2004－ICS－138　(8)
2004／12／4

Analysis of Hepatitis Dataset by Using Cl-GBI

Akira Mogi, Phu Chien Nguyen, Kouzou Ohara, Hiroshi Motoda
and Takashi Washio†

Constructing a classifier for structured data for which the attribute value type data representation is not straightforward is challenging. Graph is one such structured data and we developed DT-GBI in which B-GBI, a graph mining program, is recursively called to construct attributes in the form of subgraphs. B-GBI, although quite efficient, has some problem due to non backtracking greedy strategy with pairwise chunking. Overlapping subgraphs can't be searched and frequency counting of the same subgraph in a single graph has some ambiguity. Cl-GBI, recently proposed new approach that uses chunkingless chunks as pseud-nodes, solves this problem at the expense of computation cost. DT-ClGBI is an improved version of DT-GBI that uses Cl-GBI instead of B-GBI. Experimental results of DT-ClGBI are shown for hepatitis dataset with some discussion. The work is still in progress.

1. Introduction

Over the last few years there has been much research work on data mining in seeking for better performance. Better performance includes mining from structured data, which is a new challenge. Since structure is represented by proper relations and a graph can easily represent relations, knowledge discovery from graph-structured data poses a general problem for mining from structured data. Some examples amenable to graph mining are finding typical web browsing patterns, identifying typical substructures of chemical compounds, finding typical subsequences of DNA and discovering diagnostic rules from patient history records.

A majority of methods widely used for data mining are for data that do not have structure and that are represented by attribute-value pairs. Decision tree[10], and induction rules[1),8)] relate attribute values to target classes. Association rules often used in data mining also use this attribute-value pair representation. These method can induce rules such that they are easy to understand. However, the attribute-value pair representation is not suitable to represent a more general data structure such as graph-structured data, and there are problems that need a more powerful representation.

We developed an efficient graph mining program, Graph-Based Induction (GBI)[6),13)]. In short, GBI is a technique which was devised for the purpose of discovering typical patterns (general subgraphs and induced subgraphs) in a general graph data by recursively chunking two adjoining nodes. It can handle a graph data having loops (including self-loops) with labeled/unlabeled nodes and labeled/unlabeled, directed/undirected edges. GBI is very efficient because of its greedy search. GBI does not lose any information of graph structure after chunking, and it can use various evaluation functions in so far as they are based on frequency. Later an improved version called B-GBI (Beam-wise Graph-Based Induction)[7)] adopting the beam search was proposed to increase the search space, thus extracting more discriminative patterns while keeping the computational complexity within a tolerant level.

Further, we developed a decision tree learner DT-GBI for graph structured data, using B-GBI internally to construct attributes[2)]. Since attributes for graph structured data are not defined in advance, the role of B-GBI is to extract candidate subgraphs that are discriminative. DT-GBI has been applied to analyze promoter dataset[11)] in UCI repository and hepatitis dataset provided by Chiba University and has shown its usefulness.

However, we found some problems in use of B-GBI in DT-GBI. Admitting that the search in GBI is greedy and no backtracking is made, which makes the search incomplete and leaves many patterns unextracted, one of the drawbacks of B-GBI is that it cannot find overlap-

† Institute of Scientific and Industrial Research, Osaka University, 8-1 Mihogaoka, Ibaraki, Osaka 567-0047, Japan.

ping patterns due to the nature of chunking. Further we noticed the counting in B-GBI is not accurate in some situations. The fact that B-GBI finds a pattern does not necessarily mean that it can find all of its subpatterns. What B-GBI can find must be along the chunking path. Thus, even if a pattern exists in a graph, B-GBI is not guaranteed to find it always.

We improved B-GBI using still a notion of pairwise chunking but actually without chunking to alleviate these problems. The new B-GBI is called Cl-GBI (Chunkinless GBI) and is incorporated into DT-GBI, now called DT-ClGBI.

In what follows, we briefly summarize problems caused by chunking, explain Cl-GBI and DT-ClGBI, and show some results obtained for hepatitis dataset. The work is still in progress and what is shown here is preliminary.

2. Problems Caused by Chunking in B-GBI

B-GBI increases the search space by running GBI in parallel. A certain fixed number of pairs ranked from the top are selected to be chunked individually in parallel. To prevent each branch growing exponentially, the total number of pairs to chunk (the beam width b) is fixed at every time of chunking. Thus, at any iteration step, there is always a fixed number of chunking that is performed in parallel. This will prevent some of the overlapping patterns from being undiscovered. For example, suppose in Fig. 1 the pair B – C is most frequent, followed by the pair A – B. When b=1, there is no way that a pattern A – B – D is discovered because B-C is chunked first, but by setting b=2, A – B can be chunked in the second beam and if A – B – D is frequent enough, there is a chance that (A – B) – D is chunked at next iteration. However, setting b very large is prohibitive from the computational point of view, and B-GBI cannot solve the problem of overlapping subgraphs completely because chunking process is involved.

