance, a software agent deployed to assist an intelligence analyst would gradually build up a body of data as new information streams in over time. This streaming data could be viewed as independent increments from which common structures are to be derived. Although the data itself may be generated in very small increments, we would expect to accumulate some minimum amount before we mine it. Duplicating nodes and edges in the accumulated graph serves the purpose of giving more weight to frequently repeated patterns.

3.2.1 Sequential discovery

Storing all accumulated data and continuing to periodically repeat the entire structure discovery process is intractable both from a computational perspective and for data storage purposes. Instead we wish to devise a method by which we can discover structures from the most recent data increment and simultaneously refine our knowledge of the globally best substructures discovered so far.

However, we can often encounter a situation where sequential applications of Subdue to individual data increments will yield a series of locally best substructures that are not the globally best substructures, which would be found assuming the data could be evaluated as one aggregate block.

Figure 3 illustrates an example where Subdue is applied sequentially to each data increment as it is received. At each increment Subdue discovers the best substructure for the respective data increment, which turns out to only be locally best. However, if we aggregate the same data as depicted in Figure 4 and then apply the baseline Subdue algorithm we get a different best substructure, which in fact is globally best. This is illustrated in Figure 5. Although our simple example could easily be aggregated at each time step, realistically large data sets would be too unwieldy to do so.

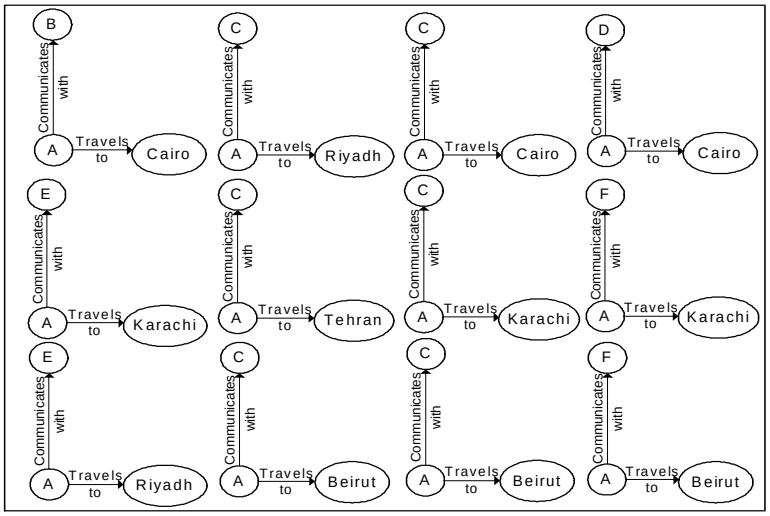


Figure 4. Accumulated graph for Subdue batch processing

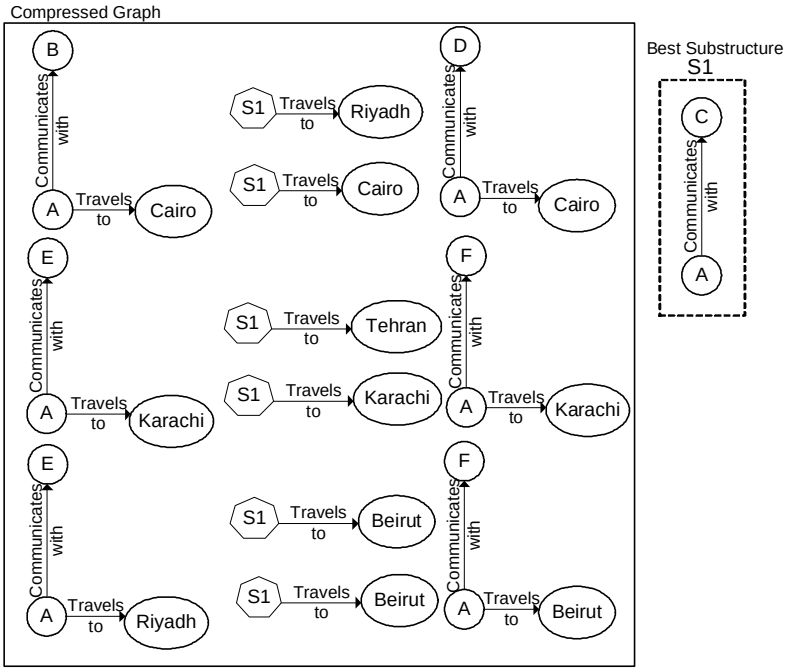


Figure 5. Result from applying Subdue to the three aggregated data increments.

In general, sequential discovery and action brings with it a set of unique challenges, which are generally driven by the underlying system that is generating the data from which structures are discovered. One problem that is almost always a concern is how to reevaluate the accumulated data at each time step in light of newly added data. There is generally a tradeoff between the amount of data that can be stored and reevaluated and the quality of the result. A summarization technique is usually employed to capture salient metrics about the data. The richness of this summarization is a tradeoff between the speed of the incremental evaluation and the range of new substructures that can be considered.

3.2.2 Summarization metrics

Our goal for this research is to develop a summarization metric that can be maintained from each incremental application of Subdue that will allow us to derive the globally best substructure without reapplying Subdue to the accumulated data.

To accomplish this goal, we rely on a few artifacts of Subdue's discovery algorithm. First, Subdue maintains a list of the *n* best substructures discovered from any dataset, where *n* is configurable by the user. The default value for *n* is 3, but any number of ranked substructures can be maintained, limited only by constraints on the beam search that Subdue uses to prune its search space.

