STRUCTURE DISCOVERY IN SEQUENTIALLY-CONNECTED DATA STREAMS

JEFFREY COBLE, DIANE J. COOK AND LAWRENCE B. HOLDER

Department of Computer Science and Engineering
The University of Texas at Arlington
Box 19015, Arlington, TX 76019, USA
{coble,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, such as counter-terrorism intelligence analysis. In this paper we describe an algorithm designed to operate over relational data received from a continuous stream. Our approach includes 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. We then describe a method by which relational dependencies that span across temporal increment boundaries can be efficiently resolved so that additional pattern instances, which do not reside entirely in a single data increment, can be discovered. We also describe a method for change detection using a measure of central tendency designed for graph data. We contrast two formulations of the change detection process and demonstrate the ability to identify salient changes along meaningful dimensions and recognize trends in a relational data stream.

Keywords: Relational Data Mining; Stream Mining, Change Detection

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, our work is focused on data mining techniques to discover relationships between entities. Our work is particularly applicable to problems where the data is event driven, such as the types of intelligence analysis performed by counter-terrorism organizations. Such problems require discovery of relational patterns between the events in the environment so that these patterns can be exploited for the purposes of prediction and action.

Also common to these domains is the continuous nature of the discovery problems. For example, Intelligence Analysts often monitor particular regions of the world or focus on long-term problems like Nuclear Proliferation over the course of many years. To assist in such tasks, we are developing data mining techniques that can operate with data that is received incrementally.

In this paper we present Incremental Subdue (ISubdue), which is the result of our efforts to develop an incremental discovery algorithm capable of evaluating data received incrementally. ISubdue iteratively discovers and refines a set of canonical patterns, considered to be most representative of the accumulated data. We also describe an approach for change detection in relational data streams and contrast two approaches to the problem formulation.

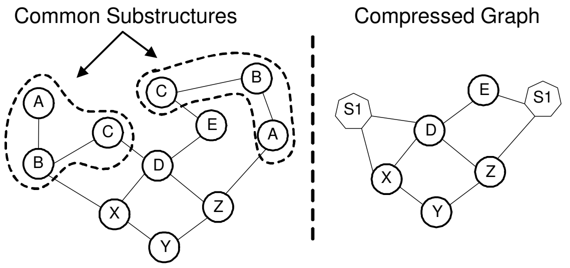


Fig. 1. Subdue discovers common substructures within relational data by evaluating their ability to compress the graph.

2. Structure Discovery

The work we describe in this paper is based upon 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. 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.

3. Incremental Discovery

For our work on ISubdue, we assume that data is received in incremental blocks. Repeatedly reprocessing the accumulated graph after receiving each new increment would be intractable because of the combinatoric nature of substructure evaluation, so instead we wish to develop methods to incrementally refine the substructure discoveries with a minimal amount of reexamination of old data.

3.1. Independent data

In our previous work[3], we developed a method for incrementally determining the best substructures within sequential data where each new increment is a distinct graph structure independent of previous increments. The accumulation of these increments is viewed as one large but disconnected graph.

We often encounter a situation where local applications of Subdue to the individual data increments will yield a set of locally-best substructures that are not the globally best substructures that would be found if the data could be evaluated as one aggregate block. To overcome this problem, we introduced a summarization metric, maintained from each incremental application of Subdue, that allows 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 creates a list of the n best substructures discovered from any dataset, where *n* is configurable by the user.

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

Second, we use the value metric Subdue maintains for each substructure. Subdue measures graph compression with the Minimum Description Length principle as illustrated in Equation 1, 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 our substructure performs, the smaller the compression ratio will be. For the purposes of our research, we have used a simple description length measure for graphs (and substructures) consisting of the number of vertices plus the number of edges. C.f. Cook and Holder 1994 for a full discussion of Subdue's MDL graph encoding algorithm[4].

Subdue's evaluation algorithm ranks the best substructure 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.

For ISubdue, 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.

After running the discovery algorithm over each newly acquired increment, we store the description length metrics for the top n local subs in that increment. By applying our algorithm over all of the stored metrics for each increment, we can then calculate the global top n substructures.

4. Sequentially Connected Data

We now turn our attention to the challenge of incrementally modifying our knowledge of the most representative patterns when dependencies exist across sequentially received data increments. As each new data increment is received, it may contain new edges that extend from vertices in the new data increment to vertices in previous increments.

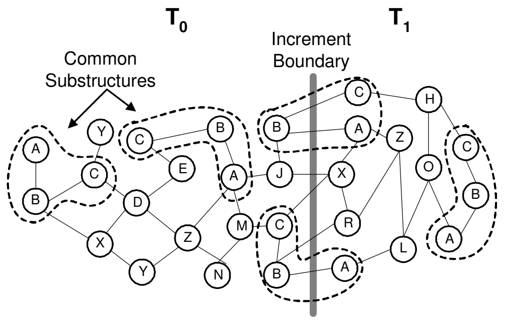


Fig. 2. Sequentially connected data

Figure 2 illustrates an example where two data increments are introduced over successive time steps. Common substructures have been identified and two instances extend across the increment boundary. Referring back to our counterterrorism example, it is easy to see how analysts would continually receive new information regarding previously identified groups, people, targets, or organizations.

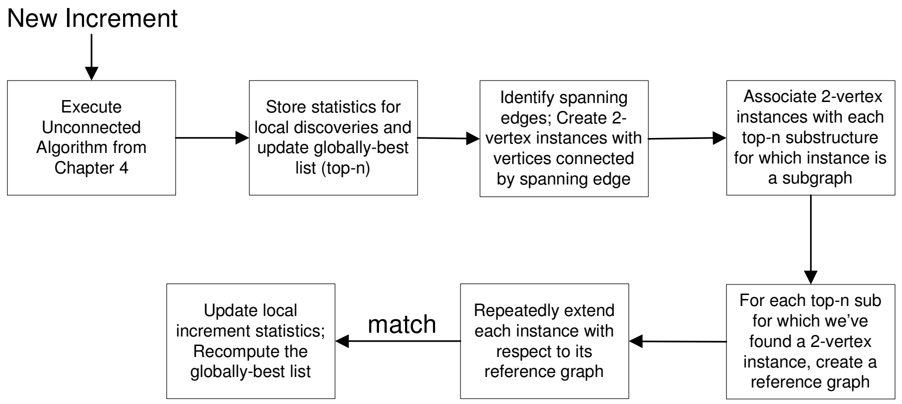

New Increment
Execute Unconnected Algorithm from Chapter 4
Store statistics for local discoveries and update globally-best list (top-n)
Identify spanning edges; Create 2-vertex instances with vertices connected by spanning edge
Associate 2-vertex instances with each top-n substructure for which instance is a subgraph
For each top-n sub for which we've found a 2-vertex instance, create a reference graph
Repeatedly extend each instance with respect to its reference graph
match
Update local increment statistics; Recompute the globally-best list

Fig. 3 Flowchart illustrates the high-level steps of the discovery algorithm for sequentially-connected relational data

4.1. Algorithm

Our only prerequisite for the algorithm is that any pattern spanning the increment boundary that is prominent enough to be of interest is also present in the local increments. As long as we have seen the pattern previously and above some threshold of support, then we can recover all instances that span the increment boundary. Figure 3 illustrates the basic steps of the discovery algorithm at a high level. We discuss the details of the algorithm in the following sections.

4.1.1. Approach

Let

- G_n = set of top-n globally-best substructures
- I_s = set of pattern instances associated with a substructure $s \in G_n$
- V_b = set of vertices with an edge spanning the increment boundary and that are potential members of a top-n substructure
- S_b = 2-vertex pairs of seed substructure instances with an edge spanning the increment boundary
- C_i = set of candidate substructure instances that span the increment boundary and that have the potential of growing into an instance of a top-n substructure.

The first step in the discovery process is to apply the algorithm we developed for the independent increments discussed above. This involves running Subdue discovery on the data contained exclusively within the new increment, ignoring the edges that extend to previous increments. We then update the statistics stored with the increment and compute the set of globally best substructures G_n. This process is illustrated in Figure 4.