Any subgraph that B-GBI can find is along the way in the chunking process. Thus, it happens that a pattern found in one input graph is unable to be found in the other input graph even if it does exist in the graph. An example is shown in Fig. 2, where even if the pair A – B is

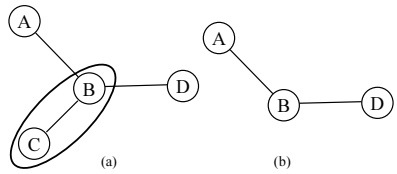

Fig. 1 Missing patterns due to chunking order.

selected for chunking and the structure D – A – B – C exists in the input graphs, we may not be able to find that structure because an unexpected pair A – B is chunked (see Fig. 2(b)). This causes a serious problem in counting the frequency of a pattern.

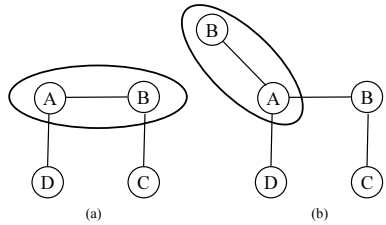

Fig. 2 A pattern is found in one input graph but not in the other.

The complete graph mining algorithms (such as AGM[3], AcGM[4], FSG[5], gSpan[12], etc.) do not face the problem of overlapping subgraphs since they can find all frequent patterns in the graph data. However, these methods are designed to find existence or non-existence of a certain pattern in one transaction and not to count how many times a certain pattern appear in one transaction. They also cannot give us the positions of each pattern in any graph transaction which is required by non-expert users. GBI (and thus B-GBI), on the other hand, is designed to find (not all) typical patterns in either a large single graph or a set of graphs but it cannot detect the positions of patterns.

There is another problem of overlapping subgraphs. Think of the graph in Fig. 3(a).

Suppose that the problem here is to find frequent connected induced subgraphs that occur at least 3 times in the graph. Figure 3(c) shows an example of frequent induced subgraph which has the support of 3. B-GBI cannot find this pattern because overlapping patterns cannot be found due to chunking unless b is set large enough. This example also explain the problem of use of downward closure property

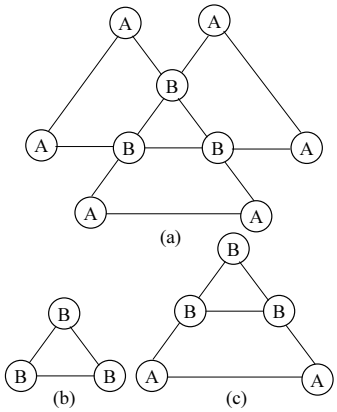

Fig. 3 An example of finding frequent patterns in a single graph.

of graph subsumption. The pattern shown by Fig. 3(b) is a subgraph of the pattern (c). However, it occurs only once in the graph. Thus, AcGM[4] cannot find this pattern because it is using downward closure property.

We have devised a novel algorithm that can overcome the problem of overlapping subgraphs causing problems in B-GBI. The proposed algorithm, called Cl-GBI (Chunkingless Graph-Based Induction), employs a "chunkingless" strategy, where frequent pairs are never chunked but used as pseud-nodes in the subsequent steps, thus allowing extraction of overlapping subgraphs. It can also give the positions of patterns present in each graph transaction as well as be applied to find frequent patterns in a single large graph or graph datasets.

3. Chunkingless Graph-Based Induction (Cl-GBI)

3.1 Approach

The basic ideas of Cl-GBI are as follows. Those pairs that connect two adjoining nodes in the graphs are counted and the top b (beam width) frequent pairs are selected. In B-GBI, graphs in the respective states are then copied into b states, each of which corresponds to one of the b selected pairs. The selected pair (in each of the b states) is registered as one node and this node is assigned a new label. Then, the graphs in each state are rewritten by replacing all the occurrences of the selected pair with a node with the newly assigned label (pair-wise chunking).