Second, we use the value metric Subdue maintains for each substructure. Subdue measures graph compression with the Minimum Description Length principle as described in section 2. For I-Subdue, we must use a modified version of the compression metric to find the globally best substructure, illustrated in Equation 2.

$$Compress_m(S_i) = \frac{DL(S_i) + \sum_{j=1}^{m} DL(G_j \mid S_i)}{\sum_{j=1}^{m} DL(G_j)} \tag{2}$$

With Equation 2 we calculate the compression achieved by a particular substructure, S_i, up through and including the current data increment m. The $DL(S_i)$ term is the description length of the substructure, S_i, under consideration. The term

$$\sum_{j=1}^{m} DL(G_j \mid S_i)$$

represents the description length of the accumulated graph after it is compressed by the substructure S_i.

Finally, the term

$$\sum_{j=1}^{m} DL(G_j)$$

represents the full description length of the accumulated graph.

$$arg\ max(i)\left[\frac{\sum_{j=1}^{m} DL(G_j)}{DL(S_i) + \sum_{j=1}^{m} DL(G_j \mid S_i)}\right] \tag{3}$$

At any point we can then reevaluate the substructures using Equation 3 (inverse of Equation 2), choosing the one with the highest value as globally best.

The process of computing the global substructure value takes place in addition to the normal operation of Subdue on the isolated data increment. We only need to store the requisite description length metrics after each iteration for use in our global computation.

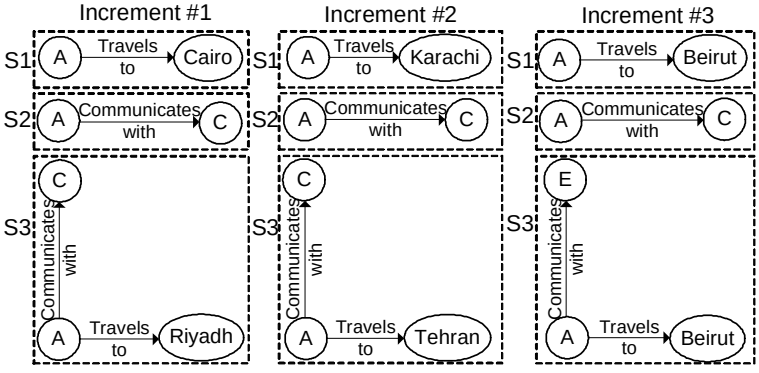


Figure 6. The top n=3 subs from each local increment.

Table 1. Substructure values computed independently for each iteration. Highlighted cells indicate maximum values in each increment.

Increment #	New Substructures from Increment #1			New Substructures from Increment #2		New Substructures from Increment #3	
	S_{11}	S_{12}	S_{13}	S_{21}	S_{23}	S_{31}	S_{33}
1	1.2182	1.04808	0.9815				
2		1.04808		1.21882	0.981511		
3		1.03804				1.15126	0.966017

Table 2. Using I-Subdue to calculate the global value of each substructure. The description length of each graph iteration (G_j) and of each substructure (S_i) are shown. Highlighted cells indicate the global best substructure at each increment.

Global Best Calculation After Iteration #	New Substructures from Iteration #1			New Substructures from Iteration #2		New Substructures from Iteration #3		
	S_{11}	S_{12}	S_{13}	S_{21}	S_{23}	S_{31}	S_{33}	$DL(G_j)$*
1	1.2182	1.04808	0.981511					117
2	1.0983	1.1235	0.9906	1.0986	0.9906			117
3	1.0636	1.1474	0.9937	1.0638	0.9937	1.0455	0.9884	116
$DL(S_i)$*	15	15	25.75489	15	25.754888	15	26.50978	

*measured in bits

As an illustration of our approach, consider the results from the example depicted in Figure 3. The top *n=3* substructures from each iteration are shown in Figure 6. Table 1 lists the values returned by Subdue for the local top *n* substructures discovered in each increment. The second best substructures in increments 2 and 3 (S_{22}, S_{32}) are the same as the second best substructure in increment 1 (S_{12}), which is why the column corresponding to S_{12} has a value for each increment. The values in Table 1 are the result of the compression evaluation metric from Equation 1. The locally best substructures illustrated in Figure 3 have the highest values, demarcated by the highlighted cells in Table 1.

Table 2 depicts our application of I-Subdue to the increments from Figure 3. After each increment is received, we apply Equation 3 to select the globally best substructure. The values in Table 2 are the inverse of the compression metric from Equation 2. As an example, the calculation of the compression metric for substructure S_{12} after iteration 3 would be:

$$\frac{DL(S_{12}) + DL(G_1 \mid S_{12}) + DL(G_2 \mid S_{12}) + DL(G_3 \mid S_{12})}{DL(G_1) + DL(G_2) + DL(G_3)}$$

Consequently the value of S_{12} would be:

$$\frac{117 + 117 + 116}{15 + 96.63 + 96.63 + 96.74} = 1.1474$$

For this computation we rely on the metrics computed by Subdue when it evaluates substructures in a graph, namely the description length of the discovered substructure, the

description length of the graph compressed by the substructure, and the description length of the graph. By storing these values after each increment is processed, we can retrieve the globally best substructure using Equation 3. Figure 7 illustrates the basic algorithm, where Subdue is invoked to discover the candidate substructures and the byproduct evaluation metrics are collected and used to calculate the globally best substructures after each new data increment is processed.