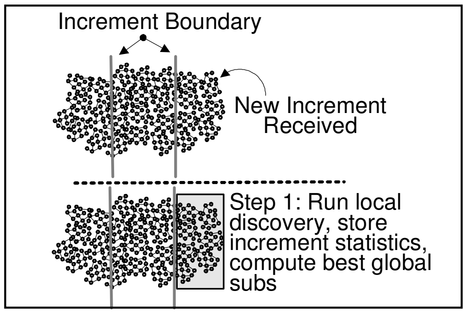
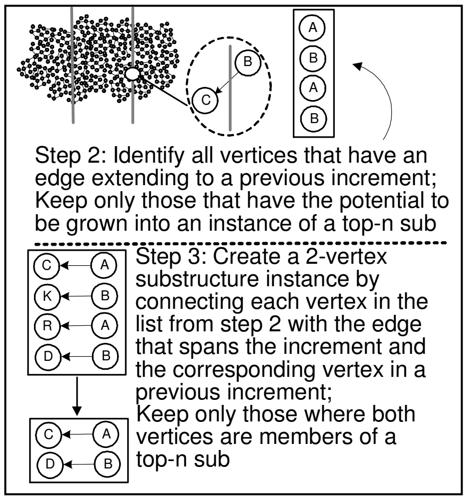

Increment Boundary
New Increment Received
Step 1: Run local discovery, store increment statistics, compute best global subs

Fig. 4. The first step of the sequential discovery process is to evaluate the local data in the new increment

We perform this step to take advantage of all available data in forming our knowledge about the set of patterns that are most representative of the system generating the data. Although the set of top-n substructures computed at this point in the algorithm does not consider substructure instances spanning the increment boundary and therefore will not be accurate in terms of the respective strength of the best substructures, it will be more accurate than if we were to ignore the new data entirely prior to addressing the increment boundary.


A
B
A
B
B
C
Step 2: Identify all vertices that have an edge extending to a previous increment; Keep only those that have the potential to be grown into an instance of a top-n sub
C ← A
K ← B
R ← A
D ← B
Step 3: Create a 2-vertex substructure instance by connecting each vertex in the list from step 2 with the edge that spans the increment and the corresponding vertex in a previous increment; Keep only those where both vertices are members of a top-n sub
C ← A
D ← B
Fig. 5. The second step is to identify all boundary vertices that could possibly be part of an instance of a top n pattern. The third step is to create 2-vertex substructure instances by joining the vertices that span the increment boundary.

The second step of our algorithm is to identify the set of boundary vertices, V_b, where each vertex has a *spanning edge* that extends to a previous increment and is potentially a member of one of the top-n best substructures in G_n. We can identify all boundary vertices, V_b, in O(m), where m is the number of edges in the new increment. Where $p = |V_b| << m$, then for each boundary vertex in V_b we can identify those that are potential members of a top-n substructure in O(k), where k is the number of vertices in the set of substructures G_n, for a total complexity of O(pk). Figure 5 illustrates this process.

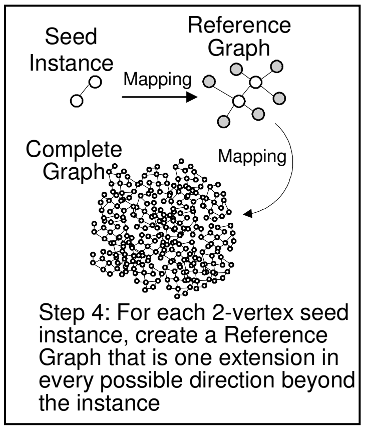

Seed Instance
Mapping
Reference Graph
Mapping
Complete Graph
Step 4: For each 2-vertex seed instance, create a Reference Graph that is one extension in every possible direction beyond the instance

Fig. 6. To facilitate efficient instance extension, we create a reference graph, which we keep extended one step ahead of the instances it represents.

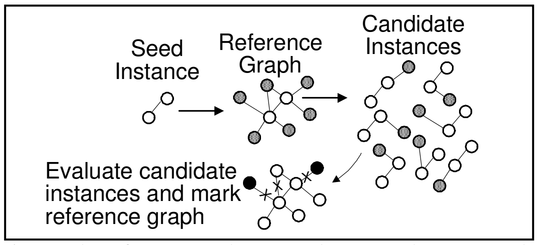
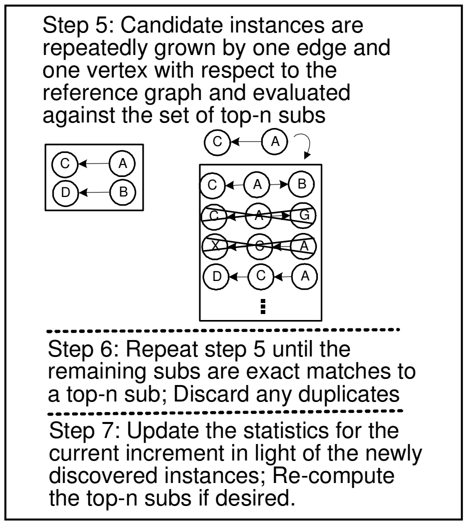


Fig. 7. The reference graphs are used as a template to extend new candidate instances for evaluation against the top-n substructures. Failed extensions are propagated back into the reference graph with marked edges and vertices, to guide future extensions.

For the third step we create a set of 2-vertex substructure seed instances by connecting each vertex in V_b with the spanning edge to its corresponding vertex in a previous increment. We immediately discard any instance where the second vertex is not a member of a top-n substructure (all elements of V_b are already members of a top-n substructure), which again can be done in O(pk). A copy of each seed instance is associated with each top-n substructure, $s_i \in G_n$, for which it is a subset.

To facilitate an efficient process for growing the seed instances into potential instances of a top-n substructure, we now create a set of reference graphs. We create one reference graph for each copy of a seed instance, which is in turn associated with one top-n substructure. Figure 6 illustrates this process.


Step 5: Candidate instances are repeatedly grown by one edge and one vertex with respect to the reference graph and evaluated against the set of top-n subs

Step 6: Repeat step 5 until the remaining subs are exact matches to a top-n sub; Discard any duplicates

Step 7: Update the statistics for the current increment in light of the newly discovered instances; Re-compute the top-n subs if desired.

Fig. 8. The fifth and sixth steps repeatedly extend the set of seed instances until they are either grown into a substructure from S_t or discarded.

We create the initial reference graph by extending the seed instance by one edge and vertex in all possible directions. We can then extend the seed instance with respect to the reference graph to create a set of candidate instances C_i, for each top-n substructure $s_i \in G_n$, illustrated in Figure 7. The candidate instances represent an extension by a single edge and a single vertex, with one candidate instance being generated for each possible extension beyond the seed instance. We then evaluate each candidate instance, $c_{ij} \in C_i$ and keep only those where c_{ij} is still a subgraph of s_i. This evaluation requires a subgraph isomorphism test, which is an NP-complete algorithm, but since most patterns discovered by Subdue are relatively small in size, the cost is negligible in practice. For each candidate instance that is found to not be a subgraph of a top-n substructure, we mark the reference graph to indicate the failed edge and possibly a vertex that is a dead end. This prevents redundant exploration in future extensions and significantly prunes the search space.

In the fifth step (Figure 8), we repeatedly extend each instance, $c_{ij} \in C_i$, in all possible directions by one edge and one vertex. When we reach a point where candidate instances remain but all edges and vertices in the reference graph have already been explored, then we again extend the reference graph frontier by one edge and one vertex. After each instance extension we discard any instance in C_i that is no longer a subgraph of a substructure in G_n. Any instance in C_i that is an exact match to a substructure in G_n is added to the instance list for that substructure, I_s, and removed from C_i.

Once we have exhausted the set of instances in C_i so that they have either been added to a substructure's instance list or discarded, we update the increment statistics to reflect the new instances and then we can recalculate the top-n set, G_n, for the sake of accuracy, or wait until the next increment.

4.2. Discovery Evaluation

To validate our approach to discovery from relational streams, we have conducted two sets of experiments, one on synthetic data and another on data simulated for the counterterrorism domain.

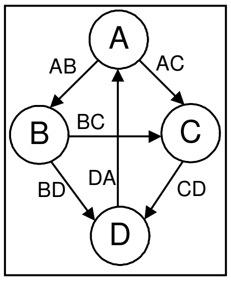


Fig. 9. Predefined substructure embedded in synthetic data.

4.2.1 Synthetic data

Our synthetic data consists of a randomly generated graph segment with vertex labels drawn uniformly from the 26 letters of the alphabet. Vertices have between one and three outgoing edges where the target vertex is selected at random and may reside in a previous data increment, causing the edge to span the increment boundary. In addition to the random segments, we intersperse multiple instances of a predefined substructure. For the experiments described here, the predefined substructure we used is depicted in Figure 9. We embed this substructure internal to the increments and also insert instances that span the increment boundary to test that these instances are detected by our discovery algorithm.