In Cl-GBI, we also register the b selected pairs as new nodes and assign b new labels to them. But those pairs are never chunked and the graphs are not "compressed". Thus, there is no need to copy the graphs into b states as in B-GBI. In the presence of the pseud-nodes (i.e., newly assigned-label nodes), we count the frequencies of pairs consisting of at least one pseud-node. The other can be either one of pseud-nodes including those already created in the previous levels or an original one. In other words, the other is one of the existing nodes. Among the remaining pairs (after selecting the most b frequent pairs) and the new pairs which have just been counted, we select the most b frequent pairs again and so on.

These steps are repeated N_e times, each of which is referred to as a level. Those pairs that satisfy a typicality criterion (e.g., Information Gain exceeds a given threshold) among all the pairs extracted in all the levels (i.e., from level 1 to level N_e) are the output of the algorithm.

A frequency threshold is used to reduce the number of pairs being considered to be typical patterns. Another possible method to reduce the number of pairs is to eliminate pairs whose typicality measure is below its threshold even if their frequency count is above the threshold. The two parameters b and N_e control the search space. Frequency threshold is another important parameter.

As in B-GBI, the Cl-GBI approach can handle both directed and undirected graphs. It also can handle both general subgraphs and induced subgraphs.

3.2 Algorithm of Cl-GBI

Input A graph database, two natural numbers b (beam width) and N_e (number of levels), and a frequency threshold θ.

Step 1 Extract all the pairs consisting of connected two nodes in the graphs, register their positions using node id (identifier) lists, and count their frequencies. Since the 2^{nd} level, extract all the pairs consisting of connected two nodes with at least one of which is a newly assigned-label node, the other can be either a newly assigned-label node including those already created in the previous levels or an original one (i.e., the other is one of the existing nodes).

Step 2 Select the most b frequent pairs from

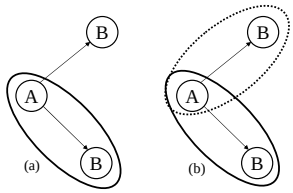

Fig. 4 An example of frequency counting.

among the pairs extracted at Step 1 (since the 2^{nd} level, from among the unselected pairs in the previous levels and the newly extracted pairs). Each of the b selected pairs is registered as a new node. If either or both nodes of the selected pair are not in the set of original nodes of the graphs (i.e., they are newly assigned-label nodes), they are restored to the original patterns before registration.

Step 3 Assign a new label to each pair selected at Step 2 but do not rewrite the graphs. Go back to Step 1.

These steps are repeated N_e times (N_e levels). All the pairs extracted at Step 1 in all the levels (i.e. level 1 to level N_e) including those that are not newly labeled are ranked based on a typicality criterion (e.g., Information Gain). It is worth noting that those pairs that have frequency count below a frequency threshold θ are eliminated, which means that there are three parameters b, N_e, θ to control the search.

We use the same canonical labeling employed in[7] to count the number of occurrences of a pattern in a graph transaction. However, canonical label alone cannot solve the frequency counting problem completely as shown in Fig. 4. Suppose that the pair A $\rightarrow$ B is registered as a pseud-node N in Fig. 4(a). How many times the pair N $\rightarrow$ B should be counted? If only canonical label is considered, the answer is 2 as shown in Fig. 4(b). However, N $\rightarrow$ B should be counted once. We solved this problem by incorporating the canonical label with the node id set. If both the canonical label and the node id set are identical for two subgraphs, we regard that they are the same and count once.

The output of Cl-GBI algorithm is a set of ranked typical patterns, each of which comes together with the positions of occurrences in every transaction of the graph data (given by node id lists) as well as the number of occurrences in each graph transaction.

The above algorithm was implemented in C++ and tested against artificial data and it was conformed to work as expected. For example, Cl-GBI was able to find all (a total of 35) frequent induced subgraphs, including the one shown in Fig. 3(c), in a few seconds.

However, as is easily predicted, this algorithm can find all the subgraphs by setting b and N_e large enough. One of the nice aspects of B-GBI is that the size of the input graph keep reducing progressively as the chunking proceeds, and thus the number of pairs to consider also progressively decreases accordingly. In case of Cl-GBI, the number of pairs to consider keeps increasing because pseud-nodes keeps increasing as the search proceeds. Thus, it is important to select appropriate values for b and N_e.

3.3 Unsolved problem of Cl-GBI

We found that there is still a problem in frequency counting that use of both the canonical label and the node id set cannot solve. Think of the graph in Fig. 5(a). The three subgraphs A – A – A illustrated in Figs. 5 (b), (c), and (d) share the same canonical label and the same node id set. Our current Cl-GBI cannot distinguish between these three. However, this problem arises only in general subgraph. It causes no problem in case of enumerating frequent induced subgraphs.

