Fundamenta Informaticae XXI (2001) 1001–1030
IOS Press

Constructing a Decision Tree for Graph-Structured Data and its Applications

Warodom Geamsakul
Institute of Scientific and Industrial Research
Osaka University, JAPAN
warodom@ar.sanken.osaka-u.ac.jp

Tetsuya Yoshida
Institute of Scientific and Industrial Research
Osaka University, JAPAN
yoshida@ar.sanken.osaka-u.ac.jp

Kouzou Ohara
Institute of Scientific and Industrial Research
Osaka University, JAPAN
ohara@ar.sanken.osaka-u.ac.jp

Hiroshi Motoda
Institute of Scientific and Industrial Research
Osaka University, JAPAN
motoda@ar.sanken.osaka-u.ac.jp

Hideto Yokoi
Division for Medical Informatics
Chiba University Hospital, JAPAN
yokoi@telemed.ho.chiba-u.ac.jp

Katsuhiko Takabayashi
Division for Medical Informatics
Chiba University Hospital, JAPAN
takaba@ho.chiba-u.ac.jp

Abstract. A machine learning technique called Graph-Based Induction (GBI) efficiently extracts typical patterns from graph-structured data by stepwise pair expansion (pairwise chunking). It is very efficient because of its greedy search. Meanwhile, a decision tree is an effective means of data classification from which rules that are easy to understand can be obtained. However, a decision tree could not be constructed for the data which is not explicitly expressed with attribute-value pairs. This paper proposes a method called Decision Tree Graph-Based Induction (DT-GBI), which constructs a classifier (decision tree) for graph-structured data while simultaneously constructing attributes for classification using GBI. Substructures (patterns) are extracted at each node of a decision tree by stepwise pair expansion in GBI to be used as attributes for testing. Since attributes (features) are constructed while a classifier is being constructed, DT-GBI can be conceived as a method for feature construction. The predictive accuracy of a decision tree is affected by which attributes (patterns) are used and how they are constructed. A beam search is employed to extract good enough discriminative patterns within the greedy search framework. Pessimistic pruning is incorporated to avoid overfitting to the training data. Experiments using a DNA dataset were conducted to see the effect of the beam width and the number of chunking at each node of a decision tree. The results indicate that DT-GBI that uses very little prior domain knowledge can construct a decision tree that is comparable to other classifiers constructed using the domain knowledge. DT-GBI was also applied to analyze a real-world hepatitis dataset as a part of evidence-based medicine. Four classification tasks of the hepatitis data were conducted using only the time-series data of blood inspection and urinalysis. The

Address for correspondence: Institute of Scientific and Industrial Research, Osaka University, JAPAN
8-1 Mihogaoka, Ibaraki, Osaka 567-0047, Japan

preliminary results of experiments, both constructed decision trees and their predictive accuracies as well as extracted patterns, are reported in this paper. Some of the patterns match domain experts' experience and the overall results are encouraging.

Keywords: graph mining, graph-based induction, decision tree, beam search, evidence-based medicine

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 [23, 24], and induction rules [17, 5] relate attribute values to target classes. Association rules often used in data mining also uses 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.

A powerful approach that can handle relation and thus, structure, would be Inductive Logic Programming (ILP) [19] which employs Predicate Logic or its subset such as Prolog (Horn clauses) as its representation language. It has a merit that domain knowledge and acquired knowledge can be utilized as background knowledge, and has been applied to many application areas [7], especially to the domain of computational chemistry [18, 22]. However ILP systems tend to be less efficient in computational time and space than other machine learning systems because they have to search a (possibly very large) lattice of clauses and test the coverage of clauses during the search process [22]. Although many approaches to improve the efficiency of ILP have been proposed [3, 22], still more work on the search is needed [22]. One approach commonly adopted by most ILP algorithms to reduce the search space is to introduce declarative bias [20] such as types and input/output modes of predicate arguments, which is provided by a user as a part of background knowledge. Well designed biases can effectively reduce the search space without sacrificing predictive accuracy of learned knowledge, although ill designed ones would affect the accuracy adversely, as well as the efficiency of the algorithm. However it is not always easy for a user who is not familiar with logic programming to correctly specify an effective bias in a manner which is specific to each system.

AGM (Apriori-based Graph Mining) [11, 12] was developed for the purpose of mining the association rules among the frequently appearing substructures in a given graph data set. A graph transaction is represented by an adjacency matrix, and the frequent patterns appearing in the matrices are mined through the extended algorithm of the basket analysis. This algorithm can extract all connected/disconnected substructures by complete search. However, its computation time increases exponentially with input graph size and support.

SUBDUE [6] is an algorithm for extracting subgraphs which can best compress an input graph based on MDL (Minimum Description Length) principle. The found substructures can be considered concepts. This algorithm is based on a computationally-constrained beam search. It begins with a substructure comprising only a single vertex in the input graph, and grows it incrementally expanding a node in it. At each expansion it evaluates the total description length (DL) of the input graph which is defined as the sum of the two: DL of the substructure and DL of the input graph in which all the instances of the substructure are replaced by single nodes. It stops when the substructure that minimizes the total description length is found. In this process SUBDUE can find any number of substructures in the input graph based on a user-defined parameter (default 3). After the best substructure is found, the input graph is rewritten by compressing all the instances of the best structure, and the next iteration starts using the rewritten graph as a new input. This way, SUBDUE finds more abstract concepts at each round of iteration. However, it does not maintain strictly the original input graph structure after compression because its aim is to facilitate the global understanding of the complex database by forming hierarchical concepts and using them to approximately describe the input data.

Graph-Based Induction (GBI) [30, 14] is a technique which was devised for the purpose of discovering typical patterns in a general graph data by recursively chunking two adjoining nodes. It can handle a graph data having loops (including self-loops) with colored/uncolored nodes and 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. It is not, however, suitable for graph-structured data where many nodes share the same label because of its greedy recursive chunking without backtracking, but it is still effective in extracting patterns from such graph-structured data where each node has a distinct label (*e.g.,* World Wide Web browsing data) or where some typical structures exist even if some nodes share the same labels (*e.g.,* chemical structure data containing benzene rings etc).

This paper proposes a method called Decision Tree Graph-Based Induction (DT-GBI), which constructs a classifier (decision tree) for graph-structured data while simultaneously constructing attributes for classification using GBI [27, 28]. Substructures (patterns) are extracted at each node of a decision tree by stepwise pair expansion (pairwise chunking) in GBI to be used as attributes for testing. Since attributes (features) are constructed while a classifier is being constructed, DT-GBI can be conceived as a method for feature construction. The predictive accuracy of a decision tree is affected by which attributes (patterns) are used and how they are constructed. A beam search is employed to extract good enough discriminative patterns within the greedy search framework. Pessimistic pruning is incorporated to avoid overfitting to the training data. To classify graph-structured data after the construction of a decision tree, attributes are produced from data before the classification.

There are two systems which are close to DT-GBI in the domain of ILP: TILDE [2] and S-CART [13]. Although they were developed independently, they share the same theoretical framework and can construct a first order logical decision tree, which is a binary tree where each node in the tree is associated with either a literal or a conjunction of literals. Namely each node can represent a relational or structured data like DT-GBI. However in both systems, all available literals and conjunctions that can appear in a node have to be defined as declarative biases beforehand. In other words, although they can utilize a substructure represented by one or more literals to generate a new node in a decision tree, available structures are limited to those that are predefined. In contrast, DT-GBI can construct a pattern or substructure used in a node of a decision tree from a substructure that has been constructed before as well as from an initial node in the original graph-structured data.

DT-GBI was evaluated against a DNA dataset from UCI repository and applied to analyze a real-world hepatitis dataset which was provided by Chiba University Hospital as a part of evidence-based medicine. Since DNA data is a sequence of symbols, representing each sequence by attribute-value pairs by simply assigning these symbols to the values of ordered attributes does not make sense. The sequences were transformed into graph-structured data and the attributes (substructures) were extracted by GBI to construct a decision tree. To analyze the hepatitis dataset, temporal records in the provided dataset was converted into graph-structured data with respect to time correlation so that both intra-correlation of individual inspection and inter-correlation among inspections are represented in graph-structured data. Four classification tasks of the hepatitis data were conducted by regarding as the class labels: the stages of fibrosis (experiment 1 and 2), the types of hepatitis (experiment 3), and the effectiveness of interferon therapy (experiment 4). Decision trees were constructed to discriminate between two groups of patients using no biopsy results but only the time sequence data of blood inspection and urinalysis. The results of experiments are reported and discussed in this paper.

There are some other analyses already conducted and reported on these datasets. For the promoter dataset, [26] reports the results obtained by the M-of-N expression rules extracted from KBANN (Knowledge Based Artificial Neural Network). The obtained M-of-N rules are complicated and not easy to interpret. Furthermore, domain knowledge is required in KBANN to configure the initial artificial neural network. [15, 16] report another approach to construct a decision tree for the promoter dataset using the extracted patterns (subgraphs) by pairwise chunking as attributes and C4.5. For the hepatitis dataset, [29] analyzed the data by constructing decision trees from time-series data without discretizing numeric values. [10] proposed a method of temporal abstraction to handle time-series data, converted time phenomena to symbols and used a standard classifier. [9] used multi-scale matching to compare time-series data and clustered them using rough set theory. [21] also clustered the time-series data of a certain time interval into several categories and used a standard classifier. These analyses examine the temporal correlation of each inspection separately and do not explicitly consider the relations among inspections. Thus, these approaches do not correspond to the structured data analysis.

Section 2 briefly describes the framework of GBI. Section 3 explains our method called DT-GBI for constructing a classifier with GBI for graph-structured data and illustrates how a decision tree is constructed with a simple example. Section 4 describes the experiments using a DNA dataset to see the effect of the beam width and the number of chunking at each node of a decision tree. Section 5 describes the application DT-GBI to the hepatitis dataset and reports the results of experiments: constructed decision trees, their predictive accuracies and extracted discriminative patterns. Section 6 concludes the paper with a summary of the results and the planned future work.