Figure 10 illustrates the results for a progression of five experiments. The x-axis indicates the number of increments that were processed and the respective size in terms of vertices and edges. To illustrate the experiment methodology, consider the 15-increment experiment. We provided ISubdue with the 15 increments in sequential order as fast as the algorithm could process them. The time (38 seconds) depicted is for processing all 15 increments. We then aggregated all 15 increments and processed them with Subdue for the comparison. The five results shown in Figure 10 are not cumulative, meaning that each experiment includes a new set of increments. It is reasonable to suggest then that adding five new increments – from 15 to 20 – would require approximately three additional seconds of processing time for ISubdue, whereas Subdue would require the full 1130 seconds because of the need to reprocess all of the accumulated data. Figure 11 depicts a similar set of experiments

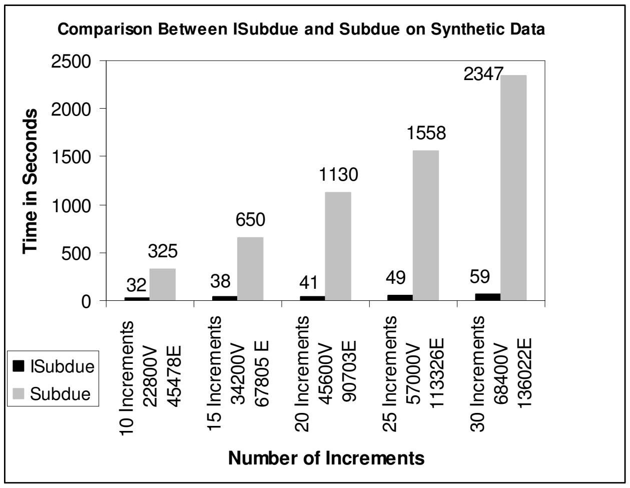


Fig. 10. Comparison of ISubdue and Subue on on increasing number of increments for synthetic data.

In addition to the speedup achieved by virtue of the fact that ISubdue need only process the new increment, additional speedup is achieved because of a sampling effect. This is illustrated in Figure 10 where each independent experiment produces a significant run-time improvement for ISubdue even when processing an identical amount of data as standard Subdue. The sampling effect is an artifact of the way in which patterns are grown from the data. Since ISubdue is operating from smaller samples of data, there are fewer possible pattern instances to evaluate. There are limiting conditions to the speedup achievable with the sampling effect but a full discussion is beyond the scope of this paper.

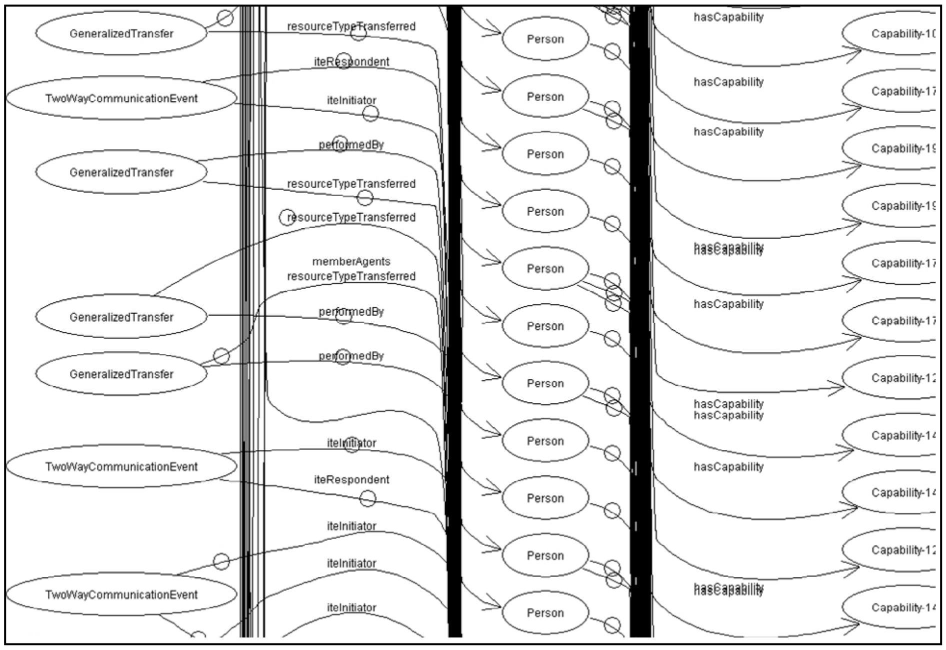


Fig. 11. A section of the graph representation of the counterterrorism data used for our evaluation.

4.2.2. Counterterrorism Data

The counterterrorism data was generated by a simulator created as part of the Evidence Assessment, Grouping, Linking, and Evaluation (EAGLE) program, sponsored by the U.S. Air Force Research Laboratory. The simulator was created by a program participant after extensive interviews with Intelligence Analysts and several studies with respect to appropriate ratios of noise and clutter. The data we use for discovery 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

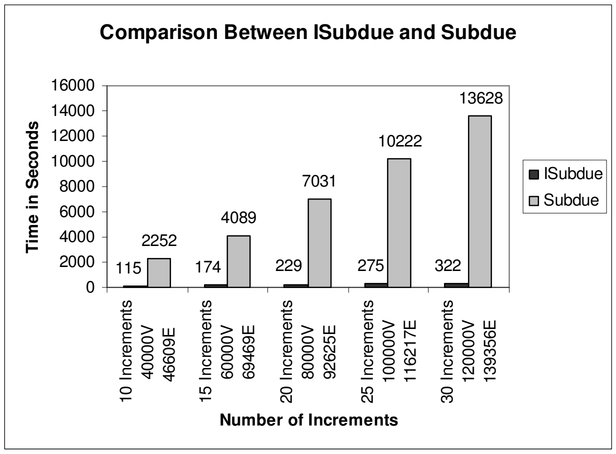


Fig. 12. Comparison of run-times for ISubdue and Subdue on increasing numbers of increments for counterterrorism data.

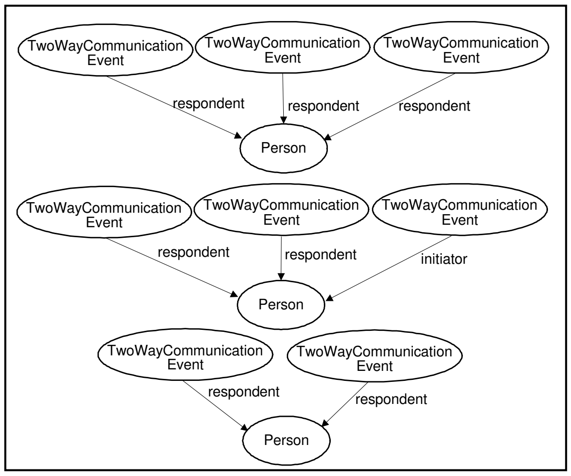

TwoWayCommunication Event
TwoWayCommunication Event
TwoWayCommunication Event
respondent
respondent
respondent
Person
TwoWayCommunication Event
TwoWayCommunication Event
TwoWayCommunication Event
respondent
respondent
initiator
Person
TwoWayCommunication Event
TwoWayCommunication Event
respondent
respondent
Person

Fig. 13. The top 3 substructures discovered by both ISubdue and Subdue for the counterterrorism data.

The data also involves targets and groups, groups being comprised of member agents who are the participants in the aforementioned events. All data is generalized so that no specific names are used. Figure 11 illustrates a small cross-section of the data used in our experiments.

The intent of this experiment was to evaluate the performance of our research on ISubdue against the performance of the original Subdue algorithm. We are interested in measuring performance along two dimensions, run-time and the best reported substructures.

Figure 12 illustrates the comparative run-time performance of ISubdue and Subdue on the same data. As for the synthetic data, ISubdue processes all increments successively whereas Subdue batch processes an aggregation of the increments for the comparative result. Each experiment was independent as it was for the synthetic data.

Figure 13 depicts the top three substructures consistently discovered by both ISubdue and Subdue for all five experiments introduced in Figure 12.