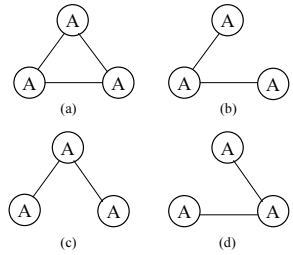

Fig. 5 Counting frequency of general subgraphs.

4. Decision Tree based on Chunkingless Graph-Based Induction (DT-ClGBI)

Since the value for an attribute is either yes (the classifying pattern exists) or no (the classifying pattern does not exist), the constructed decision tree is represented as a binary tree. Data (graphs) are divided into two groups, namely, the one with the pattern and the other

without the pattern. The above process is summarized in Fig. 6.

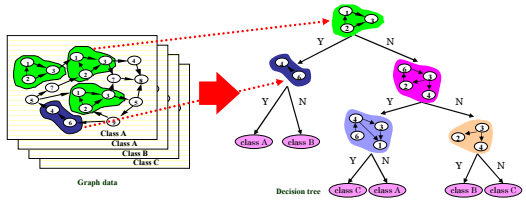

Fig. 6 Decision tree for classifying graph-structured data

4.1 Feature Construction by Cl-GBI

In DT-ClGBI discriminative patterns are extracted using Cl-GBI and used as attributes for classifying graph-structured data. When constructing a decision tree, all the pairs (subgraphs) in data are enumerated within the capacity of Cl-GBI and one pair is selected. The data (graphs) are divided into two groups, namely, the one with the pair and the other without the pair. This process is recursively applied at each node of a decision tree and a decision tree is constructed while attributes (pairs) for classification task are created on the fly. The algorithm of DT-ClGBI is summarized in Figure 7. Parameters b, N_e and θ can be different at each node. However, what has been found in the upstream nodes are passed down below not to repeat the same finding. Cl-GBI is used as a method for feature construction, since features (subgraphs) useful for classification task are constructed during the tree construction.

4.2 Pruning Decision Tree

Once the tree is constructed, "pessimistic pruning"[10] is performed to improve predictive accuracy based on the confidence interval for binomial distribution.

5. Experiments

We tested DT-ClGBI to hepatitis dataset to assess its performance. The work is still in progress and we report here only the initial results. We used the same dataset of experiment 1 (prediction of fibrosis) and experiment 4 (interferon therapy) reported in[9]. The average size of the graph for each experiment is 303 and 75, and the number of graphs for each experiment is 108 (F0+F1:65, F4:43) and 94 (response:38, non-response:56).

Because the computation time of Cl-GBI that is called from inside of DT-ClGBI is much larger than our estimation, we had to set the values of b and N_e small and the value of θ large. For example, one run of Cl-GBI with b=3, N_e=3, θ=0.5 took 24 min. by a Linux machine with 1.6 GHz Athlon XP 1900+ and 3GB Memory for the dataset in experiment 1. However, Cl-GBI with b=6, N_e=8, θ=0.5 for the same dataset could not return results after 2 days. Thus, we only run Cl-GBI at the root node and set N_e=0 for the other nodes, meaning that we only used the patterns constructed at the root node. We were not able to run 10 fold cross validation for these datasets. Thus, we only show one decision tree for each case with associated patterns for some nodes.

```
DT-ClGBI(D)
  Create a node DT for D
  if termination condition reached
    return DT
  else
    P := Cl-GBI(D) (with b, N_e, θ specified)
    Select the best pair p from P
    Divide D into D_y (with p) and D_n (without p)
    for D_i := D_y, D_n
      DT_i := DT-ClGBI(D_i)
      Augment DT by attaching DT_i as its
      child along yes(no) branch
  return DT
```

Fig. 7 Algorithm of DT-GBI

Figure 8 is an example of decision tree for experiment 1. The tree size is 29, and its predictive accuracy is 75%. Figure 9 is an example of decision tree for experiment 4. The tree size is 16 and its predictive accuracy is 78%. Since the size is large, only the first four levels are shown. The patterns that appear in the nodes are small. The number of subgraphs found by Cl-GBI for each experiment is 859 and 4858. However, with the parameters used, the size of the maximum subgraph is only 4 and 10 for each experiment. This explains why the tree size is large and the accuracy is not good enough. Lowering θ is required and for this to be practical, more efficient implementation of Cl-GBI and good heuristic for ranking pairs are required. The more detailed results and the progress made thereafter will be reported later.