2. Graph-Based Induction (GBI)

2.1. Principle of GBI

GBI employs the idea of extracting typical patterns by stepwise pair expansion as shown in Figure 1. In the original GBI an assumption is made that typical patterns represent some concepts/substructures and "typicality" is characterized by the pattern's frequency or the value of some evaluation function of its frequency. We can use statistical indexes as an evaluation function, such as frequency itself, Information Gain [23], Gain Ratio [24] and Gini Index [4], all of which are based on frequency. In Figure 1 the shaded pattern consisting of nodes 1, 2, and 3 is thought typical because it occurs three times in the

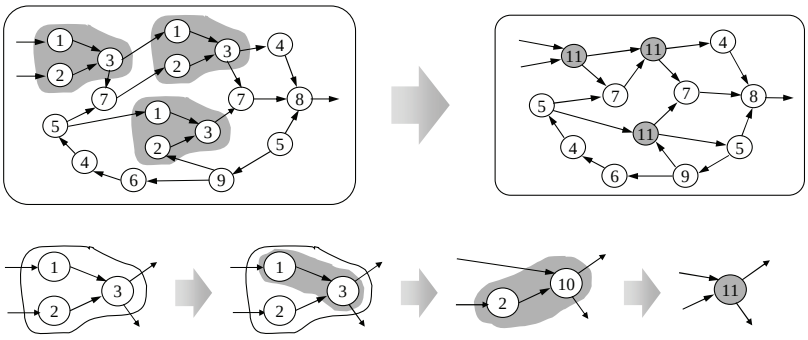

Figure 1. The basic idea of the GBI method

GBI(G)
  Enumerate all the pairs P_{all} in G
  Select a subset P of pairs from P_{all} (all the pairs in G)
    based on typicality criterion
  Select a pair from P_{all} based on chunking criterion
  Chunk the selected pair into one node c
  G_c := contracted graph of G
  while termination condition not reached
    $P := P \cup$ GBI(G_c)
  return P

Figure 2. Algorithm of GBI

graph. GBI first finds the 1→3 pairs based on its frequency, chunks them into a new node 10, then in the next iteration finds the 2→10 pairs, chunks them into a new node 11. The resulting node represents the shaded pattern.

It is possible to extract typical patterns of various sizes by repeating the above procedures. Note that the search is greedy. No backtracking is made. This means that in enumerating pairs no pattern which has been chunked into one node is restored to the original pattern. Because of this, all the "typical patterns" that exist in the input graph are not necessarily extracted. The problem of extracting all the isomorphic subgraphs is known to be NP-complete. Thus, GBI aims at extracting only meaningful typical patterns of a certain size. Its objective is not finding all the typical patterns nor finding all the frequent patterns.

As described earlier, GBI can use any criterion that is based on the frequency of paired nodes. However, for finding a pattern that is of interest any of its subpatterns must be of interest because of the nature of repeated chunking. In Figure 1 the pattern 1→3 must be typical for the pattern 2→10 to be typical. Said differently, unless pattern 1→3 is chunked, there is no way of finding the pattern 2→10. The frequency measure satisfies this monotonicity. However, if the criterion chosen does not satisfy this monotonicity, repeated chunking may not find good patterns even though the best pair based on the criterion is selected at each iteration. To resolve this issue GBI was improved to use two criteria, one for frequency measure for chunking and the other for finding discriminative patterns after chunking. The

latter criterion does not necessarily hold monotonicity property. Any function that is discriminative can be used, such as Information Gain [23], Gain Ratio [24] and Gini Index [4], and some others.

The improved stepwise pair expansion algorithm is summarized in Figure 2. It repeats the following four steps until the chunking threshold is reached (normally minimum support value is used as the stopping criterion).

Step 1 Extract all the pairs consisting of connected two nodes in the graph.

Step 2a Select all the typical pairs based on the typicality criterion from among the pairs extracted in Step 1, rank them according to the criterion and register them as typical patterns. If either or both nodes of the selected pairs have already been rewritten (chunked), they are restored to the original patterns before registration.

Step 2b Select the most frequent pair from among the pairs extracted in Step 1 and register it as the pattern to chunk. If either or both nodes of the selected pair have already been rewritten (chunked), they are restored to the original patterns before registration. Stop when there is no more pattern to chunk.

Step 3 Replace the selected pair in Step 2b with one node and assign a new label to it. Rewrite the graph by replacing all the occurrences of the selected pair with a node with the newly assigned label. Go back to Step 1.

The output of the improved GBI is a set of ranked typical patterns extracted at Step 2a. These patterns are typical in the sense that they are more discriminative than non-selected patterns in terms of the criterion used.

Note that GBI is capable of restoring the original pattern represented by the initial nodes and edges from a chunk (Step 2a, Step 2b). This is made possible because we keep track of the information about to which node within the chunk an edge is connected to during the recursive chunking process. Further, GBI can limit the type of subgraph to induced subgraph. In this case, a pair that has more than one edge is chunked into one node in one step.

2.2. Beam-wise Graph-Based Induction (B-GBI)

Since the search in GBI is greedy and no backtracking is made, which patterns are extracted by GBI depends on which pair is selected for chunking. There can be many patterns which are not extracted by GBI. A beam search is incorporated to GBI, still, within the framework of greedy search [15] in order to relax this problem, increase the search space, and extract more discriminative patterns while still keeping the computational complexity within a tolerant level. 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) 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.

Figure 3 shows how search is conducted in B-GBI when beam width is set to five. First, five frequent pairs are selected from the graphs at the starting state in search (cs in Figure 3). Graphs in cs are then copied into the five states (c11 $\sim$ c15)[1], and each of five pairs is chunked in the copied graphs at the

[1]The current implementation is very naive. All the beam copies of the graph are kept in memory and, within each beam, pair is

respective state. At the second cycle in search, pairs in graphs are enumerated in each state and five frequent pairs are selected from all the sates. In this example, two pairs are selected from c11, one pair from c13, and two pairs from c14.

At the third cycle in search, graphs in c11 are copied into c21 and c22, graphs in c13 are copied into c23, and graphs in c24 are copied into c24 and c25. As in the second cycle, the selected pairs are chunked in the copied graphs. The states without the selected pairs (in this example c12 and c15) are discarded.

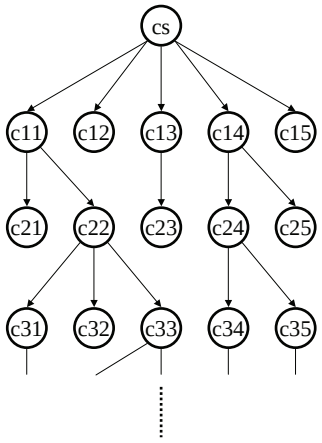

Figure 3. Beam search in B-GBI (beam width = 5)

2.3. Canonical Labeling

GBI assigns a new label for each newly chunked pair. Because it recursively chunks pairs, it happens that the new pairs that have different labels because of different chunking history happen to be the same pattern (subgraph).

To identify whether the two pairs represent the same pattern or not, each pair is represented by its canonical label[25, 8] and only when the label is the same, they are regarded as identical. The basic procedure of canonical labeling is as follows. Nodes in the graph are grouped according to their labels (node colors) and the degrees of node (number of edges attached to the node) and ordered lexicographically. Then an adjacency matrix is created using this node ordering. When the graph is undirected, the adjacency matrix is symmetric, and the upper triangular elements are concatenated scanning either horizontally or vertically to codify the graph. When the graph is directed, the adjacency matrix is asymmetric, and all the elements in both triangles are used to codify the graph in a similar way. If there are more than one node that have identical node label and identical degrees of node, the ordering which results in the maximum (or minimum) value of the code is searched. The corresponding code is the canonical label. Let M be the number of nodes in a graph, N be the number of groups of the nodes, and $p_i (i = 1, 2, \ldots, N)$ be the number of the nodes within group i. The search space can be reduced to $\prod_{i=1}^{N}(p_i!)$ from $M!$ by this grouping and lexicographical ordering. The code of an adjacency matrix for the case in which elements in the upper triangle are vertically concatenated is defined as

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ & a_{22} & \ldots & a_{2n} \\ & & \ddots & \vdots \\ & & & a_{nn} \end{pmatrix} \qquad code(A) = a_{11}a_{12}a_{22}a_{13}a_{23}\ldots a_{nn} \quad (1)$$

$$= \sum_{j=1}^{n}\sum_{i=1}^{j}\left((L+1)^{\{(\sum_{k=j+1}^{n} k)+j-i\}}a_{ij}\right). \quad (2)$$

Here L is the number of different edge labels. We choose the option of vertical concatenation. It is possible to further prune the search space. Elements of the adjacency matrix of higher ranked nodes form higher elements (digits) of the code. Thus, once the locations of higher ranked nodes in the adjacency

counted independently of the other beams. Thus, many of the pair countings are redundant. We expect that code optimization can reduce the computation time to less than half.

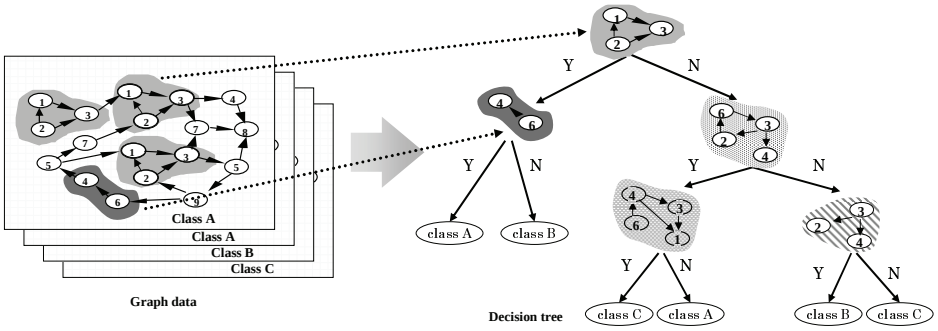

Figure 4. Decision tree for classifying graph-structured data

matrix are fixed, corresponding higher elements of the code are also fixed and are not affected by the order of elements of lower ranks. This reduces the search space of $\prod_{i=1}^{N}(p_i!)$ to $\sum_{i=1}^{N}(p_i!)$.

However, there is still a problem of combinatorial explosion for a case where there are many nodes of the same labels and the same degrees of node such as the case of chemical compounds because the value of p_i becomes large. What we can do is to make the best of already determined nodes of higher ranks. Assume that the order of the nodes $v_i \in V(G)(i = 1, 2, \ldots, N)$ is already determined in a graph G. Consider finding the order of the nodes $u_i \in V(G)(i = 1, 2, \ldots, k)$ of the same group that gives the maximum code value. The node that comes to v_{N+1} is the one in $u_i(i = 1, \ldots, k)$ that has an edge to the node v_1 because the highest element that v_{N+1} can make is a_{1N+1} and the node that makes this element non-zero, that is, the node that is linked to v_1 gives the maximum code. If there are more than one node or no node at all that has an edge to v_{N+1}, the one that has an edge to v_2 comes to v_{N+1}. Repeating this process determines which node comes to v_{N+1}. If no node can be determined after the last comparison at v_N, permutation within the group is needed. Thus, the computational complexity in the worst case is still exponential. Our past experience indicates that computation required for canonical labeling is less than 10% of the total computation time in GBI [16].