4.3 Qualitative Analysis

In this paper we have described an algorithm to facilitate complete discovery in connected graph data to ensure that we can accurately evaluate the prevalence of specific patterns, even when those patterns are connected across temporal increment boundaries. The basis for using Subdue is that the prevalence of patterns is important, with the prevalence being derived from the number of instances present in the data. If we did not fully evaluate the increment boundaries, we would lose pattern instances and therefore patterns could not be accurately evaluated for their prevalence.

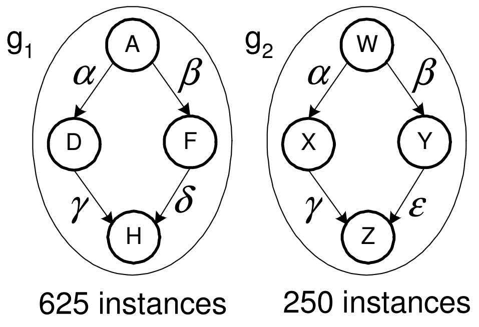


Fig. 14 Two patterns embedded in the random graphs in each data increment and spanning the increment boundary

To illustrate this process, we conducted an experiment using synthetic data similar to that described in section 4.2.1, where the increments were processed both with and without boundary evaluation. The synthetic data consisted of 50 increments, each with 560 vertices and approximately 1175 edges. Interspersed within the random graph data are instances of two different patterns, illustrated in Figure 14, shown with the total number of instances each. The difference is that most of the instances for the first pattern span the increment boundaries, where all of the instances of the second pattern fall completely within single increments.

We ran three separate tests on this data. The first is a benchmark test with the original Subdue algorithm run over the aggregate data, which totals 53,500 vertices and 108,666 edges. As expected, the most prevalent pattern (g_1) is reported by Subdue as the most prevalent with 625 instances discovered.

The second test was run with ISubdue with boundary evaluation enabled. Again, the most prevalent pattern (g_1) was found with 624 instances discovered (occasionally the order of instance growth will result in the loss of an instance and happens for both Subdue and ISubdue).

The last test was run with ISubdue with boundary evaluation disabled. In this case, the second pattern (g_2) is returned as the most prevalent with 250 instances discovered. The first pattern (g_1) was reported as the fourth best with 125 instances discovered. The second-best pattern was a subset of pattern g_1 and the third-best pattern was a small subset of pattern g_2.

Clearly this experiment illustrates the importance of including our boundary evaluation algorithm into the full ISubdue capability. Without the ability to recover pattern instances that do not reside entirely within a single increment we sacrifice the ability to provide accurate discovery results.

5. Detecting Change

Researchers from several fields, such as Machine Learning and Econometrics, have developed methods to address the challenges of a continuous system, like the sliding window approach[5], where only the last *n* data points are used to learn a model. Many of these approaches have been used with success in attribute-based data mining. Other attribute-based methods, such as those involving Ensemble Classifiers[6] and continuously updated Decision Trees[7], have also been successfully demonstrated. However, these methods do not easily translate into relational discovery because of the complex, interconnected nature of the data. Since the data is relationally structured, it can be difficult to quantify the ways in which it may change over time. For example, an attribute vector can be changed by altering the probability distribution of the discrete set of attributes. Conversely, a relational data set may contain a collection of entities and relationship types that can be configured in a large number of different permutations. For a diverse relational dataset, the number of unique patterns may be intractably large. This makes it difficult to quantify the nature of change and so it is not straightforward to apply methods that rely on sampling, such as the sliding window approach.

The remainder of this paper is devoted to describing a process by which we are able to compute a representative point in graph space for sequential sets of patterns discovered by ISubdue from successive data increments received from a continuous stream. We can use these representative points in the graph space, along with a distance calculation, to iteratively measure the evolving set of discoveries to detect and assess change. The objective of this work is to enable a method for measuring pattern drift in relational data streams, where the salient patterns may change in prevalence or structure over time. With a measure of central tendency for graph data, along with a method for calculating graph distance, we can begin to adapt time-series techniques to relational data streams.

As part of our evaluation we have experimented with two different formulations for computing the median graphs – by aggregating sequential sets of local discoveries and by considering the evolving sets of globally-ranked discoveries. Each is discussed below.

5.1. Error-correcting graph isomorphism

The ability to compute the distance between two graphs is essential to our approach for detecting and measure change in incremental discoveries. To compute this distance we rely on the error-correcting graph isomorphism (ecgi)[8]. An ecgi is a bijective function, *f*, that maps a graph g_1 to a graph g_2, such that:

$$f : V_1' \rightarrow V_2'; \quad V_1' \subseteq V_1; \quad V_2' \subseteq V_2$$

where V_1 and V_2 are the vertices from graphs g_1 and g_2, respectively. The ecgi function, $f(v) = u$, provides vertex mappings such that $v \in V_1'$ and $u \in V_2'$, where V_1' and V_2' are the sets of vertices for which a mapping from g_1 to g_2 can be found. The vertices $V_1 - V_1'$ in g_1 are deleted and the vertices $V_2 - V_2'$ in g_2 are inserted. A cost is assigned for each deletion and insertion operation. Depending on the formulation, a direct substitution may not incur cost, which is intuitive if we are looking to minimize cost based on the difference between the graphs. It should be noted that the substitution of a vertex from g_1 to g_2 may not be an identical substitution. For instance, if the vertices have different labels, then the substitution would be indirect and a cost would be incurred for the label transformation.

The edit operations for edges are similar to those for vertices. We again have the situation where the edge mappings may not be identical substitutions. The edges may differ in their labeling as well as their direction.

Figure 15 illustrates the mapping of vertices from graph g_1 to g_2. Vertex substitutions are illustrated with dashed lines in Figure 15.b, along with the label transformation. Figure 15.c depicts the edge substitution, deletion and insertion operations.

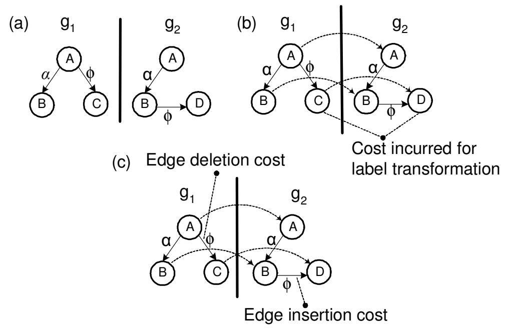


Fig. 15. Example of error-correcting graph isomorphism function mapping g_1 to g_2.

The ecgi returns the optimal graph isomorphism, where optimality is determined by the least-cost set of edit operations that transform g_1 to g_2. The costs of edit operations can be determined to suit the needs of a particular application or unit costs can be assigned. The optimal ecgi solution is known to be NP-complete and is intractable for even relatively small graphs. In the following sections we discuss an approximation method for the ecgi calculation.

5.2. Graph metrics

The objective of this work is to develop methods for applying metrics to relational data so that we can quantify change over time. At the heart of almost all statistical analysis techniques is a measure of central tendency for the data sample being evaluated. The most prevalent of such measures is the mean. By definition, a mean is the point where the sum of all distances from the mean to the sample data points equals zero, such that:

$$\sum_{i=1}^{n} \Delta x_i = 0$$

where Δx_i is the distance from the mean to data point x_i. This definition can be rephrased to say that an equal amount of variance lies on each side of the number line from the mean. Unfortunately there is no straightforward translation of this definition into the space of graphs, since there is no concept of positive and negative that can be applied systematically. A mean graph was defined in Bunke and Guenter 2001[9], but only for a single pair of input graphs. This is possible because a mean graph can be found that is equidistant from the two input graphs, which causes it to satisfy the form of a statistical mean. There does not appear to be a way to define a mean graph for multiple input graphs that would satisfy the form or variance requirements of a statistical mean.

Fortunately, it is possible to compute a median graph[8], which is another common measure of central tendency. There is an analogous definition between the median graph and the statistical median. A statistical median is the point on the number line where exactly half of the data points lie below and half lie above. This also happens to be the point that gives us the minimum sum of distances, which is the sum of the distance from the median to each data point. No other point can be chosen with a smaller sum of distances. Although there is no concept of above and below in graph space, we can rely on the sum of distances to provide a mechanism for computing a median graph. Equation 4 formalizes this notion.

$$\hat{g} = arg\ \min_{g \in U} \sum_{i=1}^{n} d(\, g, g_i \,) \tag{4}$$

The median graph, $\hat{g}$, is a graph that minimizes the sum of distances to all n input graphs and U is the universe of graphs that can be generated from the available labels.

