Extension of Graph-Based Induction for General Graph Structured Data

Takashi Matsuda, Tadashi Horiuchi, Hiroshi Motoda and Takashi Washio

I.S.I.R., Osaka University, 8-1 Mihogaoka, Ibaraki, Osaka 567-0047, JAPAN

Abstract. A machine learning technique called Graph-Based Induction (*GBI*) efficiently extracts typical patterns from directed graph data by stepwise pair expansion (pairwise chunking). In this paper, we expand the capability of the Graph-Based Induction to handle not only tree structured data but also multi-inputs/outputs nodes and loop structure (including a self-loop) which cannot be treated in the conventional way. The method is verified to work as expected using artificially generated data and we evaluated experimentally the computation time of the implemented program. We, further, show the effectiveness of our approach by applying it to two kinds of the real-world data: World Wide Web browsing data and DNA sequence data.

1 Introduction

Inductive learning, which tries to find useful rules and patterns from data, has been an important area of investigation. Conventional learning methods use an attribute-value table as a data representation language and represent the relation between attribute values and classes by use of decision tree [Quinlan86] and rules [Michalski90,Clark89]. Association rules [Agrawal94] widely used in the area of data mining belong to this type of data representation. However, the attribute-value table is not suitable for representing more general and structural data. Inductive logic programming (ILP) [Muggleton89] which uses the first-order predicate logic can represent general relationship in data. ILP has a merit that domain knowledge and acquired knowledge can be utilized as background knowledge. However, its state of the art is not so matured that anyone can use the technique easily.

By paying attention to the fact that many structural data involving relationship can be represented by a colored directed graph, we have proposed Graph-Based Induction (*GBI*) [Yoshida97] which can efficiently extracts typical patterns from a directed graph data of limited types by stepwise pair expansion called "pairwise chunking". The expressiveness of the *GBI* stands between the attribute-value table and the first-order predicate logic.

In this paper, we expand the capability of the *GBI* so that it can handle not only a tree structured data but also a graph data with multi-inputs/outputs nodes and loop structure (including a self-loop) which cannot be treated in the conventional way. The method is verified to work as expected using artificially generated data and we evaluated experimentally the computation time of the implemented program. We show the effectiveness of our approach by applying

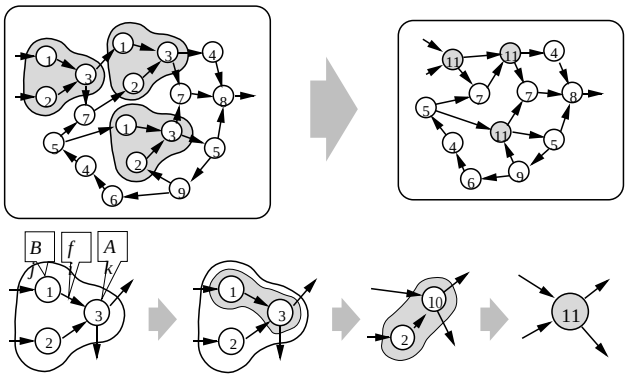

Fig. 1. The idea of graph contraction by pairwise chunking

it to two kinds of the real scale data. One is an application to extracting typical patterns from WWW browsing histories data. Another is an application to extracting classification rules from two kinds of DNA sequence data.

2 Graph-Based Induction

2.1 Framework of Graph-Based Induction

The original *GBI* was so formulated to minimize the graph size by replacing each found pattern with one node that it repeatedly contracted the graph. The graph size definition reflected the sizes of extracted patterns as well as the size of contracted graph. This prevented the algorithm from continually contracting, which meant the graph never became a single node. Because finding a subgraph is known to be NP-hard, the ordering of links is constrained to be identical if the two subgraphs are to match, and an opportunistic beam search similar to genetic algorithm was used to arrive at suboptimal solutions. In this algorithm, the primitive operation at each step in the search was to find a good set of linked pair nodes to chunk (pairwise chunking) [Yoshida95].

The central intuition behind our *GBI* is as follows: a pattern that appears frequently enough in a colored directed graph is worth paying attention to and may represent an important concept in the environment (which is implicitly embedded in the input graph). In other words, the repeated patterns in the input graph represent typical characteristics of the given environment.