3. Decision Tree Graph-Based Induction (DT-GBI)

3.1. Decision Tree for Graph-structured Data

We formulate the construction of a decision tree for graph-structured data by defining attributes and attribute-values as follows:

- attribute: a pattern/subgraph in graph-structured data
- value for an attribute: existence/non-existence of the pattern in a graph

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 Figure 4. One remaining question is how to determine classifying patterns which are used as attributes for graph-structured data. Our approach is described in the next subsection.

```
DT-GBI(D)
  Create a node DT for D
  if termination condition reached
    return DT
  else
    P := GBI(D) (with the number of chunking specified)
    Select a pair p from P
    Divide D into D_y (with p) and D_n (without p)
    Chunk the pair p into one node c
    D_yc := contracted data of D_y
    for D_i := D_yc, D_n
      DT_i := DT-GBI(D_i)
      Augment DT by attaching DT_i as its child along yes(no) branch
  return DT
```

Figure 5. Algorithm of DT-GBI

3.2. Feature Construction by GBI

In our Decision Tree Graph-Based Induction (DT-GBI) method, typical patterns are extracted using B-GBI and used as attributes for classifying graph-structured data. When constructing a decision tree, all the pairs in data are enumerated 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 as described above. The selected pair is then chunked in the former graphs and these graphs are rewritten by replacing all the occurrences of the selected pair with a new node. 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-GBI is summarized in Figure 5. As shown in Figure 5, the number of chunking applied at one node is specified as a parameter. For instance, when it is set to 10, chunking is applied 10 times to construct a set of pairs P, which consists of all the pairs constructed from each chunking (the 1st chunking to the 10th chunking). Note that the pair p selected from P is not necessarily constructed by the last chunking (*e.g.,* the 10th chunking). If it is at the 6-th chunking that is smaller than 10, the state is retracted to that stage and all the chunks constructed after the 6-th stage are discarded.

Each time when an attribute (pair) is selected to divide the data, the pair is chunked into a larger node in size. Thus, although initial pairs consist of two nodes and the edge between them, attributes useful for classification task are gradually grown up into larger pair (subgraphs) by applying chunking recursively. In this sense the proposed DT-GBI method can be conceived as a method for feature construction, since features, namely attributes (pairs) useful for classification task, are constructed during the application of DT-GBI.

Note that the criterion for chunking and the criterion for selecting a classifying pair can be different. In the following experiments, frequency is used as the evaluation function for chunking, and information gain is used as the evaluation function for selecting a classifying pair[2].

[2]We did not use information gain ratio because DT-GBI constructs a binary tree.

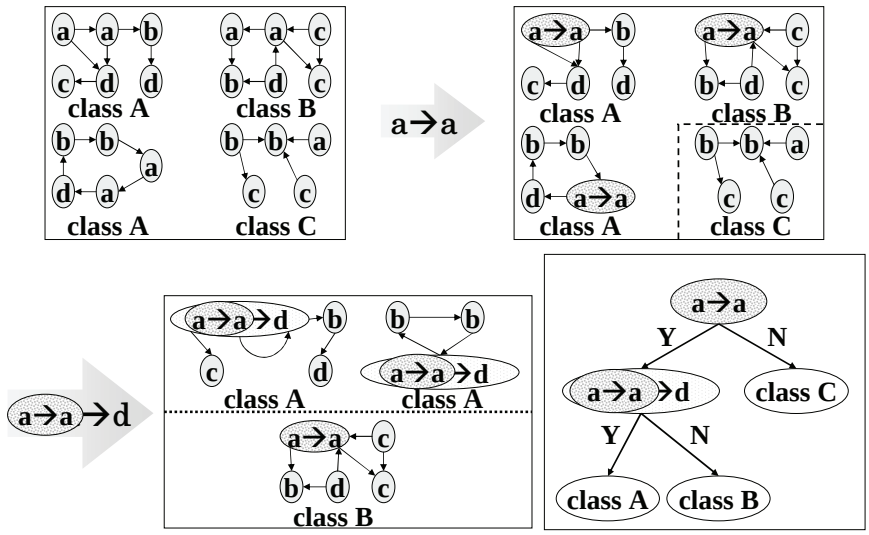

Figure 6. Example of decision tree construction by DT-GBI

Graph	a→a	a→b	a→c	a→d	b→a	b→b	b→c	b→d	c→b	c→c	d→a	d→b	d→c
1 (class A)	1	1	0	1	0	0	0	1	0	0	0	0	1
2 (class B)	1	1	1	0	0	0	0	0	0	1	1	1	0
3 (class A)	1	0	0	1	1	1	0	0	0	0	0	1	0
4 (class C)	0	1	0	0	0	1	1	0	1	0	0	0	0

Figure 7. Attribute-value pairs at the first step

3.3. Working example of DT-GBI

Suppose DT-GBI receives a set of 4 graphs in the upper left-hand side of Figure 6. The number of chunking applied at each node is set to 1 to simplify the working of DT-GBI in this example. By enumerating all the pairs in these graphs, 13 kinds of pairs are extracted from the data. These pairs are: a→a, a→b, a→c, a→d, b→a, b→b, b→c, b→d, c→b, c→c, d→a, d→b, d→c. The existence/non-existence of the pairs in each graph is converted into the ordinary table representation of attribute-value pairs, as shown in Figure 7. For instance, for the pair a→a, graph 1, graph 2 and graph 3 have the pair but graph 4 does not have it. This is shown in the first column in Figure 7.

Next, the pair with the highest evaluation for classification (i.e., information gain) is selected and used to divide the data into two groups at the root node. In this example, the pair "a→a" is selected. The selected pair is then chunked in graph 1, graph 2 and graph 3 and these graphs are rewritten. On the other hand, graph 4 is left as it is.

The above process is applied recursively at each node to grow up the decision tree while constructing the attributes (pairs) useful for classification task at the same time. Pairs in graph 1, graph 2 and graph 3 are enumerated and the attribute-value tables are constructed as shown in Figure 8. After selecting the pair "(a→a)→d", the graphs are separated into two partitions, each of which contains graphs of a single class. The constructed decision tree is shown in the lower right-hand side of Figure 6.

Graph	a→a→b	a→a→c	a→a→d	b→a→a	b→a		d→c
1 (class A)	1	0	1	0	0	…	1
2 (class B)	1	1	0	0	0	…	0
3 (class A)	0	0	1	1	1	…	0

Figure 8. Attribute-value pairs at the second step

3.4. Pruning Decision Tree

Recursive partitioning of data until each subset in the partition contains data of a single class often results in overfitting to the training data and thus degrades the predictive accuracy of decision trees. To avoid overfitting, in our previous approach [27] a very naive *prepruning* method was used by setting the termination condition in DT-GBI in Figure 5 to whether the number of graphs in D is equal to or less than 10. A more sophisticated *postpruning* method, is used in C4.5 [24] (which is called "pessimistic pruning") by growing an overfitted tree first and then pruning it to improve predictive accuracy based on the confidence interval for binomial distribution. To improve predictive accuracy, pessimistic pruning used in C4.5 is incorporated into the DT-GBI by adding a step for postpruning in Figure 5.

3.5. Classification using the Constructed Decision Tree

Unseen new graph data must be classified once the decision tree has been constructed. Here again, the problem of subgraph isomorphism arises to test if the input graph contains the subgraph specified in the test node of the tree. To alleviate this problem and to be consistent with the GBI basic principle, we also apply pairwise chunking to generate candidates of subgraphs. However, assuming that both the training and test data come from the same underlying distribution, no search is performed to decide which pair to chunk next. We remember the order of chunking during the construction of the decision tree and use this order to chunk each test data and check if one of the canonical labels of constructed chunks is equal to the canonical label of the test node. Since there is no search, the increase in computation time at the classification stage is negligible.

4. Application to Promoter Dataset

DT-GBI was tested using a DNA dataset in the UCI Machine Learning Repository[1]. A promoter is a genetic region which initiates the first step in the expression of an adjacent gene (*transcription*). The promoter dataset consists of strings that represent nucleotides (one of A, G, T, or C). The input features are 57 sequential DNA nucleotides and the total number of instances is 106 including 53 positive instances (sample promoter sequence) and 53 negative instances

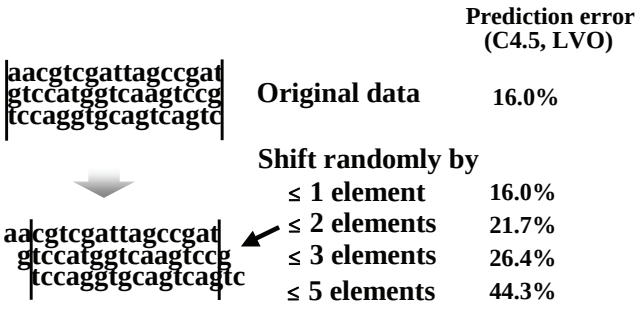

Figure 9. Change of error rate by shifting the sequence in the promoter dataset

(non-promoter sequence). This dataset was explained and analyzed in [26]. The data is so prepared that each sequence of nucleotides is aligned at a reference point, which makes it possible to assign the n-th attribute to the n-th nucleotide in the attribute-value representation. In a sense, this dataset is encoded using domain knowledge. This is confirmed by the following experiment. Running C4.5[24] with a standard setting gives a prediction error of 16.0% by leaving one out cross validation. Randomly shifting the sequence by 3 elements gives 21.7% and by 5 elements 44.3%. If the data is not properly aligned, standard classifiers such as C4.5 that use attribute-value representation do not solve this problem, as shown in Figure 9.

One of the advantages of graph representation is that it does not require the data to be aligned at a reference point. In our approach, each sequence is converted to a graph representation assuming that an element interacts up to 10 elements (See Figure 10)[3]. Each sequence thus results in a graph with 57 nodes and 515 edges. Note that a sequence is represented as a directed graph since it is known from the domain knowledge that the influence between nucleotides is directed.

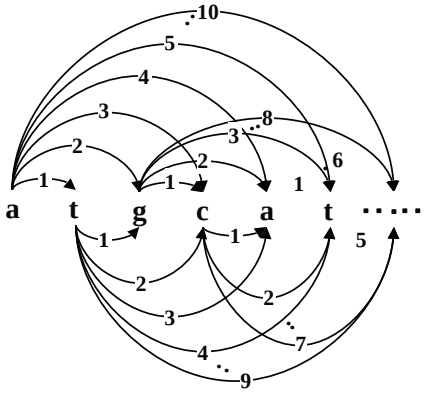

Figure 10. Conversion of DNA Sequence Data to a graph

A decision tree was constructed in either of the following two ways: 1) apply chunking n_r times only at the root node and only once at other nodes of a decision tree, 2) apply chunking n_e times at every node of a decision tree. Note that n_r and n_e are defined as the maximum depth in Figure 3. Thus, there is more chunking taking place during the search when the beam width is larger for a fixed value of n_r and n_e. The pair (subgraph) that is selected for each node of the decision tree is the one which maximizes the information gain among all the pairs that are enumerated. Pessimistic pruning described in subsection 3.4 was conducted by setting the confidence level to 25% after the decision tree is constructed. The beam width b is another parameter to be optimized. The parameters in the application of DT-GBI are summarized in Table 1.