The ecgi defined above provides a method to measure a potential median for its proximity to the input graphs. To deal with the intractability of computing the ecgi between a candidate median graph and multiple input graphs, a stochastic approximation technique was introduced in Jiang et. al 2001[8], which relies on a genetic algorithm[10] to reduce the search space for optimal graph transformations. The genetic algorithm uses a chromosome to represent a vertex mapping from a potential median graph to the input graphs. The fitness function completes the mapping by finding an optimal vertex label mapping, edge mapping, and edge label mapping. The process of computing the fitness of a chromosome is the process of inducing an optimal median graph based on the vertex mapping in the chromosome and the input graphs. The genetic algorithm approach reduces the problem complexity to quadratic time and although it is an approximation technique, in practice it produces near optimal results. In the following section we introduce our modifications to the fitness function needed to address the issue of weighting discovered patterns so that they appropriately influence the median graph induction.

5.2.1. Weighting patterns

Our purpose for computing the median graph is to summarize a set of discovered patterns. We perform this summarization over the sets of patterns discovered by ISubdue as data is incrementally processed. In order to compute a median graph that is representative of the reported patterns, we must weight the computation process by the relative strength of each pattern. Consider the set of patterns, *G*, and their respective metrics, reported for a series of increments.

$$G = \{(g_1, n_1, i_1), (g_2, n_2, i_2), (g_3, n_3, i_3), \ldots, (g_m, n_m, i_m)\}$$

Where g_j is the pattern, n_j is the number of instances, and i_j is the increment data (size = $|V_j| + |E_j|$).

For the purposes of illustration, assume the graphs $\{g_1, g_2, g_3\}$ were discovered in three distinct data increments, $\{i_1, i_2, i_3\}$. The metrics needed for weighting patterns are then:

$$|g_1| = 10;\ n_1 = 30;\ |i_1| = 5200$$
$$|g_2| = 10;\ n_2 = 20;\ |i_2| = 6800$$
$$|g_3| = 10;\ n_3 = 25;\ |i_3| = 6000$$

The aggregate size for a pattern is simply the size of the pattern multiplied by the number of instances, $|g_j| * |i_j|$. We normalize the aggregate size value relative to the size of the increments in which the pattern was discovered and the overall size of all increments for which we are computing the median graph. The normalized aggregate size is computed using Equation 5.

$$size(g_j) = \frac{|i_j|}{\sum_{k=1}^{m} |i_k|} * \left(|g_j| * |n_j|\right) \tag{5}$$

We can then weight each pattern using Equation 6.

$$w(g_j) = \frac{size(g_j)}{\sum_{i=1}^{m} size(g_i)} \tag{6}$$

For the patterns described above, the normalized aggregate sizes would be calculated using Equation 5 as follows:

$$size(g_1) = \frac{5200}{5200+6800+6000} * (10*30) \approx 87$$

$$size(g_2) = \frac{6800}{5200+6800+6000} * (10*20) \approx 76$$

$$size(g_3) = \frac{6000}{5200+6800+6000} * (10*25) \approx 83$$

The respective weights would then be calculated using Equation 6 as follows:

$$w(g_1) = \frac{87}{87+76+83} \approx 0.35$$
$$w(g_2) = \frac{76}{87+76+83} \approx 0.31$$
$$w(g_3) = \frac{83}{87+76+83} \approx 0.34$$

Naturally these weights sum to one.

Now that we have the weights, our objective is to scale the cost values used in the genetic algorithm's fitness function so that the relative strength of each pattern has its respective influence on the median graph. The genetic algorithm uses its fitness function to determine the cost of mapping a potential median graph to the input graphs. Figure 16 illustrates a median graph that was induced from weighted input graphs. For each potential edge configuration of the median graph, the cost is computed as follows:

$$cost = 0.6 * c_{es}^{a} + 0.2 * c_{es}^{b} + 0.2 * c_{es}^{c}$$

where c_{es} is the edge substitution cost from the median graph edge to the respective edge in the input graph. Each possible edge configuration for the median graph is evaluated with this weighted cost function and the one with the least cost is selected, which is shown in the figure. In this case, since input graph (a) is heavily weighted, it is less costly to transform the corresponding edge for both of the input graphs (b) and (c) than to transform an alternate configuration for (a).

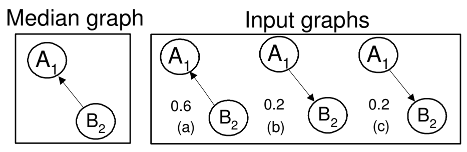


Fig. 16. Mapping costs are weighted based on pattern prevalence

5.3. Change-detection evaluation

The purpose of computing a median graph that represents the best patterns for a set of increments is so that we have points for comparison across a large time span without the computational burden of comparing every discovered pattern from every increment. A secondary benefit is that a median graph is less susceptible to anomalies that might appear in local data increments, making it more useful for analyzing broader trends.

5.3.1. Measuring Similarity

We previously defined the ecgi as a measure of graph distance, but a single numerical distance metric is insufficient for analytical purposes. By definition, relational data is constructed of entities and relationships corresponding to vertices and edges in graph configurations. At a minimum we require a similarity metric for both dimensions.

To measure a degree of change along these dimensions, we first set the cost of all edit operations to unit cost and we can then define the maximum possible change between two graphs along each dimension as simply the larger number of vertices and edges, respectively, from the two graphs. We base our similarity measure on a graph distance metric derived from the maximal common subgraph[11] (mcs), shown in Equation 7.

$$d(g_1,g_2) = 1 - \frac{|mcs(g_1,g_2)|}{max(|g_1|,|g_2|)} \tag{7}$$

Any disjoint areas of the two graphs incur an error-correction cost in the ecgi. When the edit costs are set to unit value, the total incurred cost is the compliment of the size of the maximal common substructure. Therefore a measure of dissimilarity between two graphs g_1 and g_2 can be computed on each dimension as follows:

For $g_1 = (V_1, E_1)$ and $g_2 = (V_2, E_2)$, where V is the vertex set and $E \subseteq V \times V$ is the edge set, the vertex distance between the two graphs can be computed using Equation 8.

$$d(V_1,V_2) = \frac{\sum_{j=1}^{k_1} c_{vs}(u_j,v) + \sum_{j=1}^{k_2} c_{vd}(u_j) + \sum_{j=1}^{k_3} c_{vi}(v_j)}{\max(|V_1|,|V_2|)} \tag{8}$$

Where k_1 is the number of vertex substitutions from graph g_1 to g_2, k_2 is the number of vertex deletions from g_1 and k_3 is the number of vertex insertions into g_2. It follows intuitively that $k_1 + k_2 + k_3 = max(|V_1|,|V_2|)$ and if there is no common substructure in the two graphs, then the costs will sum to equal $max(|V_1|,|V_2|)$ and $d(V_1, V_2) = 1$. The cost functions are defined as follows.

- $c_{vs}(u_j, v)$: cost of substituting vertex u_j from graph g_1 to vertex v in graph g_2. The cost includes the label transformation costs.
- $c_{vd}(u_j)$: cost of deleting vertex u_j from graph g_1
- $c_{vi}(v_j)$: cost of inserting vertex v_j into graph g_2

$$d(E_1,E_2) = \frac{\sum_{j=1}^{m_1} c_{es}\left(\left(u_1^j,u_2^j\right),\left(v_1,v_2\right)\right) + \sum_{j=1}^{m_2} c_{ed}\left(u_1^j,u_2^j\right) + \sum_{j=1}^{m_3} c_{ei}\left(v_1^j,v_2^j\right)}{max(|E_1|,|E_2|)} \qquad (9)$$

The edge metric is calculated similarly to the vertex distance, where m_1 is the number of edge substitutions from graph g_1 to g_2, m_2 is the number of edge deletions from g_1 and m_3 is the number of edge insertions into g_2. Then $m_1 + m_2 + m_3 = max(|E_1|,|E_2|)$ and if there is no common substructure in the two graphs, then the costs will sum to equal $max(|E_1|,|E_2|)$ and $d(E_1, E_2) = 1$. The cost functions are defined as follows.