In circumstances where a specific substructure is not present in a particular data increment, such as S_{31} in iteration 2, then

$$DL(G_2 \mid S_{31}) = DL(G_2)$$

and the substructure's value would be calculated as follows:

$$\frac{117+117+116}{15+117+117+85.76} = 1.0455$$

```
//Call I-Subdue on the new data increment Gj
I-Subdue(Gj)
  //Subdue returns description length values and top n substructures for current data increment,
  //which are stored for global calculations
  CandidateSubstructures[], SubstructureSizes[], CompressedGraphSizes[], size_Gj ⇐ Subdue(Gj)
  total_graph_size = total_graph_size + size_Gj
/*************************************************************************/
Get_Global_Best(total_graph_size,CandidateSubstructures[], SubstructureSizes[],
CompressedGraphSizes[])
  best_value = 0
  global_best_substructure = nil
  for(i=1 to sizeof(CandidateSubstructures))
    size_si = CandidateSubstructureSizes[i]
          compressed_graph_size = 0
    for(j=1 to num_data_increments)
      compressed_graph_size = compressed_graph_size +
      CompressedGraphSizes[i][j])     //DL(Gj|Si)
    value_si = graph_size/(size_si + compressed_graph_size)
    if value_si > best_value
        best_value = value_si
        global_best_substructure = CandidateSubstructures[i]
  return global_best_substructure
```

Figure 7. Application of I-Subdue to store metrics returned from running Subdue over a single data increment, then calculating the global best substructure using the collected metrics.

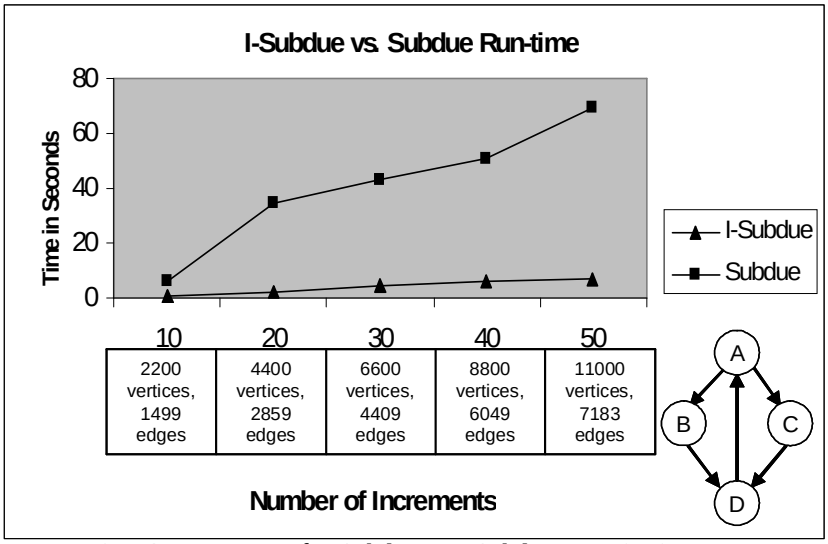


Figure 8. Comparison of I-Subdue vs. Subdue on 10-50 increments. Each increment provides 220 new vertices with 0 or 1 outgoing edges.

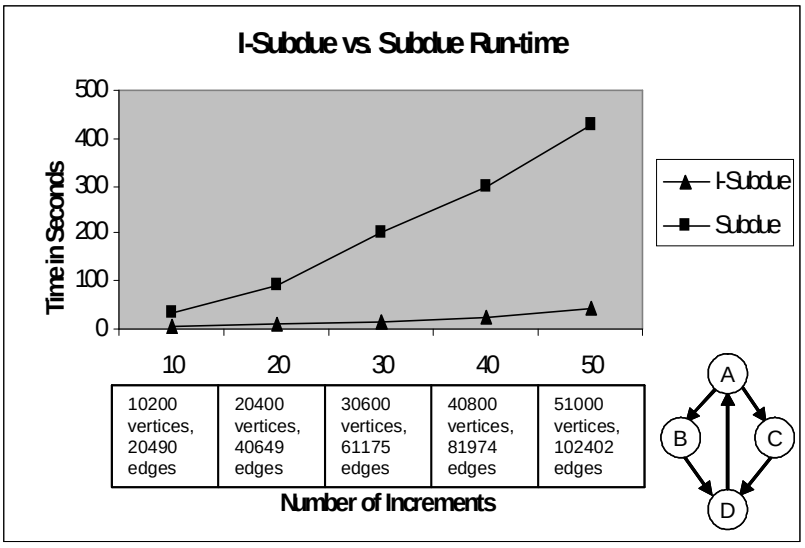


Figure 9. Comparison of I-Subdue vs. Subdue on 10 – 50 increments. Each increment provides 1020 new vertices with 1 to 4 outgoing edges.

3.2.3 Experimental evaluation

To illustrate the relative value of I-Subdue with respect to performance in processing incremental data, we have conducted experiments with a synthetic data generator. This data generator takes as input a library of data labels, configuration parameters governing the size of random graph patterns, and one or more specific substructures to be embedded within the random data. The random graph segments can also be configured in terms of the density of the edge connectivity. The data generator produces a new data increment when invoked by I-Subdue and for comparison purposes archives the cumulative set of increments into a file for batch processing by Subdue.