In order to see how these parameters affect the performance of the decision tree constructed by DT-GBI and optimize their values, the whole dataset was divided into training, validation and test data. The final prediction error rate was evaluated by the average of 10 runs of 10 fold cross-validation (a total of 100 decision trees) using the optimized parameters. For one randomly chosen 9 folds data (90% of all data) of the 10-fold cross-validation of each run, we performed 9-fold cross-validation using one fold as validation data. Thus, we constructed a total of 90 decision trees for one set of parameter values and took the average for the error estimation. Intuitively the larger n_r, n_e and b, the better the results are because the search space increases. Further, n_e should give better performance than n_r. All the results indicated that the errors level off at some value of these parameters. Thus the optimal value for each parameter is the smallest value at which the error becomes minimum. All the experiments in this paper were conducted on the same PC (CPU: Athlon XP2100+(1.73GHz), Memory: 3GB, OS: RedHat Linux 8.03).

[3]We do not have domain knowledge about the interaction and there is nothing to justify this. We could have connected all the nucleotides and constructed a complete graph.

Table 1. parameters in the application of DT-GBI

n_r	apply chunking n_r times at the root node and only once at the other nodes of a decision tree
n_e	apply chunking n_e times at every node of a decision tree
b	beam width

The first experiment focused on the effect of the number of chunking at each node of a decision tree and thus b was set to 1. The parameter n_r and n_e were changed from 1 to 10 in both 1) and 2). Figure 11 shows the result of experiments. In this figure the dotted line indicates the error rate for 1) and the solid line for 2). The best error rate was 8.62% when n_r = 5 for 1) and 7.02% when n_e = 4 for 2). As mentioned above, the decrease of error rate levels off when the number of chunking increases for both 1) and 2). The result reveals that there is an appropriate size for an optimal subgraph at each node and n_r and n_e must be large enough to find subgraphs of that size. Further increasing n_r and n_e would not affect the result.

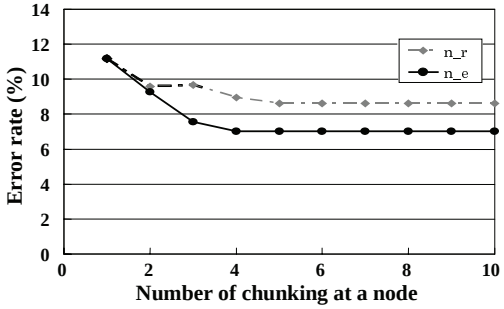

Figure 11. Effect of chunking (b=1)

The second experiment focused on the effect of beam width b, changing its value from 1 to 15. The number of chunking was fixed at the best number which was determined by the first experiment in Figure 11, namely, n_r = 5 for 1) and n_e = 4 for 2). The result is summarized in Figure 12. The best error rate was 4.15% when b = 12 for 1) (n_r = 5) and 3.83% when b = 10 for 2) (n_e = 4). Note that the error rate does not necessarily decrease monotonically as the beam width increases. This is because the search space is not necessarily monotonic although it certainly increases as the beam width increases. However, we confirmed that the error rate for the training data always monotonically decreases as the beam width increases. These properties hold for all the experiments in section 5, too.

Setting these parameters to the above best values for the validation data, the true prediction error is estimated by taking the average of 10 runs of 10-fold cross-validation using all the data. The final error rate is 4.06% for 1) n_r=5, b=12, and 3.77% for 2)n_e=4, b=10. Figure 13 and Figure 14 shows examples of the decision trees. Each of them is taken from the 10 decision trees of the best run of the 10 runs of 10-fold cross-validation. The numbers for n (non-promoter)and p (promoter) in the nodes show how the data are split when all

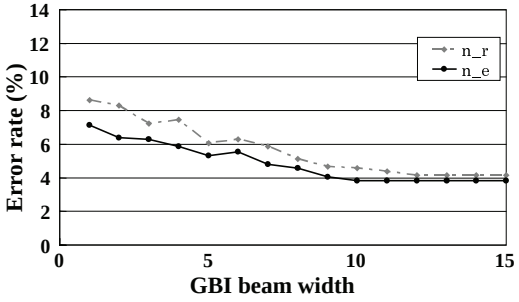

Figure 12. Effect of beam width (n_r=5, n_e=4)

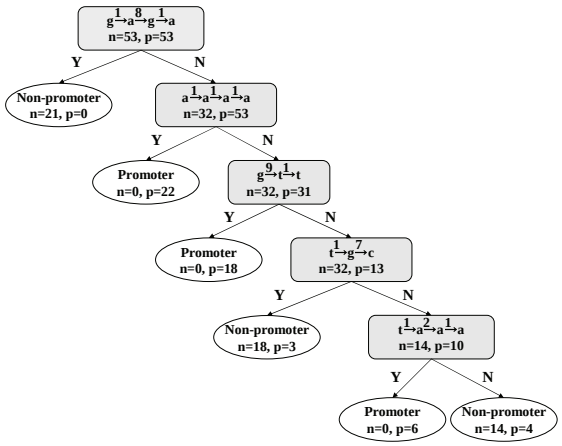

Figure 13. One of the ten trees from the best run (n_r = 5, b = 12)

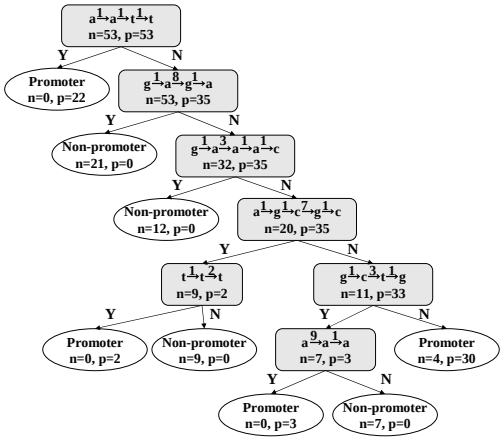

Figure 14. One of the ten trees from the best run (n_e = 4, b = 10)

Table 2. Contingency table with the number of instances (promoter vs. non-promoter)

Actual Class	Predicted Class					
	deci. tree in Fig. 13 (n_r)		deci. tree in Fig. 14 (n_e)		Overall (n_e)	
	promoter	non-promoter	promoter	non-promoter	promoter	non-promoter
promoter	5	1	5	1	506	24
non-promoter	1	5	0	6	16	514

106 instances are fed in the tree. The contingency tables of these decision trees for test data in cross validation as well as the overall contingency table (n_e only), which is calculated by summing up the contingency tables for the 100 decision trees, are shown Table 2.

The computation time depends on the values of the parameters as well as the data characteristics (graph size and connectivity). It is almost linear to the parameter values and roughly quadratic to the graph size. The average time for a single run to construct a decision tree is 99 sec. for n_e=4 and b=10. The time for classification is 0.75 sec. and is negligibly small because of the reason described in Subsection 3.5.

The result reported in [26] is 3.8% (they also used 10-fold cross validation) which is obtained by the M-of-N expression rules extracted from KBANN (Knowledge Based Artificial Neural Network). The obtained M-of-N rules are complicated and not easy to interpret. Since KBANN uses domain knowledge to configure the initial artificial neural network, it is worth mentioning that DT-GBI that uses very little domain knowledge induced a decision tree with comparable predictive accuracy. Comparing the decision trees in Figures 13 and 14, the trees are not stable. Both gives a similar predictive accuracy but the patterns in the test nodes are not the same. According to [26], there are many pieces of domain knowledge and the rule conditions are expressed by the various combinations of these pieces. Among

these many pieces of knowledge, the pattern (a → a → a → a) in the second node in Figure 13 and the one (a → a → t → t) in the root node in Figure 14 match their domain knowledge, but the others do not match. We have assumed that two nucleotides that are apart more than 10 nodes are not directly correlated. Thus, the extracted patterns have no direct edges longer than 9. It is interesting to note that the first node in Figure 13 relates two pairs (g → a) that are 7 nodes apart as a discriminatory pattern. Indeed, all the sequences having this pattern are concluded to be non-promoter from the data. In theory it is not self-evident from the constructed tree whether a substring (chunk) in a test node includes a previously learned chunk or constructed by the chunking at this stage, but in practice the former holds for most of the cases. At each node, the number of actual chunking is maximum n_e and the pair that gives the largest information gain is selected. From the result it is not clear whether the DT-GBI can extract the domain knowledge or not. The data size is too small to make any strong claims.

[15, 16] report another approach to construct a decision tree for the promoter dataset. B-GBI is used as a preprocessor to extract patterns (subgraphs) and these patterns are passed to C4.5 as attributes to construct a decision tree. In [16] they varied the number of folds and found that the best error rate decreases as the number of folds becomes larger, resulting in the minimum error rate of 2.8% in Leave One Out. The corresponding best error rate for 10 fold cross validation is 6.3% using the patterns extracted by B-GBI with b = 2. In contrast the best prediction error rate of DT-GBI is 3.77% when n_e = 4, b = 10. This is much better than 6.3% above and comparable to 3.8 % obtained by KBANN using M-of-N expression [26].

5. Application to Hepatitis Dataset

Viral hepatitis is a very critical illness. If it is left without undergoing a suitable medical treatment, a patient may suffer from cirrhosis and fatal hepatoma. The progress speed of the condition is slow and subjective symptoms are not noticed easily, hence, in many cases, it has already become very severe when they are noticed. Although periodic inspection and proper treatment are important in order to prevent this situation, there are problems of expensive cost and physical burden on a patient. Although there is an alternative much cheaper method of inspection (blood test and urinalysis), the amount of data becomes enormous since the progress speed of condition is slow.

The hepatitis dataset provided by Chiba University Hospital contains long time-series data (from 1982 to 2001) on laboratory examination of 771 patients of hepatitis B and C. The data can be split into two categories. The first category includes administrative information such as patient's information (age and date of birth), pathological classification of the disease, date of biopsy and its result, and the effectiveness of interferon therapy. The second category includes a temporal record of blood test and urinalysis. It contains the result of 983 types of both in- and out-hospital examinations.

First, effect of parameters n_r, n_e and b on the prediction accuracy was investigated for each experiment using validation dataset as in the promoter case in section 4, but only for the first run of the 10 runs of 10-fold cross-validation. It was confirmed that for all the experiments chunking parameters n_r and n_e can be set to 20, but the beam width b was found experiment dependent, and the narrowest beam width that brings to the lowest error rate was used. The pessimistic pruning was used with the same confidence level (25%) as in the promoter case. The prediction accuracy was evaluated by taking the average of 10 runs of 10-fold cross-validation. Thus, 100 decision trees were constructed in total.