- $c_{es}((u_1^j, u_2^j), (v_1, v_2))$: cost of substituting edge (v_1, v_2) in graph g_2 with edge (u_1^j, u_2^j) from graph g_1. The cost includes the label transformation costs.
- $c_{ed}(u_1^j, u_2^j)$: cost of deleting edge (u_1^j, u_2^j) from graph g_1
- $c_{ei}(v_1, v_2)$: cost of inserting edge (v_1, v_2) in graph g_2

Using these cost calculations, we are able to provide a distance measure along both vertex and edge dimensions as well as the overall graph distance metric.

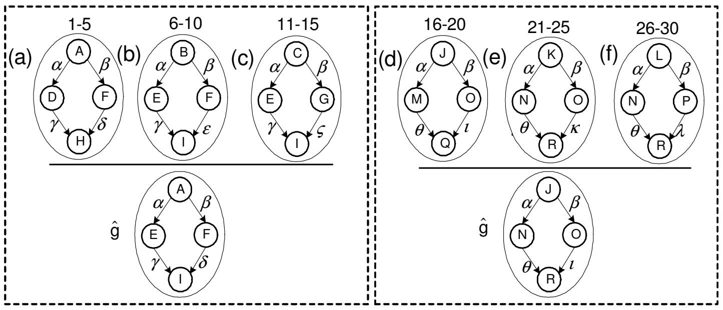


Fig. 17. Synthetic data contains six unique substructures and two alternating cycles.

5.3.2. Synthetic data

To illustrate the use of a median graph and the similarity measure, we have created a synthetic, continuous dataset containing a set of patterns that appear cyclically over time. The data was designed with the counter-terrorism problem in mind, where there are often a larger number of entities (people, places, etc) connected with a relatively few number of unique relationship types. The experiment was conducted over 165 data increments, each containing 550 vertices and approximately 1100 edges. Each increment has embedded within it 10 instances of one of the six patterns from Figure 17.

The figure illustrates the patterns embedded within the first 30 increments along with an example of a median graph computed for a 15-increment set. Some of the increments have five instances of the pattern that span across the increment boundary. For the purposes of clear illustration, we only varied the embedded patterns by their labeling, which represents a shift in entities for vertex label changes and a shift in relationships for edge label changes.

The majority of the graph data within each increment is random, with $(26 + 26^2)$ unique vertex labels drawn from the set $\mathcal{L}_V = \{A, B, C, \ldots, Z\}$, where labels are one or two characters long. The two-character labels are a random concatenation of the set members. Edges are selected randomly from the set $\mathcal{L}_E = \{\alpha, \beta, \gamma, \delta, \varepsilon, \varsigma, \eta, \theta, \iota, \kappa, \lambda\}$.

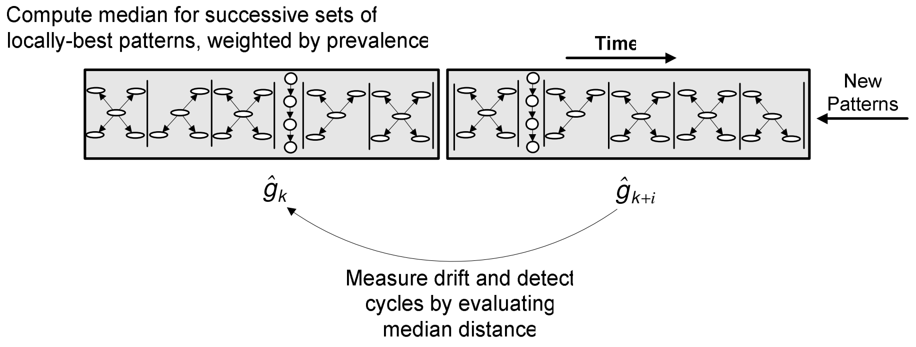

Compute median for successive sets of locally-best patterns, weighted by prevalence
Time
New Patterns
$\hat{g}_k$
$\hat{g}_{k+i}$
Measure drift and detect cycles by evaluating median distance

Fig. 18 As local increments are processed serially, a sequence of prevalent patterns is collected

5.3.2.1. Measuring local discoveries

Figure 18 illustrates one potential approach to the problem formulation, where each new data increment is evaluated using ISubdue, giving us a relational pattern that is locally-best in the new increment. These locally-discovered patterns are used to refine the globally-best pattern using the methods described above. We can assess change by computing median graphs using aggregated sets of these locally-best patterns received over time, such as the two sets depicted in Figure 18. Each pattern is weighted based on its strength relative to the increment in which it is discovered and the set of increments with which it is grouped. We then use the graph metrics described above to compute representative points for sets of local discoveries so that we can measure the change over time.

Using the synthetic data generator described above, we collected sets of the single locally-best pattern discovered in each increment. For this experiment, we aggregated the best local discovery from each of 15 increments for the computation of each median.

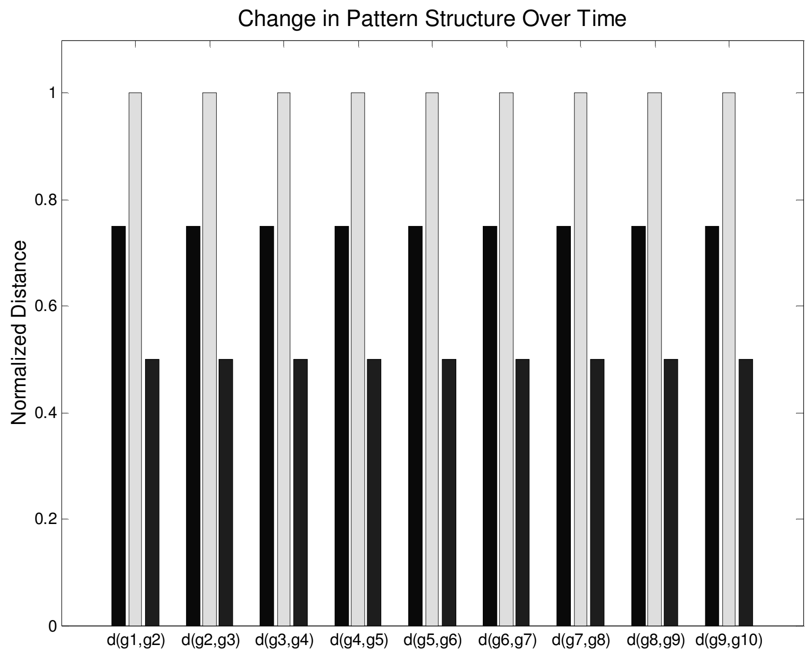


Fig. 19. Each 3-bar set depicts the total graph distance, vertex distance and edge distance, respectively

Figure 19 illustrates the distance between each median graph and its successor, which is a drift measurement of one group of 15 increments to the next group of 15. For this experiment, there is a significant transition in terms of the overall graph distance. Each median differs from the next by a normalized distance of 0.75. The pairwise vertex and edge distances for each comparison are also shown, where the vertex distance is the maximum of 1.0, representing a complete shift but only half of the edges transition. For many domain problems, characterizing the change along different dimensions is important. For example, in the counter-terrorism domain, analysts would want to know if the people involved in a particular threat pattern have changed or if the behavioral pattern involving known terrorists has changed.

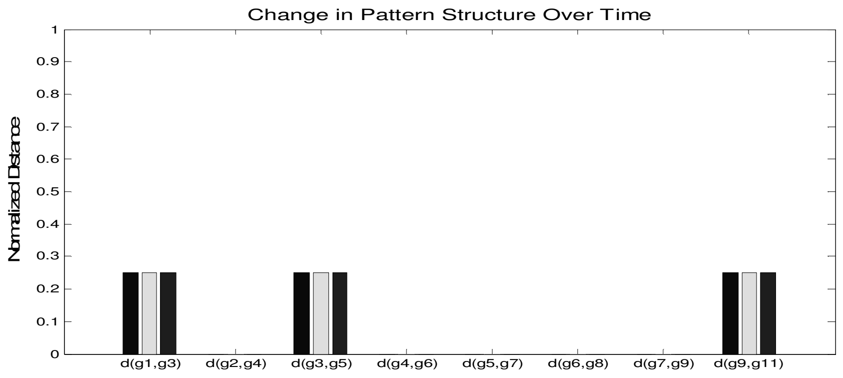


Fig. 20. By phase-shifting the median comparison we can detect cycles in the data

By phase-shifting the comparison between medians, we can identify potential cycles in the data. Figure 20 illustrates another pairwise comparison for the same synthetic data, where each median is compared to its second successor, e.g. median 1 is compared to median 3, etc. We see the distances in most cases drop to 0 and the remaining distances are insignificant, a result of minor variations in the medians caused by pattern weights.