For the first experiment, illustrated in Figure 8, we compared the performance of I-Subdue versus Subdue at benchmarks ranging from 10 to 50 increments. Each increment introduced 220 new vertices, within which five instances of the four-vertex substructure pictured in Figure 8 were embedded. The quality of the result, in terms of the number of instances found, was the same for both I-Subdue and Subdue. The running times were the only discernable qualitative difference. The major simplifying factor for this first experiment was the vertex degree. Each vertex had 0 or 1 outgoing edges with a 50% probably for each. A less densely connected graph greatly simplifies the substructure search space.

The results from the second experiment are depicted in Figure 9. For this experiment, we increased the increment size to 1020 vertices and gave each vertex an outward degree ranging from 1 to 4 edges. Each degree value was chosen with 25% probability, which means that on average there are about twice as many edges as vertices. This more densely connected graph begins to illustrate the significance of the run-time difference between I-Subdue and Subdue. Again, five instances of the four-vertex substructure shown in Figure 9 were embedded within each increment. The discovery results were the same for both I-Subdue and Subdue with the only qualitative difference being the running time.

3.3 Partitioned Discovery

For this work we are concerned with addressing large, monolithic datasets, which are prevalent in many real-world domains. To address this challenge, 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. We also illustrate a method to recover information lost in the form of edge cuts from the graph partitioning. We illustrate the scalability with results from experiments performed using protein databases and artificial datasets.