In the following subsections, both the average error rate and examples of decision trees are shown

in each experiment together with examples of extracted patterns. Two decision trees were selected out of the 100 decision trees in each experiment: one from the 10 trees constructed in the best run with the lowest error rate of the 10 runs of 10-fold cross validation, and the other from the 10 trees in the worst run with the highest error rate. In addition, the contingency tables of the selected decision trees for test data in cross validation are shown as well as the overall contingency table, which is calculated by summing up the contingency tables for the 100 decision trees in each experiment.

5.1. Data Preprocessing

5.1.1. Cleansing and Conversion to Table

In numeric attributes, letters and symbols such as H, L, or + are deleted. Values in nominal attributes are left as they are. When converting the given data into an attribute-value table, both a patient ID (MID) and a date of inspection are used as search keys and an inspection item is defined as an attribute. Since all patients do not necessarily take all inspections, there are many missing values after this data conversion. No attempt is made to estimate these values and those missing values are not represented in graph-structured data in the following experiments. In future, it is necessary to consider the estimation of missing values.

5.1.2. Averaging and Discretization

This step is necessary due to the following two reasons: 1) obvious change in inspection across successive visits may not be found because the progress of hepatitis is slow, and 2) the date of visit is not synchronized across different patients. In this step, the numeric attributes are averaged and the most frequent value is used for nominal attributes over some interval. Further, for some inspections (GOT, GPT, TTT, and ZTT), standard deviations are calculated over six months and added as new attributes.

When we represent an inspection result as a node label, the number of node labels become too large and this lowers the efficiency of DT-GBI because frequent and discriminative patterns cannot be extracted properly. Therefore, we reduced the number of node labels by discretizing attribute values. For general numerical values, the normal ranges are specified and values are discretized into three (“L” for low, “N” for normal, and “H” for high). Based on the range and the advice from the domain experts, the standard deviations of GOT and GPT are discretized into five (“1” for the smallest deviation, “2”,“3”,“4”,“5” for the largest deviation), while the standard deviations of TTT and ZTT are discretized into three (“1” for the smallest deviation,“2”,“3” for the largest deviation). Figure 15 illustrates the mentioned four steps of data conversion.

5.1.3. Limiting Data Range

In our analysis it is assumed that each patient has one class label, which is determined at some inspection date. The longer the interval between the date when the class label is determined and the date of blood inspection is, the less reliable the correlation between them is. We consider that the pathological conditions remain the same for some duration and conduct the analysis for the data which lies in the range. Furthermore, although the original dataset contains hundreds of examinations, feature selection was conducted with the expert to reduce the number of attributes. The duration and attributes used depend on the objective of the analysis and are described in the following results.

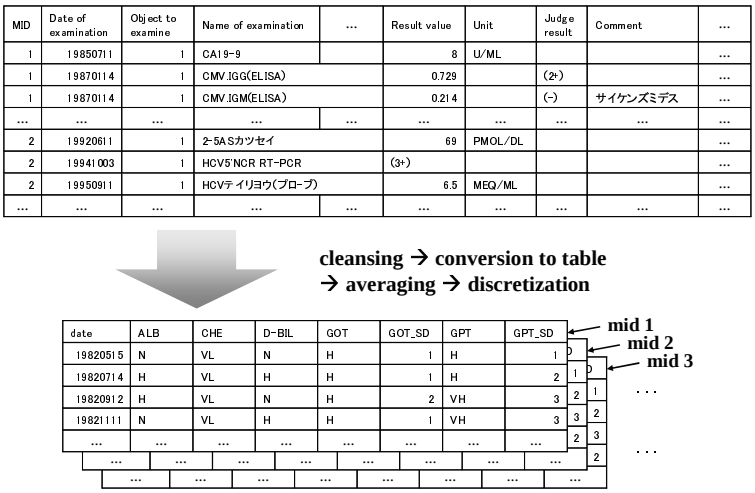

Figure 15. Averaging and discretization of inspection data

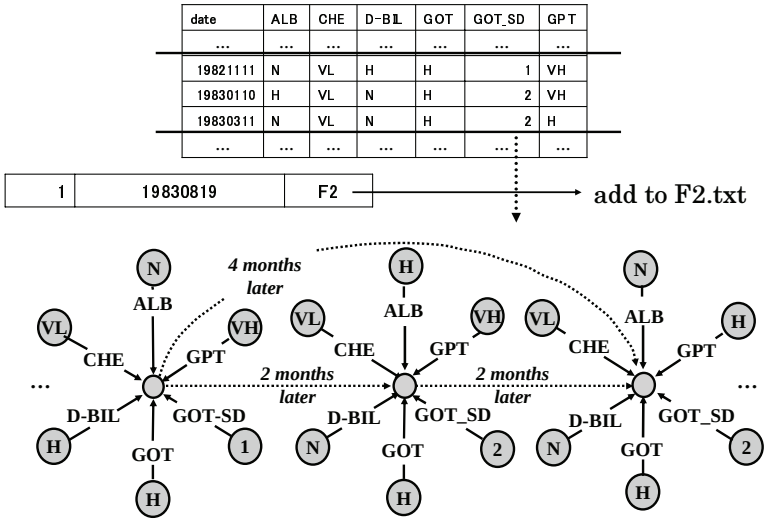

Figure 16. An example of converted graph-structured data

5.1.4. Conversion to Graph-Structured Data

When analyzing data by DT-GBI, it is necessary to convert the data to graph structure. One patient record is mapped into one directed graph. Assumption is made that there is no direct correlation between two sets of pathological tests that are more than a predefined interval (here, two years) apart. Hence, direct time correlation is considered only within this interval. Figure 16 shows an example of converted graph-structured data. In this figure, a star-shaped subgraph represents values of a set of pathological examination in a two-month interval. The center node of the subgraph is a hypothetical node for the interval. An edge pointing to a hypothetical node represents an examination. The node connected to the edge represents the value (preprocessed result) of the examination. The edge linking two hypothetical

Table 3. Size of graphs (fibrosis stage)

stage of fibrosis	F0	F1	F2	F3	F4	Total
No. of graphs	4	125	53	37	43	262
Avg. No. of nodes	303	304	308	293	300	303
Max. No. of nodes	349	441	420	414	429	441
Min. No. of nodes	254	152	184	182	162	152

nodes represents time difference.

5.1.5. Class Label Setting

In the first and second experiments, we set the result (progress of fibrosis) of the first biopsy as class. In the third experiment, we set the subtype (B or C) as class. In the fourth experiment, the effectiveness of interferon therapy was used as class label.

5.2. Classifying Patients with Fibrosis Stages

For the analysis in subsection 5.2 and subsection 5.3, the average was taken for two-month interval. As for the duration of data considered, data in the range from 500 days before to 500 days after the first biopsy were extracted for the analysis of the biopsy result. When biopsy was operated for several times on the same patient, the treatment (*e.g.,* interferon therapy) after a biopsy may influence the result of blood inspection and lower the reliability of data. Thus, the date of first biopsy and the result of each patient are searched from the biopsy data file. In case that the result of the second biopsy or after differs from the result of the first one, the result from the first biopsy is defined as the class of that patient for the entire 1,000-day time-series.

Fibrosis stages are categorized into five stages: F0 (normal), F1, F2, F3, and F4 (severe = cirrhosis). We constructed decision trees which distinguish the patients at F4 stage from the patients at the other stages. In the following two experiments, we used 32 attributes. They are: ALB, CHE, D-BIL, GOT, GOT_SD, GPT, GPT_SD, HBC-AB, HBE-AB, HBE-AG, HBS-AB, HBS-AG, HCT, HCV-AB, HCV-RNA, HGB, I-BIL, ICG-15, MCH, MCHC, MCV, PLT, PT, RBC, T-BIL, T-CHO, TP, TTT, TTT_SD, WBC, ZTT, and ZTT_SD. Table 3 shows the size of graphs after the data conversion.

As shown in Table 3, the number of instances (graphs) in cirrhosis (F4) stage is 43 while the number of instances (graphs) in non-cirrhosis stages (F0 + F1 + F2 + F3) is 219. Imbalance in the number of instances may cause a biased decision tree. In order to relax this problem, we limited the number of instances to the 2:3 (cirrhosis:non-cirrhosis) ratio which is the same as in [29]. Thus, we used all instances from F4 stage for cirrhosis class (represented as LC) and select 65 instances from the other stages for non-cirrhosis class(represented as non-LC), 108 instances in all. How we selected these 108 instances is described below.

Table 4. Average error rate (%) (fibrosis stage)

run of	F4 vs.{F0,F1}		F4 vs.{F2,F3}	
10 CV	n_r=20	n_e=20	n_r=20	n_e=20
1	14.81	11.11	27.78	25.00
2	13.89	11.11	26.85	25.93
3	15.74	12.03	25.00	19.44
4	16.67	15.74	27.78	26.68
5	16.67	12.96	25.00	22.22
6	15.74	14.81	23.15	21.30
7	12.96	9.26	29.63	25.93
8	17.59	15.74	25.93	22.22
9	12.96	11.11	27.78	21.30
10	12.96	11.1	27.78	25.00
average	15.00	12.50	26.67	23.52
Standard Deviation	1.65	2.12	1.80	2.39

5.2.1. Experiment 1: F4 stage vs {F0+F1} stages

All 4 instances in F0 and 61 instances in F1 stage were used for non-cirrhosis class in this experiment. The beam width b was set to 15. The overall result is summarized in the left half of Table 4. The average error rate was 15.00% for n_r=20 and 12.50% for n_e=20. As can be predicted, the error is always smaller for n_e than for n_r if their values are the same. Figures 17 and 18 show one of the decision trees each from the best run with the lowest error rate (run 7) and from the worst run with the highest error rate (run 8) for n_e=20, respectively. Comparing these two decision trees, we notice that three patterns that appeared at the upper levels of each tree are identical. The contingency tables for these decision trees and the overall one are shown in Table 5. It is important not to diagnose non-LC patients as LC patients to prevent unnecessary treatment, but it is more important to classify LC patient correctly because F4 (cirrhosis) stage might lead to hepatoma. Table 5 reveals that although the number of misclassified instances for LC (F4) and non-LC ({F0+F1}) are almost the same, the error rate for LC is larger than that for non-LC because the class distribution of LC and non-LC is imbalanced (note that the number of instances is 65 for LC and 108 for non-LC). The results are not favorable in this regards. Predicting minority class is more difficult than predicting majority class. This tendency holds for the remaining experiments. By regarding LC (F4) as positive and non-LC ({F0+F1}) as negative, decision trees constructed by DT-GBI tended to have more false negative than false positive.