In practice the number of increments represented by each median might be selected through an iterative search process or be modified to find precise drift boundaries once a cycle has been detected. However, the selection of the set size might be overly influential on the efficacy of the median comparison. The median might also be unduly influenced by minor variations in the rankings of local discoveries. We describe an alternative formulation below.

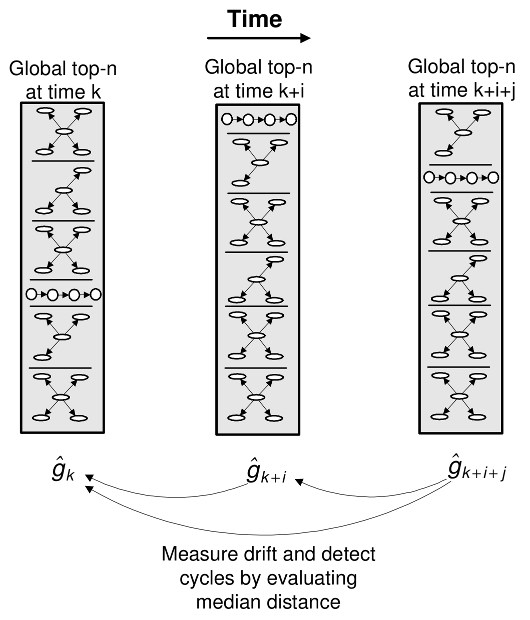


Fig. 21. The median-graph change detection process can be applied to the evolving set of globally-best top-n patterns to measure the drift or detect cycles

5.3.2.2. Measuring globally-ranked patterns

Since the process of incremental discovery that we defined in sections 3 and 4 will continuously update the set of top-n globally-best patterns (the n best patterns discovered so far, where n is specified by the user), we can compute the median over these evolving pattern sets and measure the drift or detect cycles as described above. Each pattern in the ordered set would be weighted with the same process we described in section 5.2.1, based on its prevalence and the size of the increment in which it was discovered. Medians are computed for the new globally-best list of patterns after each new increment is processed, but could also be computed on some larger interval (e.g. every five increments). Figure 21 illustrates this concept.

Using the synthetic data generator described above, we collected the complete set of globally-best patterns that were available after each new increment was processed and then computed a median graph for each. Each set of globally-best patterns contained every pattern that had been discovered up to that point in the ISubdue discovery process (no limiting n-value was specified). Alternatively, we could have truncated the list by keeping only the best *n* patterns. The median graph was then computed using the weighting method described in section 5.2.1 so that each pattern on the globally-best list contributed to the median in proportion to its representation in the aggregate data.

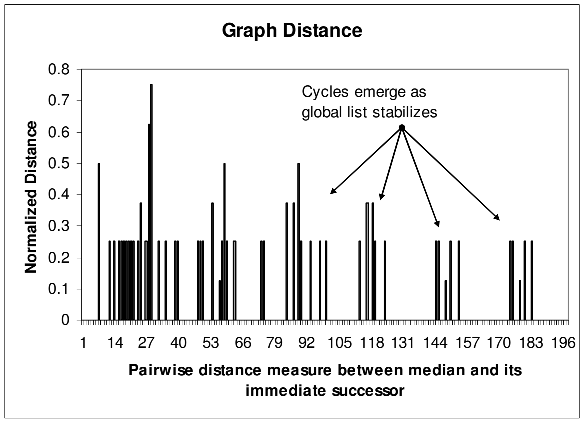


Fig. 22. Distance between median graphs computed for successive sets of globally-best patterns. Distance shown for total graph only.

Figure 22 illustrates the pairwise distance comparison between each median and its immediate successor for the 200 increments processed in this experiment. Initially, the distance comparisons appear somewhat erratic. This is because it takes some time for the globally-best set of patterns to stabilize. Although each increment has the same number of vertices and nearly the same number of edges, the vertex and edge labels are assigned randomly. This means that the way in which patterns are grown by the core Subdue algorithm for the local data in each increment can produce some randomness in the local discoveries. The best pattern discovered in each local increment is always as expected, but the non-best discoveries (2nd, 3rd, etc) are often subsets of the best pattern and these can vary with respect to the way in which the instances are grown, due to variance in the data. Within the first few cycles of the data, these variations will influence the median until a representative sample of these non-best patterns has been accumulated and their influence has been averaged out. As Figure 22 clearly indicates, this noise begins to fall out as time goes on and clear cycles can be observed.

For this synthetic data, both formulations yield information that indicates the presence of a data cycle.

5.3.3. Domain Experiments

We conducted a second set of experiments using the counter-terrorism data generated by the simulator described in section 4.2.2. For this experiment we used data generated by the simulator for threat groups – groups known to engage in terrorist activity. Each group is represented by a graph structure that aggregates a collection of member agents, which are then associated with specific capabilities and resources. Groups may also be associated with specific operational modes, which are collections of capabilities and resources. To illustrate trend detection, we selected the first four threat group cases from five different datasets and repeated this process seven times to generate 350 increments totaling 85,670 vertices and 66,041 edges. Increments were limited to 500 vertices plus their edges, causing group graphs to be severed so that they spanned increment boundaries. Medians were computed for the evolving set of globally-best patterns using the same process described above. Figure 23 illustrates an example of a pattern discovered from the threat group data. This pattern represents a particular operating mode for a group, involving two capabilities and one resource.

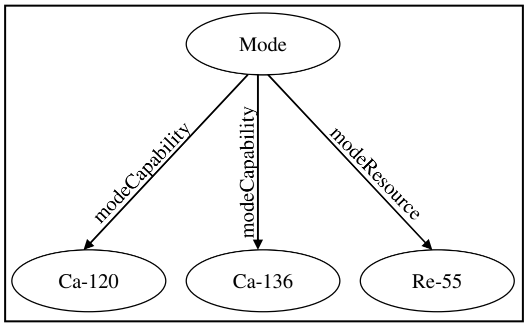


Fig. 23. An example of a discovered pattern from the counterterrorism Threat Group data

5.3.3.1 Measuring local discoveries

We again first computed median graphs over 5-increment aggregates of locally-best discoveries. Figure 24 illustrates the pairwise distance comparisons between the current increment and its immediate successor, computed with the similarity measure from section 6.1. These values reflect a drift over the vertex and edge dimensions for 10 successive increments. Figure 24 illustrates two full cycles through the data, where the first six data points represent the first cycle and the last six represent the second cycle.

The center (seventh) data point is the pivot point, where the last median from the first cycle is measured against the first median of the second cycle. The two cycles are separated and overlaid in the graphic for illustrative purposes. A cycle could be detected in the same manner as illustrated for the synthetic data experiment, where pairwise distances are continually computed with a phase shift until we find a precipitous drop in the measurements. In this case, when the median g_1 was compared to median g_8 the distance would be zero.

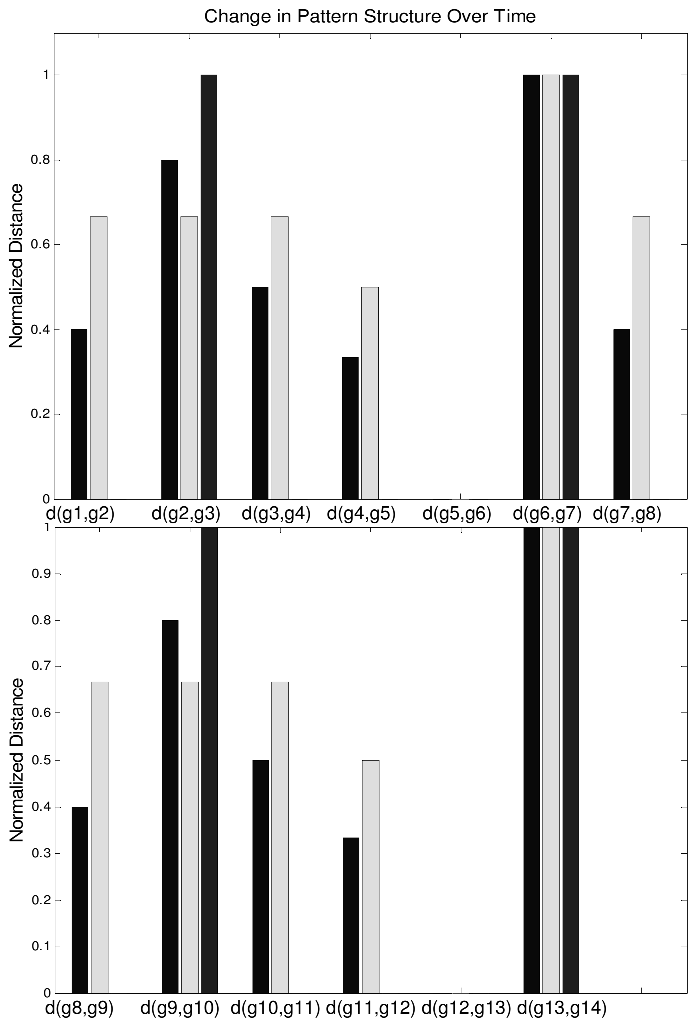


Fig. 24. The two cycles depicted in Figure 17 are separated and overlaid for illustration

5.3.3.2. Measuring globally-ranked patterns

Figure 25 depicts the successive pairwise graph distance comparison between the medians computed for the globally-best pattern sets, where the distance is between the median for the current globally-best pattern set and the median for the pattern set from the prior time step. The median graph is recomputed after each new increment is put through the discovery process and the global pattern list is updated.