In SSP-Subdue we partition the input graph into *x* partitions. The value of x here is selected by the user to ensure that each partition is small enough to fit in dynamic memory. 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[10]. Once all evaluations are

```
//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 10. SSP-Subdue Algorithm

complete, we gather the results and determine the global best discoveries. This is a serial approach and does not rely on parallel hardware. Figure 10 summarizes the basic algorithm and the metric used for evaluating substructures globally is described below.

As a part of this research, we have generated a variant of the MDL measure, described in section 2, which is used to rank discoveries globally.

SSP-Subdue measures graph compression using our measure variant given in Equation 4, 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.

$$CompressionRatio_j(S) = \frac{DL(S) + DL(G_j \mid S)}{DL(G_j)} \qquad (4)$$

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 5 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.

$$Compression(S) = \frac{DL(S) + \sum_{j=1}^{x} DL(G_j \mid S)}{\sum_{j=1}^{x} DL(G_j)} \qquad (5)$$

The substructure with the minimum value of the compression ratio obtained from Equation 5 is ranked as globally the best substructure.

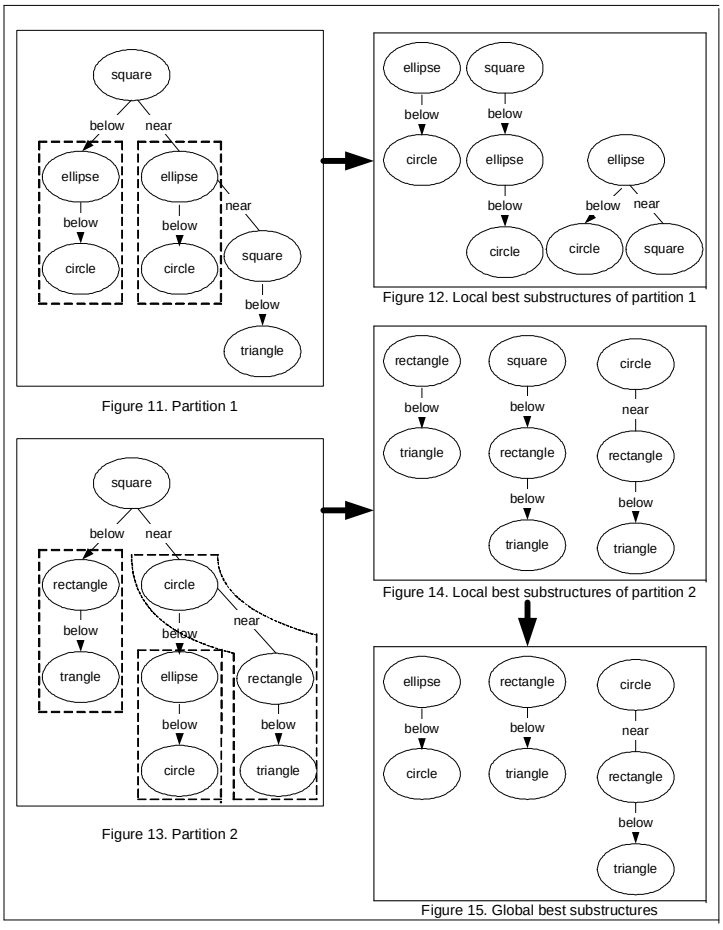


Figure 11. Partition 1

Figure 12. Local best substructures of partition 1

Figure 13. Partition 2

Figure 14. Local best substructures of partition 2

Figure 15. Global best substructures

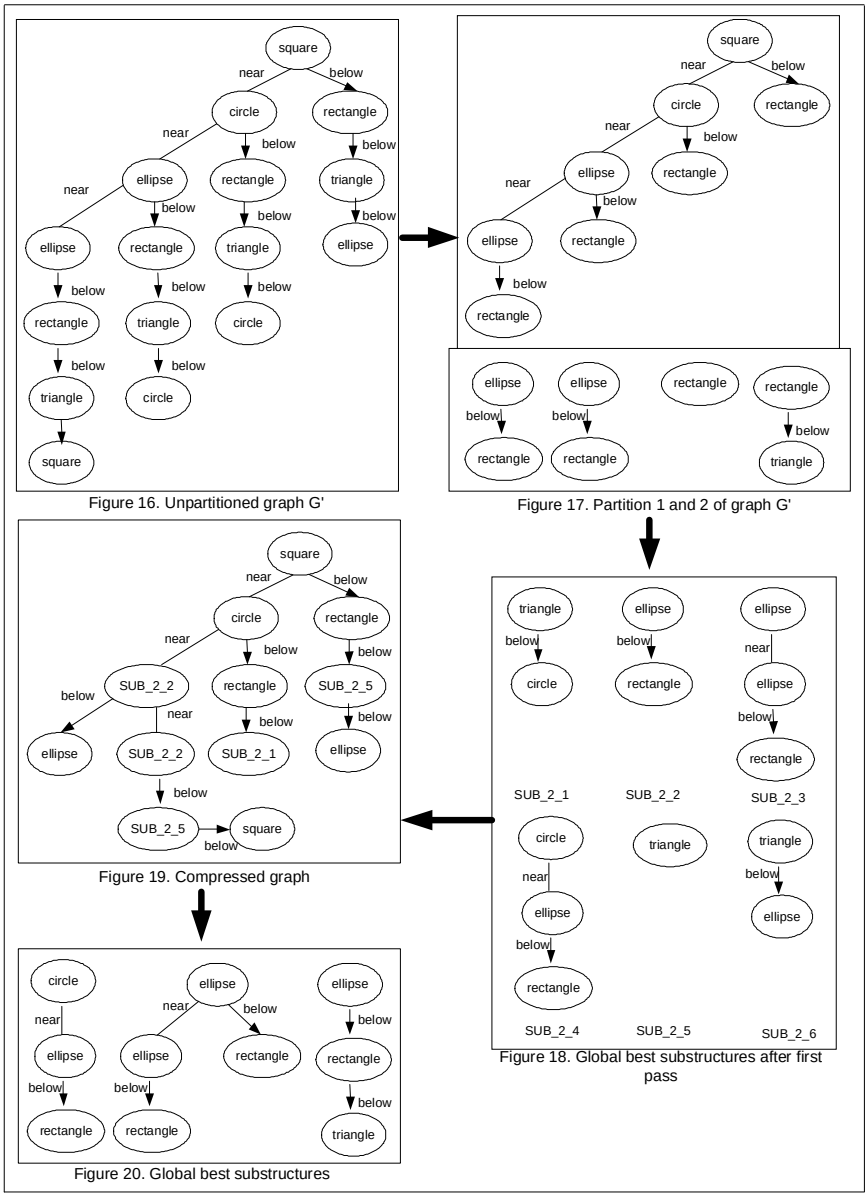


Figure 16. Unpartitioned graph G'

Figure 17. Partition 1 and 2 of graph G'

Figure 19. Compressed graph

Figure 18. Global best substructures after first pass

Figure 20. Global best substructures

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 11 and the best substructures local to this partition, shown in Figure 12, are stored for global evaluation. Next, Subdue is run on partition 2 shown in Figure 13 and the best substructures local to this partition, shown in Figure 14, 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 5 to produce the globally-best substructures shown in Figure

15. The instances of these globally best substructures are highlighted in the two partitions.

3.3.1 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, shown in Figure 16, 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 17, the substructures illustrated in Figure 18 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 19

After running SSP-Subdue on the compressed graph shown in Figure 19, the substructures in Figure 20 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. However, this problem can still exist in other forms. As shown in Figure 21, suppose that ellipse→rectangle→triangle is the desired substructure (the same label is applied to every

edge here), and appears half the time as ellipse→rectangle→triangle→square and the other half as ellipse→rectangle→triangle→oval. If one boundary occurs between ellipse and rectangle and another between rectangle and triangle, it is possible that triangle→square or triangle→oval will be the best local discoveries, and after compression the overall best substructure will still not be found as half of the triangles are now parts of the compressed substructure.

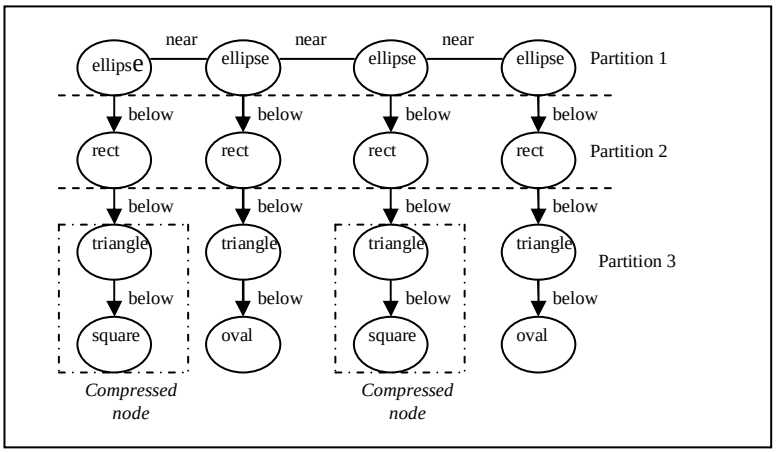


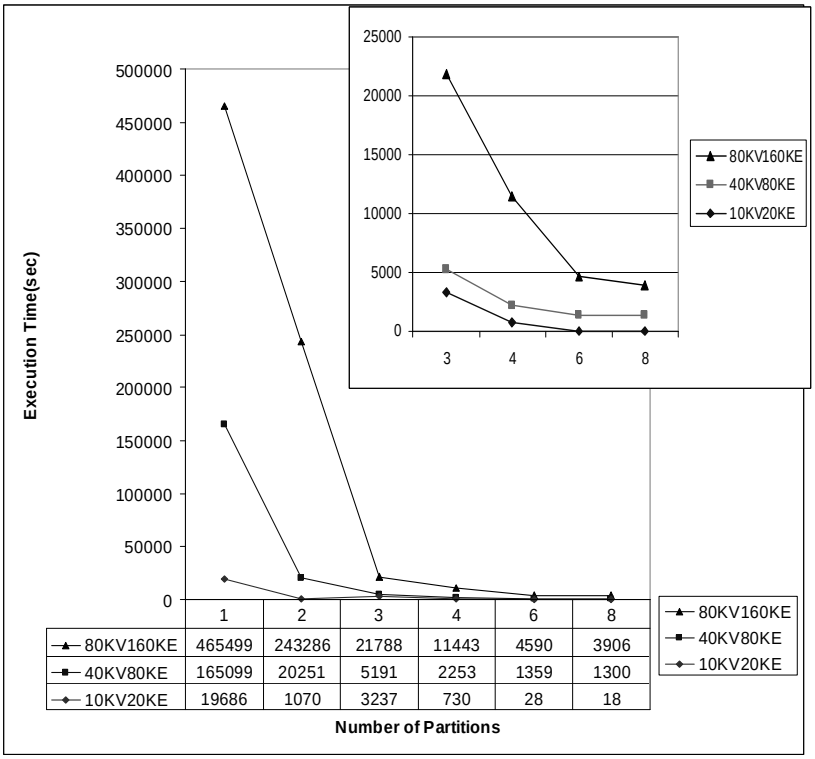
Figure 21 Best substructure broken across partition boundaries, its subgraph occurring as best local substructure

3.3.2 Experimental evaluation

Experiments that demonstrate scalability of a serial partitioned version of Subdue were performed using artificially-generated datasets and protein databases.

3.3.2.1 Artificial datasets