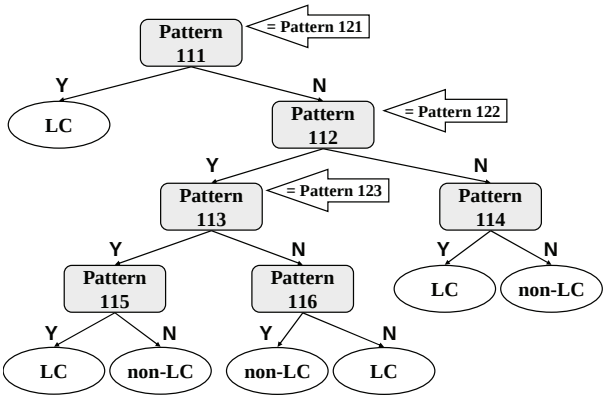

Figure 17. One of the ten trees from the best run in exp.1 (n_e=20)

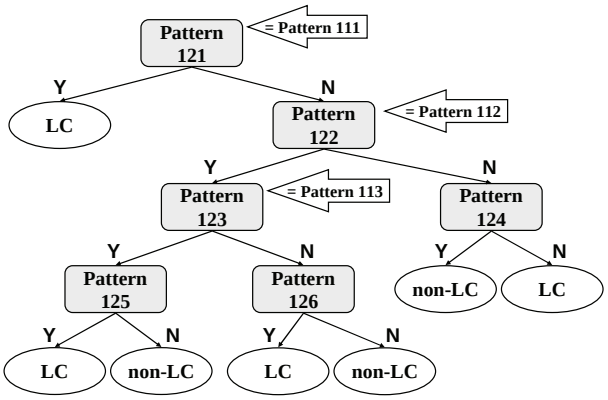

Figure 18. One of the ten trees from the worst cycle in exp.1 (n_e=20)

Table 5. Contingency table with the number of instances (F4 vs. {F0+F1})

Actual Class	Predicted Class					
	decision tree in Figure 17		decision tree in Figure 18		Overall	
	LC	non-LC	LC	non-LC	LC	non-LC
LC	3	1	4	1	364	66
non-LC	1	5	4	3	69	581

LC = F4, non-LC = {F0+F1}

5.2.2. Experiment 2: F4 stage vs {F3+F2} stages

In this experiment, we used all instances in F3 and 28 instances in F2 stage for non-cirrhosis class. The beam width b was set to 14 in experiment 2. The overall result is summarized in the right-hand side of Table 4. The average error rate was 26.67% for n_r=20 and 23.52% for n_e=20. Figures 21 and 22 show

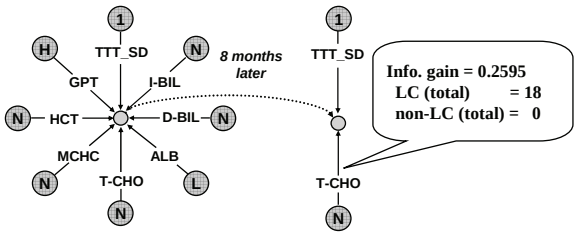

Figure 19. Pattern 111 = Pattern 121 (if exist then LC)

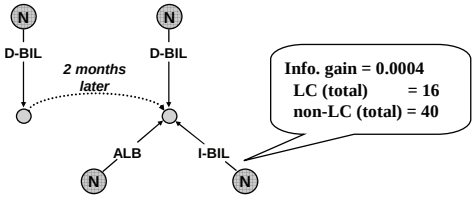

Figure 20. Pattern 112 = Pattern 122

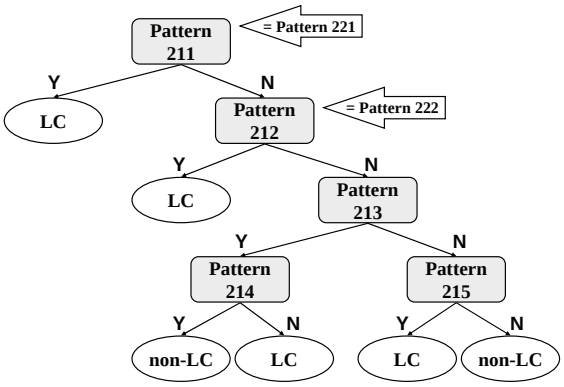

Figure 21. One of the ten trees from the best run in exp.2 (n_e=20)

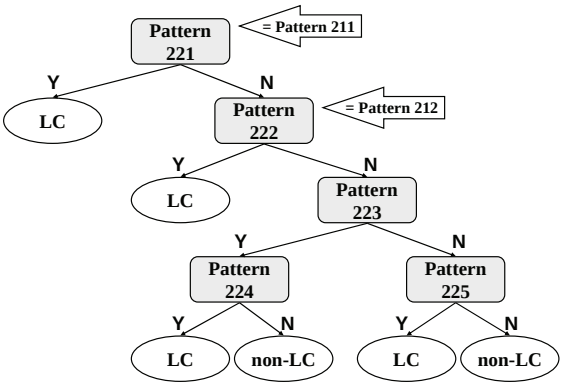

Figure 22. One of the ten trees from the worst run in exp.2 (n_e=20)

Table 6. Contingency table with the number of instances (F4 vs. {F3+F2})

Actual Class	Predicted Class					
	decision tree in Figure 21		decision tree in Figure 22		Overall	
	LC	non-LC	LC	non-LC	LC	non-LC
LC	3	1	2	2	282	148
non-LC	2	5	3	4	106	544

LC = F4, non-LC = {F3+F2}

examples of decision trees each from the best run with the lowest error rate (run 3) and the worst run with the highest error rate (run 4) for n_e=20, respectively. Comparing these two decision trees, we notice that two patterns that appeared at the upper levels of each tree are identical. The contingency tables for these decision trees and the overall one are shown in Table 6. Since the overall error rate in experiment 2 was larger than that of experiment 1, the number of misclassified instances increased. By regarding LC (F4) as positive and non-LC ({F3+F2}) as negative as described in experiment 1, decision trees constructed by DT-GBI tended to have more false negative than false positive for predicting the stage of fibrosis. This tendency was more prominent in experiment 2 compared with experiment 1.

5.2.3. Discussion for the analysis of Fibrosis Stages

The average prediction error rate in the first experiment is better than that in the second experiment, as the difference in characteristics between data in F4 stage and data in {F0+F1} stages is intuitively larger than that between data in F4 stage and data in {F3+F2}. The averaged error rate of 12.50% in experiment 1 is fairly comparable to that of 11.8% obtained by the decision tree reported in [29]. In their analysis, a node in the decision tree is a time series data of an inspection of a patient. Distance between two time series data for the same inspection with different patients is calculated by the dynamic time warp method. Two data are similar if their distance is below a threshold. The problem is reduced to

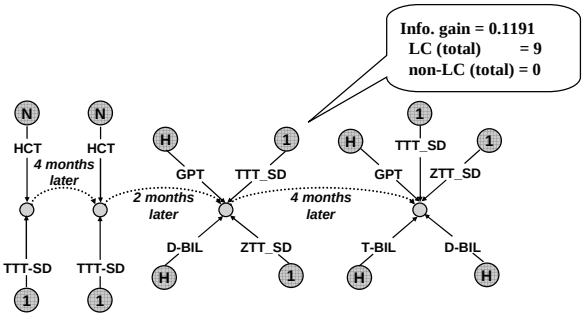

Figure 23. Pattern 211 = Pattern 221 (if exist then LC)

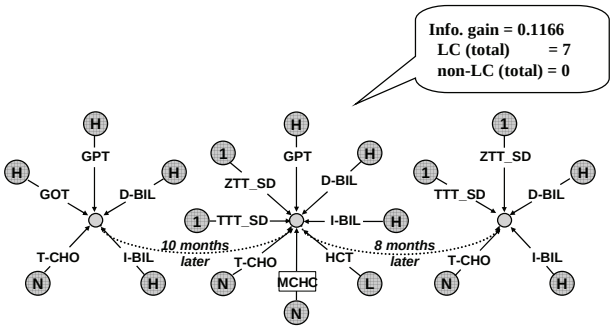

Figure 24. Pattern 212 = Pattern 222

date	ALB	D-BIL	GPT	HCT	I-BIL	MCHC	T-CHO	TTT_SD	···
19930517	L	N	H	N	N	N	N	1	···
19930716	L	L	H	N	N	N	N		···
19930914	L	L	H	N	N	N	N	1	···
19931113	L	N	H	N	N	N	N		···
19940112	L	L	H	N	N	N	N	1	···
19940313	L	N	N	N	N	N	N	1	···
19940512	L	N	H	N	N	N	N	1	···
19940711	L	N	H	N	N	N	N	1	···
19940909	L	L	H	N	N	N	N	1	···
19941108	L	N	N	N	N	N	N	1	···
19950107	L	N	N	L	N	N	N	1	···
19950308	L	N	N	N	N	N	N	1	···
19950507	L	N	H	N	N	N	N	1	···
19950706	L	N	N	L	N	N	N	1	···
19950904	L	L	N	L	N	L	N	1	···
19951103	L	L	N	N	N	N	N	1	···

Figure 25. Data of No.203 patient

finding which inspection data of which patient with what value for threshold to use as a test at each node. While their method is unable to treat inter-correlation among inspections it can treat intra-correlation of individual inspection with respect to time without discretization.

Patterns shown in Figures 19, 20, 23 and 24 are sufficiently discriminative since all of them are used at the nodes in the upper part of all decision trees. The certainty of these patterns is ensured as, for almost all patients, they appear after the biopsy. These patterns include inspection items and their values that are typical of cirrhosis. These patterns may appear only once or several times in one patient. Figure 25 shows the data of a patient for whom pattern 111 exists. As we made no attempt to estimate missing values, the pattern was not counted even if the value of only one attribute is missing. At data in the Figure 25, pattern 111 would have been counted four if the value of TTT_SD in the fourth line had been "1" instead of missing.

The computation time has increased considerably compared with the promoter experiment because the graph size is larger and the time is roughly quadratic to the graph size. The average time for a single run to construct a decision tree is 2,698 sec. for n_e=20 and b=15. The time for classification is 1.04 sec.

Table 7. Size of graphs (hepatitis type)

hepatitis type	Type B	Type C	Total
No. of graphs	77	185	262
Avg. No. of nodes	238	286	272
Max. No. of nodes	375	377	377
Min. No. of nodes	150	167	150

Table 8. Average error rates (%) (hepatitis type)

run of 10 CV	Type B vs. Type C	
	n_r=20	n_e=20
1	21.76	18.65
2	21.24	19.69
3	21.24	19.17
4	23.32	20.73
5	25.39	22.80
6	25.39	23.32
7	22.28	18.65
8	24.87	19.17
9	22.80	19.69
10	23.83	21.24
Average	23.21	20.31
Standard Deviation	1.53	1.57

which is still kept small because of the reason described in Subsection 3.5.