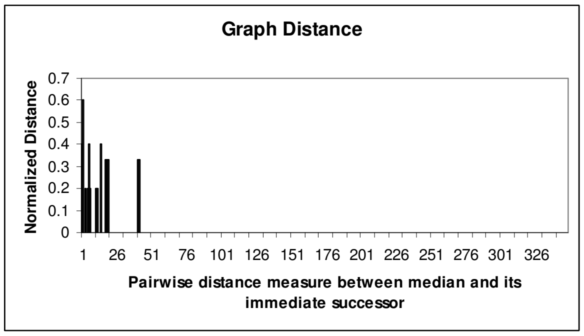


Fig. 25. Distance between median graphs computed for successive sets of globally-best patterns. Distance shown for total graph only.

In this case, the median graph stabilizes quickly and the distance measures drop to zero. This experiment behaved differently from the synthetic data experiment because the patterns that comprise the globally-best set are diverse and not equally represented, which allowed the globally-best list to converge to a specific median graph early in the sequential processing. The oscillation we see in the synthetic data experiment is an artifact of the similarity among the embedded patterns and the identical strength of each. For the synthetic experiment, as each new increment is processed, the median is influenced in an oscillating fashion.

The experiments discussed in this paper are for relational data streams containing pattern cycles, chosen for illustrative clarity. We have also conducted a number of experiments where our approach accurately measures various rates of pattern drift over time.

5.3.4 Contrasting approaches

The work we have presented here regarding change detection in relational data streams is meant to serve as an entry point into the topic. To gain some perspective on the depth of this challenge, consider the field of Time Series Analysis[12], which has evolved over the span of decades and is built upon a foundation of statistical analysis methods for numerical data. Time Serious Analysis provides many techniques for numerical data that each can reveal specific types of information about time-series data. We have presented two possible formulations for computing the median graph from relational data streams – using sequentially-discovered sets of local patterns and using the evolving set of global patterns – each with a range of value. There are many other formulations, such as sets of local pattern discoveries or a sliding window of local or global pattern discoveries.

As we have seen with the two approaches presented here, neither provides all of the interesting information under all conditions. For example, the experiment depicted in Figure 25 does not reveal the presence of a cycle but does reveal that the cycle isn't significant enough to change the overall global pattern ranking. As part of our ongoing work, we seek to develop a mapping of which formulations can reveal which information and under what circumstances. This will allow us to use various formulations in conjunction with one another to reveal as much information as possible, much like time-series techniques for numerical data.

6. Conclusions and Future Work

In this paper we have described an algorithm for relational pattern discovery from data streams when the data is connected across increment boundaries. We have also described a method for change detection in graph-based data and contrasted two formulations for the change-detection process. Each algorithm has been empirically demonstrated to be effective on both synthetic and domain data.

6.1. Increment Size

We have learned from our experimentation that the size of the data increments must be chosen with some care. If data increments are too small, then the local discovery process we use as a precursor to our boundary evaluation may be overly biased to incomplete patterns. In practice, it is often possible to select an appropriately sized increment boundary given some knowledge about the domain. However, there are situations where the data may obey irregular cycles and therefore the increment size shouldn't be set to a fixed size. In our future work we intend to explore statistical and information theoretic measures for dynamically selecting an increment size.

6.2. Formalizing relational change detection

In this paper we have built upon our work in relational data mining from continuous streams to demonstrate how a measure of central tendency for graph data, along with a two-dimensional metric for describing the distance between two graphs, can be used to detect and analyze certain types of change in relational data mined from a continuous stream. We have illustrated the applicability of our method on both synthetic and domain-specific data.

The ability to identify changes in relational data streams opens several significant research questions. First, we wish to develop rigorous analogues to traditional statistical techniques that are applicable to graph-based data and that are mathematically sound when using the median as a measure of central tendency. Another significant question is how to adapt the discovery process once we identify a particular drift characteristic. Since the nature of change within relational data systems is more complex, due to the diverse interconnection of entities, it is not clear whether general methods can be developed for adapting the discovery process to pattern drift. We are exploring whether there are general data characteristics that, if known, can guide the drift adaptation process or if the response must be domain dependent.

Finally, as described in section 5.3.4, there is much additional work to be done to assess the applicability and limitations of various formulations of the median graph computation.

6.3. Divergence Measures

In previous work[13] we briefly explored an information-theoretic divergence measure known as Relative Entropy as an alternative divergence measure. Initial evaluation led us to believe that this approach was not widely applicable to situations where we needed to measure divergences between data represented by disparate patterns, however it does provide a sound measure of divergence for data sets represented by a different frequency of the same pattern. This is but one similarity measure among many.[14] In future work, we intend to explore a range of statistical and information-theoretic measures that may be applicable to evaluating pattern drift in relational data streams.

6.4. Acknowledgements

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.

7. References

1. Holder, L., Cook, D., Gonzalez, J., and Jonyer, I. 2002. *Structural Pattern Recognition in Graphs*. In Pattern Recognition and String Matching, Chen, D. and Cheng, X. eds. Kluwer Academic Publishers.
2. Rissanen, J. 1989. *Stochastic Complexity in Statistical Inquiry*. World Scientific Publishing Company, 1989.
3. Coble, J., Rathi, R., Cook, D., Holder, L. *Iterative Structure Discovery in Graph-Based Data*. In the International Journal of Artificial Intelligence Tools, Volume 14, No. 1 & 2, 2005.
4. 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.
5. Widmer, G.. and Kubat, M. *Learning in the Presence of Concept Drift and Hidden Contexts*. Machine Learning, 23, 69-101, 1996.
6. Wang, H., Fan, W., Yu, P. and Han, J. *Mining Concept-Drifting Data Streams Using Ensemble Classifiers.* In the 9th ACM International Conference on Knowledge Discovery and Data Mining, 2003.
7. Hulten, G., Spencer, L. and Domingos, P. *Mining Time-Changing Data Streams*. Proceedings of the 7th ACM SIGKDD Confereence on Knowledge Discovery and Data Mining, 2001.
8. Jiang, X., Münger, A., Bunke, H. *On Median Graphs: Properties, Algorithms, and Applications*, IEEE Trans. on Pattern Analysis and Machine Intelligence. 23 (10), 2001, 1144 – 1151.
9. Bunke, H. and Guenter, S. *Weighted Mean of a Pair of Graphs*. Computing, Volume 67:3, 2001, pp. 209 - 224.
10. Russell S. and Norvig P. *Artificial Intelligence: A Modern Approach, 2nd Edition.* Prentice Hall Series in Artificial Intelligence. Englewood Cliffs, New Jersey, 2002
11. Bunke, H. *Recent Developments In Graph Matching*. Proceedings of the 15th International. Conference. on Pattern Recognition, Barcelona, 2000, Vol 2, 117 – 124.

12. Brockwell, J. and Davis, R. *Introduction to Time Series and Forecasting, 2nd Edition*. Springer-Verlag, New York, NY, 2002.

13. Coble, J., *Relational Discovery in Sequentially-Connected Data Streams: Efficient Algorithms for Lossless Pattern Discovery and Change Detection*. University of Texas at Arlington, Doctoral Dissertation, May 2005.

14. Lee, L. *Measures of Distributional Similarity*. Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics, 1999, 25 – 32