The artificial graphs have been generated using a graph generator that takes as input one or more substructures and the number of instances of each substructure to be embedded in the target graph. Random vertices are then added and random edges are inserted between the vertices to generate a graph of the user-specified size. A post-processing step of randomizing the distribution of the vertices in the graph is performed for our experiments to ensure that all of the embedded substructures are not localized in one part of the graph. The size of the graphs are indicated by the notation xxKVyyKE, where KV stands for kilo vertices (1000 vertices) and KE stand for kilo edges (1000 edges). Also, in some cases a suffix of the type Nw has been added to the above notation where N denotes the number of partitions.



	1	2	3	4	6	8
80KV160KE	465499	243286	21788	11443	4590	3906
40KV80KE	165099	20251	5191	2253	1359	1300
10KV20KE	19686	1070	3237	730	28	18

Figure 22. SSP-Subdue execution time on artificial graphs of varying size and varying number of partitions

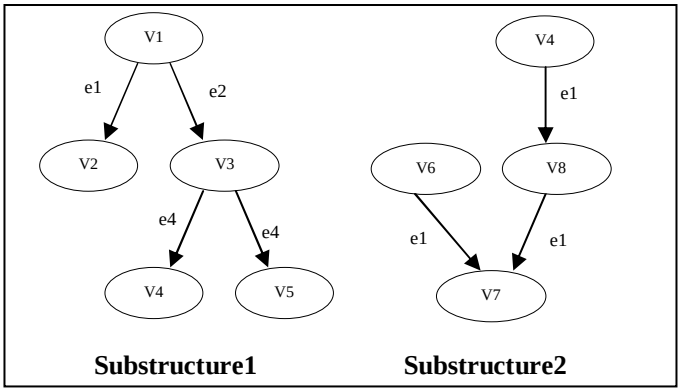


Figure 23. Substructures embedded in artificial datasets

Figure 22 plots the run time of SSP-Subdue on the artificial graphs of varying sizes as the number of partitions increases. For this experiment, the substructures shown in Figure 23 are embedded into the artificial graphs, each with about 35% coverage. The execution

times plotted for Number Of Partitions = 1 represents the execution time for the baseline Subdue algorithm. It is clear that SSP-Subdue achieves a substantial speedup. This is because the run time of Subdue is nonlinear in the size of the input graph. In SSP-Subdue, the Subdue algorithm is applied serially to small portions of the graph, so the combined run time is less than that of Subdue.

SSP-Subdue spent about 15% of its total run-time on the global evaluation of locally-discovered substructures in order to select the globally-optimal substructures.

3.3.2.2 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. Each atom is represented as a vertex whose label is the element name of that atom. For any two atoms whose Euclidean distance is between 0.4 and 1.9 angstroms (or 0.4 and 1.2 angstroms if one or both of the atoms are hydrogen), the program outputs a "bond" edge between the vertices of the two atoms. These bond distances are based on the technique used in the RasMol molecular visualizer. The graphs are not heavily connected and all the edges have the same label (i.e., “bond”). 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.

Figure 24 plots the run time of SSP-Subdue on PDB graphs of varying sizes as the

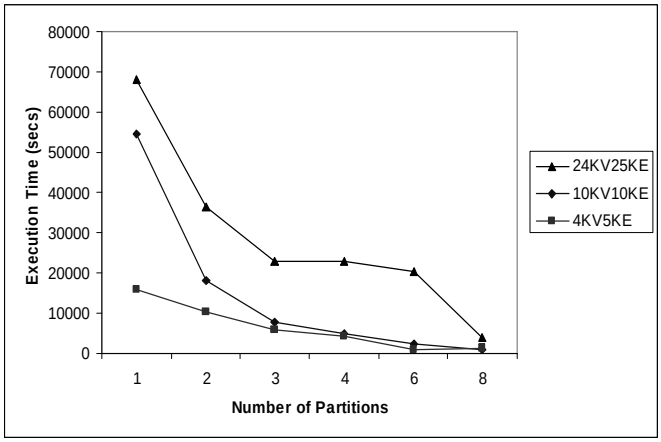


Figure 24 SSP-Subdue execution time on graphs of varying size and varying number of partitions from protein database

number of partitions increases. The execution times plotted for Number Of Partitions = 1 represents the execution time for Subdue. As in the case of artificial graphs, it is observed that a substantial speedup is achieved over Subdue. Here, we have observed an

anomalous small increase in the run time for four partitions as compared to that of three partitions for the 24KV25KE graph. This can be attributed to the fact that the random partitioning can lead to partitions that have varying degrees of connectivity (very highly connected or very sparsely connected) between vertices when partitioned into different

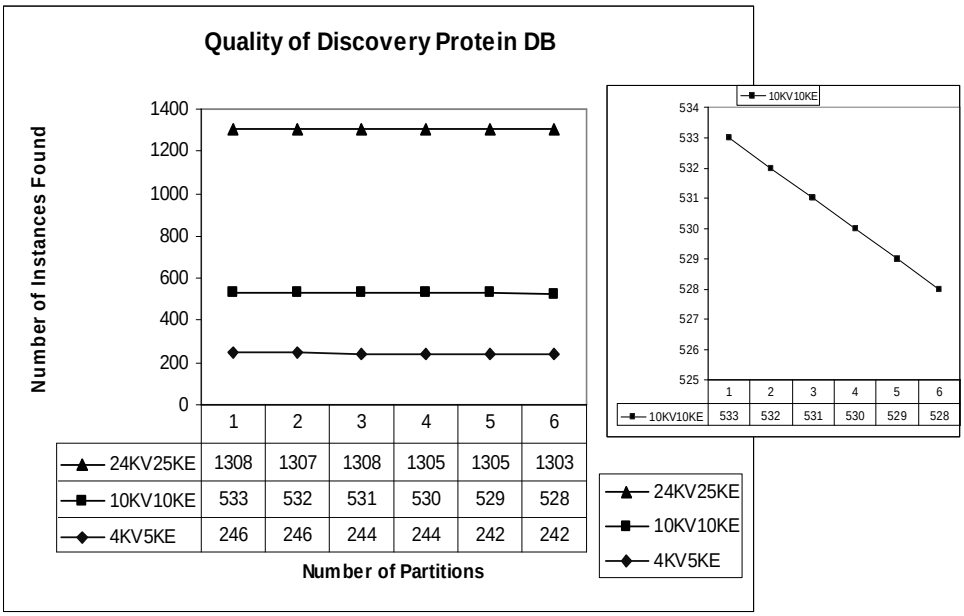


Figure 25 Number of instances of best substructure found in graphs of varying size and varying number of partitions from protein database

number of partitions. The more highly connected a graph/partition is, the more time Subdue takes to process it.