Because the search is local and stepwise, we can adopt an indirect index rather than a direct estimate of the graph size to find the promising pairs. On the basis of this notion, we generalize the original *GBI*. The idea of pairwise chunking is given in Figure 1.

The stepwise pair expansion (pairwise chunking) is performed by repeating the following three steps.

Step 1. If there are patterns which are the same as the chunked pattern in the input graph, rewrite each of them to one node of the same label.

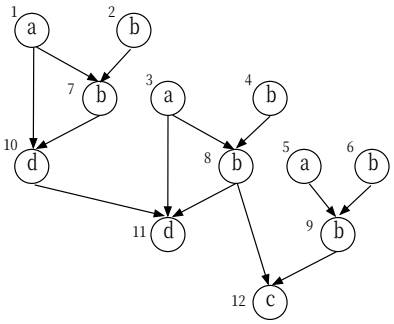

Fig. 2. An example of directed graph

Step 2. Extract all pairs which consist of the connected two nodes in the graph.
Step 3. Select the most *typical* pair among the extracted pairs and register it as the pattern to chunk.

Stepwise pair expansion is performed by repeating the above three steps from the initial state where no typical pattern is yet found. As a result, the characteristics in data can be extracted as a set of typical patterns.

2.2 Graph-Based Induction for General Graph Structured Data

In order to apply GBI to general graph structured data, we adopt a method to represent the graph structured data using a set of table forms by paying attention to the link information between nodes. More concretely, the directed graph as shown in Figure 2 can be represented using Table 1. For example, the first line in this table shows that the node No.1 has node name "a" and also has nodes No.7 and No.10 as child nodes. In this way, directed graph data can be represented using the table forms. Further, the restriction of the link ordering is no more required.

As we count the pair (parent node $\rightarrow$ child node), it is necessary to identify self-loop when the parent node and the child node are of the same kind (*E.g.* $a \rightarrow a$). Therefore, we introduce "self-loop distinction flag".

Moreover, each time we perform the pairwise chunking, we keep link information between nodes in order to be able to restore the chunked pairs to the original patterns. This is realized by keeping two kinds of node information. One is "child node information" that means which node in the pattern the link goes to, and another is the "parent node information" that means which node in the pattern the link comes from. These two kinds of information are also represented by tables (not shown here). Chunking operation can be handled by manipulating these three tables.

The basic algorithm of the proposed method which extends GBI to handle a general graph structured data is shown in Figure 3. In this implemented program, we use the simple "frequency" of pairs as the evaluation function to use for stepwise pair expansion.

Table 1. An example of table form translated from the directed graph

Node No.	Node Name	Child Node No.	
1	a	7	10
2	b	7	
3	d	8	11
4	b	8	
5	a	9	
6	b	9	
7	d	10	
8	b	11	12
9	b	12	
10	a	11	
11	b		
12	c		

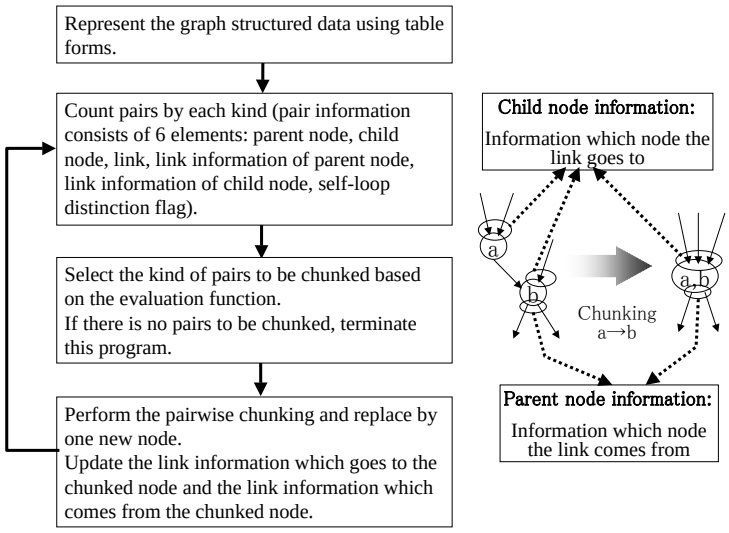

Fig. 3. Algorithm of the proposed method

3 Performance Evaluation

The method was verified to work as expected using artificially generated data and we evaluated experimentally the computation time of the implemented program. The computation time is measured for 30,000 times repetition of program execution excluding the initialization steps.