5.3. Classifying Patients with Types (B or C)

There are two types of hepatitis recorded in the dataset: B and C. We constructed decision trees which distinguish between patients of type B and type C. As in subsection 5.2, the examination records from 500 days before to 500 days after the first biopsy were used and average was taken for two-month interval. Among the 32 attributes used in subsection 5.2, the attributes of antigen and antibody (HBC-AB, HBE-AB, HBE-AG, HBS-AB, HBS-AG, HCV-AB, HCV-RNA) were not included as they obviously indicate the type of hepatitis. Thus, we used the following 25 attributes: ALB, CHE, D-BIL, GOT, GOT_SD, GPT, GPT_SD, HCT, HGB, I-BIL, ICG-15, MCH, MCHC, MCV, PLT, PT, RBC, T-BIL, T-CHO, TP, TTT, TTT_SD, WBC, ZTT, and ZTT_SD. Table 7 shows the size of graphs after the data conversion. To keep the number of instances at 2:3 ratio [29], we used all of 77 instances in type B as "Type B" class and 116 instances in type C as "Type C" class. Hence, there are 193 instances in all. The beam width b was set to 5 in this experiment (Experiment 3).

The overall result is summarized in Table 8. The average error rate was 23.21% for n_r=20 and 20.31% for n_e=20. The average computation time for constructing a single tree is 1,463 sec. for $n_e = 20$ and $b = 5$. The reduction from experiments 1 and 2 comes from the reduction of b. The error result is better than the approach in [10]. Their unpublished recent result is 26.2%[4]. In their analysis notion of

[4]Personal communication

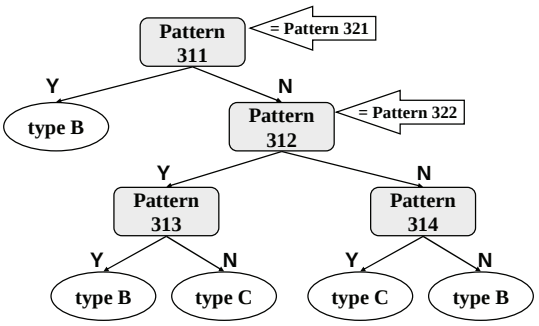

Figure 26. One of the ten trees from the best run in exp.3 (n_e=20)

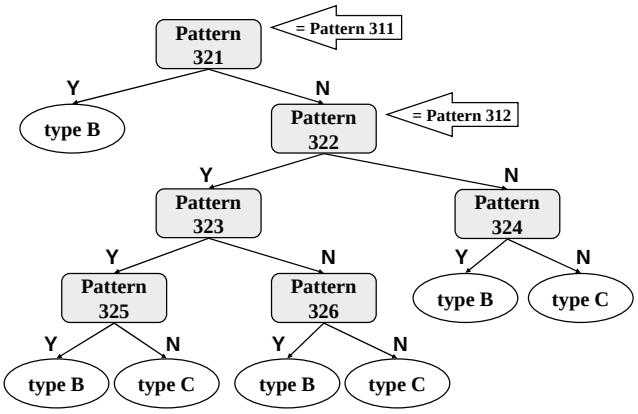

Figure 27. One of the ten trees from the worst run in exp.3 (n_e=20)

Table 9. Contingency table with the number of instances (hepatitis type)

Actual Class	Predicted Class					
	decision tree in Figure 26		decision tree in Figure 27		Overall	
	Type B	Type C	Type B	Type C	Type B	Type C
Type B	6	2	7	1	559	211
Type C	0	11	4	7	181	979

episode is introduced and the time series data is temporally abstracted within an episode using the state and the trend symbols. Thus, the abstracted data in the form of attribute-value representation is amenable to a standard rule learner. The problem is that the length of episode must be carefully determined so that the global behavior is captured by a small set of predefined concepts. Figure 26 and Figure 27 show samples of decision trees from the best run with the lowest error rate (run 1) and the worst run with the highest error rate (run 6) for n_e = 20, respectively. Comparing these two decision trees, two

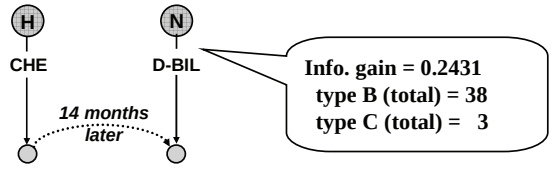

Figure 28. Pattern 311 = Pattern 321 (if exist then Type B)

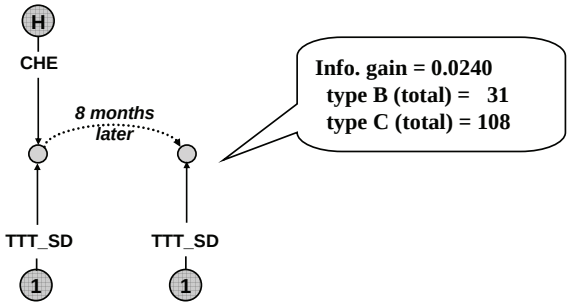

Figure 29. Pattern 312 = Pattern 322

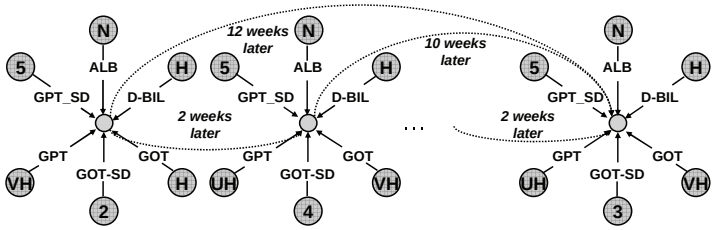

Figure 30. An example of graph-structured data for the analysis of interferon therapy

patterns (shown in Figures 28 and 29) were identical and used at the upper level nodes. There patterns also appeared at almost all the decision trees and thus are considered sufficiently discriminative. The contingency tables for these decision trees and the overall one are shown in Table 9. Since the hepatitis C tends to become chronic and can eventually lead to hepatoma, it is more valuable to classify the patient of type C correctly. The results are favorable in this regards because the minority class is type B in this experiment. Thus, by regarding type B as negative and type C as positive, decision trees constructed by DT-GBI tended to have more false positive than false negative for predicting the hepatitis type.

5.4. Classifying Interferon Therapy

An interferon is a medicine to get rid of the hepatitis virus and it is said that the smaller the amount of virus is, the more effective interferon therapy is. Unfortunately, the dataset provided by Chiba University Hospital does not contain the examination record for the amount of virus since it is expensive. However, it is believed that experts (medical doctors) decide when to administer an interferon by estimating the amount of virus from the results of other pathological examinations. Response to interferon therapy was judged by a medical doctor for each patient, which was used as the class label for interferon therapy. The class labels specified by the doctor for interferon therapy are summarized in Table 10. Note that the following experiments (Experiment 4) were conducted for the patients with label R (38 patients) and N (56 patients). Medical records for other patients were not used.

To analyze the effectiveness of interferon therapy, we hypothesized that the amount of virus in a patient was almost stable for a certain duration just before the interferon injection in the dataset. Data in the range of 90 days to 1 day before the administration of interferon were extracted for each patient and average was taken for two-week interval. Furthermore, we hypothesized that each pathological condition in the extracted data could directly affect the pathological condition just before the administration. To represent this dependency, each subgraph was directly linked to the last subgraph in each patient. An example of converted graph-structured data is shown in Figure 30.

As in subsection 5.2 and subsection 5.3, feature selection was conducted to reduce the number of attributes. Since the objective of this analysis is to predict the effectiveness of interferon therapy without referring to the amount of virus, the attributes of antigen and antibody (HBC-AB, HBE-AB, HBE-AG, HBS-AB, HBS-AG, HCV-AB, HCV-RNA) were not included. Thus, as in subsection 5.3 we used the following 25 attributes: ALB, CHE, D-BIL, GOT, GOT_SD, GPT, GPT_SD, HCT, HGB, I-BIL, ICG-15, MCH, MCHC, MCV, PLT, PT, RBC, T-BIL, T-CHO, TP, TTT, TTT_SD, WBC, ZTT, and ZTT_SD. Table 11 shows the size of graphs after the data conversion. The beam width b was set to 3 in experiment 4.

Table 10. class label for interferon therapy

label	
R	virus disappeared (Response)
N	virus existed (Non-response)
?	no clue for virus activity
R?	R (not fully confirmed)
N?	N (not fully confirmed)
??	missing

Table 11. Size of graphs (interferon therapy)

effectiveness of interferon therapy	R	N	Total
No. of graphs	38	56	94
Avg. No. of nodes	77	74	75
Max. No. of nodes	123	121	123
Min. No. of nodes	41	33	33

Table 12. Average error rate (%) (interferon therapy)

run of 10 CV	n_e=20
1	18.75
2	23.96
3	20.83
4	20.83
5	21.88
6	22.92
7	26.04
8	23.96
9	23.96
10	22.92
Average	22.60
Standard Deviation	1.90

The results are summarized in Table 12 and the overall average error rate was 22.60%. In this experiment we did not run the cases for n_r=20 because it is known that n_r gives no better results from the previous experiments. The average computation time for constructing a single tree is 110 sec. Figures 31 and 32 show examples of decision trees each from the best run with the lowest error rate (run 1) and the worst run with the highest error rate (run 7) respectively. Patterns at the upper nodes in these trees are shown in Figures 33, 34 and 35. Although the structure of decision tree in Figure 31 is simple, its prediction accuracy was actually good (error rate=10%). Note that the error for each run in Table 12 is the average error of 10 decision trees of 10-fold cross-validation, and Figure 31 corresponds to the one with the lowest error among the 10. Furthermore, since the pattern shown in Figure 33 was used at the root node of many decision trees, it is considered as sufficiently discriminative for classifying patients for whom interferon therapy was effective (with class label R).

The contingency tables for these decision trees and the overall one are shown in Table 13. By regarding the class label R (Response) as positive and the class label N (Non-response) as negative, the decision trees constructed by DT-GBI tended to have more false negative for predicting the effectiveness of interferon therapy. As in experiments 1 and 2, minority class is more difficult to predict. The patients with class label N are mostly classified correctly as "N", which will contribute to reducing the fruitless interferon therapy of patients, but some of the patients with class label R are also classified as "N", which may lead to miss the opportunity of curing patients with interferon therapy.