Figure 25 indicates the number of instances of the best substructure found by SSP-Subdue for graphs of varying sizes with a varying number of partitions. The best substructure discovered by Subdue and SSP-Subdue was the same for each of these graphs. As the graphs in the protein database are sparsely connected, the information loss at the partition boundaries is relatively less and thus the quality of discovery is not degraded drastically. An enlarged plot depicting the trend in the quality of discovery for the 10KV10KE graph with varying number of partitions has been inserted in Figure 25 to emphasize the point that the quality is affected, though not drastically.

4. Conclusion and Future Work

4.1 Sequentially-connected data

The focus of incremental discovery in this paper has been on developing the fundamental algorithms for iterative discovery refinement and was tested on data that was not

connected across temporal increment boundaries. However, many domains, including the counter-terrorism domain we discussed, will include event correlations that transcend multiple data iterations. For example, a terrorist suspect introduced in one data increment may be correlated to events that are introduced on latter increments. As each new data increment is received, it may contain new edges that extend from vertices in the new data increment to vertices received in previous increments.

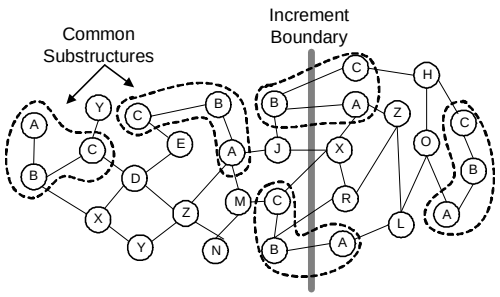


Figure 26. Sequentially connected data

Figure 26 illustrates an example where two data increments are introduced over successive time steps. Common substructures have been identified and two instances of the common substructure extend across the increment boundary.

4.2 Shifting concepts

In the traditional machine learning problem,[15,16] it is generally stated that some function F(x) is generating an attribute vector x, based on a fixed relationship, whether probabilistic or deterministic. The attribute vector x represents the observable features of the problem space. This definition extends intuitively to data mining. However, in sequential discovery problems, the domains are such that the underlying relationships between system variables often change over time. Referring back to our counter-terrorism application, it is certainly the case that terrorist organizations change their behaviors in unpredictable ways and adapt to counter-terrorism efforts. There are approaches to machine learning in the presence of shifting concepts, such as the sliding window approach,[17] where only the last *n* data points are used to update the learned model, but such approaches are often naïve in the sense that they disregard valuable information learned outside of the data window. This is akin to forgetting everything discovered about a terrorist organization's behaviors and capabilities when in fact only a small portion of their behaviors have changed, like an alteration in communication patterns. Our future work will focus on developing methods for structure discovery when the underlying system is undergoing change.

4.3 Graph partitioning

We have developed a naïve graph partitioner for use in our current research. The main motivation behind developing our own partitioner was that the entire input graph need

not be placed in dynamic memory at once. Graphs that are too large to fit in main memory can be streamed and partitioned using this naïve partitioner. It also records the edge cuts in a separate file, which is helpful for edge loss recovery as described in section 3.3.1. We have been able to show that the quality of substructures discovered by SSP-Subdue is good, even with an unsophisticated graph partitioner. The use of a more sophisticated graph partitioner should lead to improved performance of the SSP-Subdue system.