6. Conclusion

DT-ClGBI was introduced as a successor of

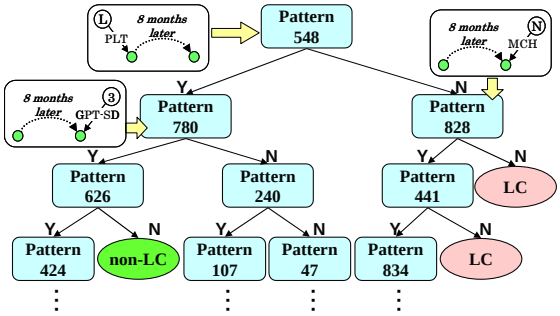

Fig. 8 An example of decision tree for fibrosis prediction (b=3, N_e=3, θ=0.5)

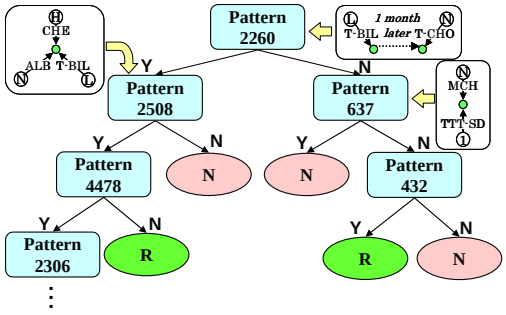

Fig. 9 An example of decision tree for interferon therapy prediction (b=3, N_e=20, θ=0.5)

DT-GBI using Cl-GBI as an attribute constructor. Cl-GBI employs the "chunkingless" strategy which helps overcome the problem of overlapping subgraphs and can give correct counting. The initial results obtained by applying it to hepatitis dataset revealed that the number of subgraphs discovered is much larger, and more efficient implementation and good heuristic are required to speed up Cl-GBI to extract larger subgraphs. We are now working on these problems.

Acknowledgments This work was partially supported by the grant-in-aid for scientific research on priority area "Active Mining" (No. 13131101, No. 13131206) funded by the Japanese Ministry of Education, Culture, Sport, Science and Technology.

References

1) Clark, P., Niblett, T.: The CN2 Induction Algorithm, *Machine Learning*, Vol. 3, pp. 261–283 (1989).
2) Gaemsakul, W., Matsuda, T., Yoshida, T., Motoda, M., and Washio, T.: Classifier Construction by Graph-Based Induction for Graph-Structured Data, *Proc. of the 7th PAKDD*, pp. 52–62 (2003).
3) Inokuchi, A., Washio, T., and Motoda, H.: Complete Mining of Frequent Patterns from Graphs: Mining Graph Data, *Machine Learning Journal*, Vol. 50, No. 3, pp. 321–354 (2003).
4) Inokuchi, A., Washio, T., Nishimura, K. and Motoda, H.: A Fast Algorithm for Mining Frequent Connected Subgraphs, *IBM Research Report RT0448*, Tokyo Research Laboratory, IBM Japan (2002).
5) Kuramochi, M., Karypis, G.: Frequent Subgraph Discovery, *Proc. of ICDM01*, pp.313–320 (2001).
6) Matsuda, T., Horiuchi, T., Motoda, H., Washio, T.: Extension of Graph-Based Induction for General Graph Structured Data, *Knowledge Discovery and Data Mining: Current Issues and New Applications (Springer Verlag LNAI 1805)*, pp. 420–431 (2000).
7) Matsuda, T., Motoda, H., Yoshida, T., and Washio, T.: Mining Patterns from Structured Data by Beam-wise Graph-Based Induction, *Proc. of DS02*, pp. 422–429 (2002).
8) Michalski, R. S.: Learning Flexible Concepts: Fundamental Ideas and a Method Based on Two-Tiered Representation, *In Machine Learning, An Artificial Intelligence Approach*, Vol. 3, pp. 63–102, (1990).
9) Ohara, K., Yoshida, T., Geamsakul, W., Motoda, H., Washio, T., Yokoi, H and Takabayashi, K.: Analysis of Hepatitis Dataset by Decision Tree Graph-Based Induction, *Proc of Discovery Challenge, Workshop held in conjunction with the 8th PKDD*, pp. 173–184 (2004).
10) Quinlan, J.R.: *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers (1993).
11) Towell, G. G., Shavlik, J. W.: Extracting Refined Rules from Knowledge-Based Neural Networks, *Machine Learning*, Vol. 13, pp. 71–101 (1993).
12) Yan, X., Han, J: gSpan: Graph-Based Structure Pattern Mining, *Proc. ICDM02*, pp. 721–724 (2002).
13) Yoshida, K., Motoda, M.: CLIP: Concept Learning from Inference Pattern, *Artificial Intelligence Journal*, Vol. 75, No. 1, pp. 63–92 (1995).