Unfortunately, only a few patterns contain time interval edges in the constructed decision trees, so we were unable to investigate how the change or stability of blood test will affect the effectiveness of interferon therapy. Figure 36 is an example of a decision tree with time interval edges for the analysis of

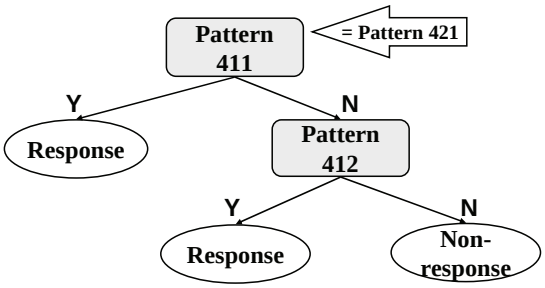

Figure 31. One of the ten trees from the best run

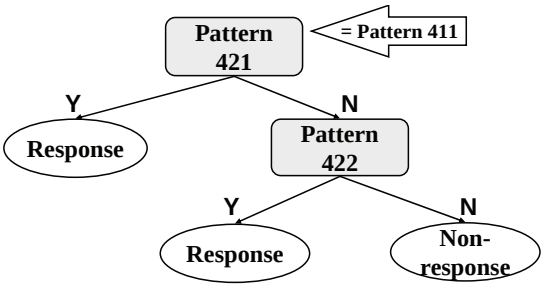

Figure 32. One of the ten trees from the worst run

Table 13. Contingency table with the number of instances (interferon therapy)

Actual Class	Predicted Class					
	decision tree in Figure31		decision tree in Figure32		Overall	
	R	N	R	N	R	N
R	3	1	2	2	250	130
N	0	6	4	2	83	477

interferon therapy and some patterns in this tree are shown in Figures 37 and 38.

6. Conclusion

This paper has proposed a method called DT-GBI, which constructs a classifier (decision tree) for graph-structured data by GBI. Substructures useful for classification task are constructed on the fly by applying repeated chunking in GBI during the construction process of a decision tree. A beam search is very effective in increasing the predictive accuracy. DT-GBI was evaluated against a DNA dataset from UCI repository and applied to analyze a real-world hepatitis dataset as a part of evidence-based medicine. The results of the promoter dataset show that DT-GBI is comparable to other method that uses domain knowledge in modeling the classifier. Four experiments were conducted on the hepatitis dataset and both

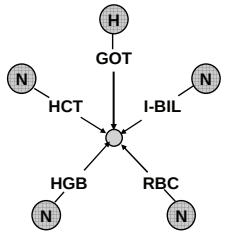

Figure 33. Pattern 411 = Pattern 421 (if exist then R)

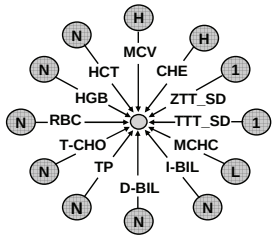

Figure 34. Pattern 412 (if exist then R, if not exist then N)

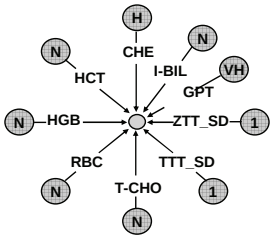

Figure 35. Pattern 422 (if exist then R, if not exist then N)

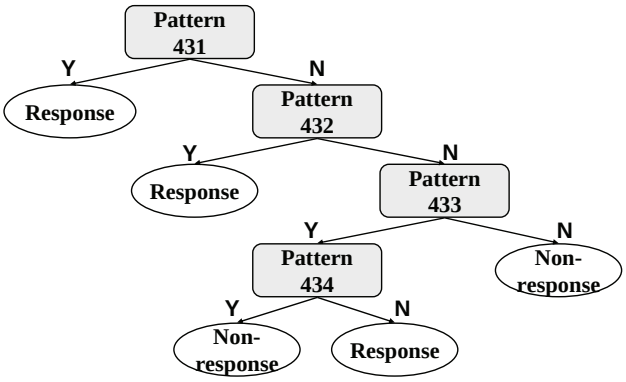

Figure 36. Example of decision tree with time interval edge in exp.4

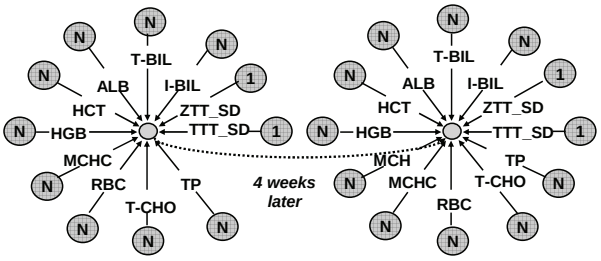

Figure 37. Pattern 431 (if exist then R)

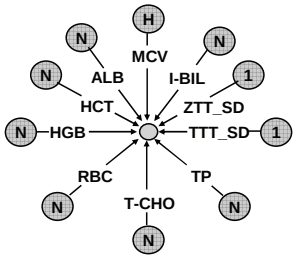

Figure 38. Pattern 432 (if exist then R, if not exist then N)

constructed decision trees and their predictive accuracies were reported. DT-GBI was able to extract discriminative patterns that reflect both intra-correlation of individual inspection and inter-correlation among inspections at different time points and use them as attributes for classification. Some of the extracted patterns match medical experts' experience. The obtained prediction error rate results are thought satisfactory by the domain experts in spite of the various noises that are embedded in the data. We thus believe that DT-GBI is a useful tool for practicing evidence-based medicine.

General finding is that the decision trees constructed by DT-GBI tended to misclassify more instances with minority class than with the majority class. Incorporating cost-sensitive learning might be effective to take into account the imbalance in class distribution and different penalties for misclassification error.

Immediate future work includes to incorporate more sophisticated method for determining the number of cycles to call GBI at each node to improve prediction accuracy. Utilizing the rate of change of information gain by successive chunking is a possible way to automatically determine the number. Effectiveness of DT-GBI against the hepatitis data set with another way of preparing data should be examined, *e.g.,* estimating missing values, randomly selecting instances from non-cirrhosis class both for training and testing, etc. The validity of extracted patterns is now being carefully evaluated and discussed by the domain experts (medical doctors). It is also necessary to experimentally compare DT-GBI with other methods, especially with ILP based ones such as TILDE and S-CART.

Acknowledgment

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. The authors are grateful to the Chiba University Hospital for providing the hepatitis dataset.

References

[1] Blake, C. L., Keogh, E., Merz, C.: UCI Repository of Machine Leaning Database, 1998, Http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[2] Blockeel, H., De Raedt, L.: Top-down induction of first-order logical decision tree, *Artificial Intelligence*, **101**, 1998, 285–297.

[3] Blockeel, H., Sebag, M.: Scalability and efficiency in multi-relational data mining, *ACM SIGKDD Explorations Newsletter*, **5**(1), 2003, 17–30.

[4] Breiman, L., Friedman, J. H., Olshen, R. A., Stone, C. J.: *Classification and Regression Trees*, Wadsworth & Brooks/Cole Advanced Books & Software, 1984.

[5] Breiman, L., Friedman, J. H., Olshen, R. A., Stone, C. J.: The CN2 Induction Algorithm, *Machine Learning*, **3**, 1989, 261–283.

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

[7] Džeroski, S.: Relational Data Mining Applications: An Overview, *Relational Data Mining*, 2001, 63–102.

[8] Fortin, S.: The graph isomorphism problem, 1996.

[9] Hirano, S., Tsumoto, S.: Mining Similar Temporal Patterns in Long Time-series Data and Its Application to Medicine, *Proceedings of 2002 IEEE International Conference on Data Mining (ICDM 2002)*, December 2002.

[10] Ho, T. B., Nguyen, T. D., Kawasaki, S., Le, S., Nguyen, D. D., Yokoi, H., Takabayashi, K.: Mining Hepatitis Data with Temporal Abstraction, *Proc. of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2003.

[11] Inokuchi, A., Washio, T., Motoda, H.: An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data, *Proc. of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, 2000.

[12] Inokuchi, A., Washio, T., Motoda, H.: Complete Mining of Frequent Patterns from Graphs: Mining Graph Data, *Machine Learning*, **50**(3), 2003, 321–354.

[13] Kramer, S., Windmer, G.: Inducing Classification and Regression Trees in First Order Logic, *Relational Data Mining*, 2001, 140–159.

[14] 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)*, 2000.

[15] Matsuda, T., Yoshida, T., Motoda, H., Washio, T.: Knowledge Discovery from Structured Data by Beam-wise Graph-Based Induction, *Proc. of the 7th Pacific Rim International Confernce on Artificial Intelligence (Springer Verlag LNAI2417)*, 2002.

[16] Matsuda, T., Yoshida, T., Motoda, H., Washio, T.: Mining Patterns from Structured Data by Beam-wise Graph-Based Induction, *Proc. of The Fifth International Conference on Discovery Science*, 2002.

[17] Michalski, R. S.: Learning Flexible Concepts: Fundamental Ideas and a Method Based on Two-Tiered Representaion, *In Machine Learning, An Artificial Intelligence Approiach*, **3**, 1990, 63–102.

[18] Muggleton, S.: Inductive Logic Programming: Issues, results and challenge of Learning Language in Logic, *Artificial Intelligence*, **114**, 1999, 283–296.

[19] Muggleton, S., de Raedt, L.: Inductive Logic Programming: Theory and Methods, *Journal of Logic Programming*, **19**(20), 1994, 629–679.

[20] Nédellec, C., Adé, H., Bergadano, F., Tausend, B.: Declarative bias in ILP, *Advances in Inductive Logic Programming*, 1996, 82–103.

[21] Ohsaki, M., Sato, Y., Kitaguchi, S., Yamaguchi, T.: A Rule Discovery Support System, *Project "Realization of Active Mining in the Era of Information Flood" Report*, March 2003.

[22] Page, D., Srinivasan, A.: ILP: A Short Look Back and a Longer Forward, *Journal of Machine Learning Research*, **4**, 2003, 415–430.

[23] Quinlan, J. R.: Induction of decision trees, *Machine Learning*, **1**, 1986, 81–106.

[24] Quinlan, J. R.: *C4.5:Programs For Machine Learning*, Morgan Kaufmann Publishers, 1993.

[25] Read, R. C., Corneil, D. G.: The graph isomorphism disease, *Journal of Graph Theory*, **1**, 1977, 339–363.

[26] Towell, G. G., Shavlik, J. W.: Extracting Refined Rules from Knowledge-Based Neural Networks, *Machine Learning*, **13**, 1993, 71–101.

[27] Warodom, G., Matsuda, T., Yoshida, T., Motoda, H., Washio, T.: Classifier Construction by Graph-Based Induction for Graph-Structured Data, *Proc. of the 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining (Springer Verlag LNAI2637)*, 2003.

[28] Warodom, G., Matsuda, T., Yoshida, T., Motoda, H., Washio, T.: Performance Evaluation of Decision Tree Graph-Based Induction, *Proc. of the International Conference on Discovery Science (Springer Verlag LNAI2843)*, 2003.

[29] Yamada, Y., Suzuki, E., Yokoi, H., Takabayashi, K.: Decision-tree Induction from Time-series Data Based on a Standard-example Split Test, *Proc. of the 12th International Conference on Machine Learning*, August 2003.

[30] Yoshida, K., Motoda, H.: CLIP : Concept Learning from Inference Pattern, *Journal of Artificial Intelligence*, **75**(1), 1995, 63–92.