The quality of discovered substructures as well as the speedup achieved by SSP-Subdue depends highly on the graph partitioning step. Subdue does not have any information about the structure of its input graphs and therefore graph partitioning algorithms, like the multi-level Kerninghan-Lin algorithm[18] or the Fiduccia-Matheyses algorithm,[19] are best suited for use in Subdue. However, we are investigating additional partitioning heuristics, beyond frequency of occurrence, which may help reduce the loss of edges that represent important information and are a part of important pattern discoveries.

We are also pursuing analytical methods that could be used to determine if it is possible to bound the probability that any of the edges in a subgraph of a randomly-partitioned graph straddle a partition boundary. It may be possible to bound the number of random partitions that should be considered to ensure that with a high probability most matching subgraphs are found. This would suggest that the algorithm be modified to try many different random partitions.

5. 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.

6. References

1. L. Holder, D. Cook, J. Gonzalez, and I. Jonyer 2002. *Structural Pattern Recognition in Graphs*. In Pattern Recognition and String Matching, Chen, D. and Cheng, X. eds. Kluwer Academic Publishers, 2002.
2. J. Rissanen, *Stochastic Complexity in Statistical Inquiry*. World Scientific Publishing Company, 1989.
3. D. Cook, and L. Holder,. *Substructure Discovery Using Minimum Description Length and Background Knowledge*. In Journal of Artificial Intelligence Research, Volume 1, pages 231-255, 1994.
4. A. Blum. *On-line Algorithms in Machine Learning.* In Proceedings of the Workshop on On-Line Algorithms, Dagstuhl, 1996.
5. N. Friedman and M. Goldszmidt. *Sequential Update of Bayesian Network Structure.* In Proceedings 13th Conf. on Uncertainty in Artificial Intelligence, 1997.

6. R. Agrawal, R. Srikant: *Mining Sequential Patterns*, Proceedings of the Int'l Conference on Data Engineering, Taipei, Taiwan, 1995.
7. G. Hulten, L. Spencer, and P. Domingos. *Mining Time-Changing Data Streams.* KDD-01, San Francisco, CA, 2001.
8. H. Wang, W. Fan, P. Yu and J. Han. *Mining Concept-Drifting Data Streams Using Ensemble Classifiers*, in the 9th ACM International Conference on Knowledge Discovery and Data Mining, 2003.
9. R. Agrawal, and G. Psaila. *Active Data Mining.* In Proceedings of the 1st Int'l Conference on Knowledge Discovery in Databases and Data Mining, 1995.
10. A. Savasere, E. Omiecinsky, and S. Navathe. *An Efficient Algorithm for Mining Association Rules in Large Databases*. 21st Int'l Cong. on Very Large Databases (VLDB), Zurich, Switzerland, 1995.
11. P. Shenoy, J.R. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah. *Turbo-charging Vertical Mining of Large Databases.* ACM SIGMOD Int'l Conference on Managementof Data, 2000.
12. J. Han, J. Pei, and Y. Yin. *Mining Frequent Patterns without Candidate Generation.* ACM SIGMOD Int'lConference on Management of Data, 2000.
13. H. Toivonen. *Sampling Large Databases for Association Rules*. In Proceedings Int. Conf. Very Large Data Bases. Morgan Kaufman, 1996.
14. 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.
15. T. Mitchell. *Machine Learning*. McGraw Hill, 1997.
16. V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, NY, USA, 1995.
17. G. Widmer. and M. Kubat. *Learning in the Presence of Concept Drift and Hidden Contexts.* Machine Learning, 23, 69-101, 1996.
18. B. W. Kerninghan and S. Lin. *An Efficient Heuristic for Partitioning Graphs.* Bell Systems Tech. *J.*, 49:421-308, 1970.
19. C. Fiduccia and R. Matheyses, *A Linear-Time Heuristic for Improving Network Partitions.* In ACM/IEEE Design Automation Conference, pp. 175-181, 1982.