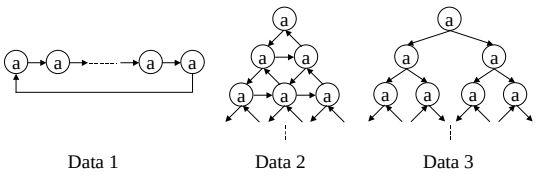

Fig. 4. Graph structured data for evaluation (Data 1, Data 2, Data 3)

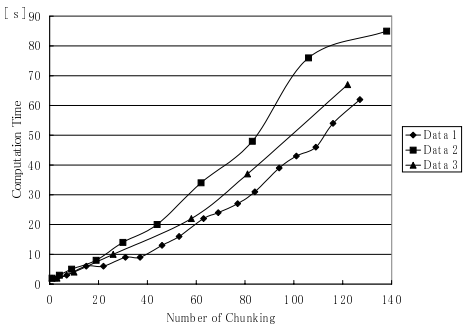

Fig. 5. Computation time and number of chunking (1)

3.1 Computation Time for Uncolored Graphs

At first, we evaluated the computation time using three kinds of graph structured data (Data 1: loop type, Data 2: lattice type, Data 3: tree type) as shown in Figure 4 for which there is only one kind of node label.

The computation time for chunking in the implemented program was measured as we increased the graph size from a small graph with several nodes to the graph which needs chunking about 120 times. Figure 5 shows the relationship between the computation time and the number of chunking.

From this figure, it is found that the computation time increases almost linearly with the number of chunking. And also it is considered that the gradient of each line depends on the average number of links going out from each node.

3.2 Computation Time for Colored Graphs

Next, we evaluated the computation time using three kinds of graph structured data (Data 4: loop type, Data 5: lattice type, Data 6: tree type) as shown in Figure 6 for which there are three kinds of node labels.

The computation time was measured in a similar way for uncolored graphs. Figure 7 shows the relationship between the computation time and the number of chunking. Overall, tendency is the same for uncolored graph. Compared with

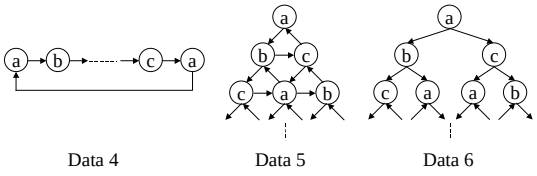

Fig. 6. Graph structured data for evaluation (Data 4, Data 5, Data 6)

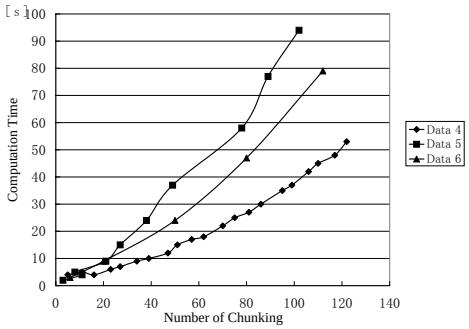

Fig. 7. Computation time and number of chunking (2)

Figure 5, it is conjectured that the number of node labels does not affect the computation time so much.

We further confirmed this tendency for both uncolored and colored graphs by measuring the computation time for chunking as we increase the graph size from 50 nodes to 1000 nodes, where graphs are artificially generated in such a way that the average number of links going out of each node remains a fixed value.

4 Extracting Typical Patterns from WWW Browsing History Data

4.1 WWW Browsing History Data

The performance of the proposed method was examined through a real scale application in this section. The data analyzed is the log file of the commercial WWW server of Recruit Co., Ltd. in Japan. The URLs on WWW form a huge graph, where each URL represents a node that is connected by many links (other URLs). When a client visits the commercial WWW site, he/she browses only a small part of the huge graph in one access session, and the browsing history of the session becomes a small graph structured data. This site's total number of

hit by the nation wide internet users always remains within the third place from the top in every month in Japanese internet record.

4.2 Experimental Method

The basic format of an access record to a URL by a client in the log file is indicated in Figure 8. This access log of the WWW server for a particular day was over $400MB$ and the number of clients who access to the WWW server was about 26,800 on that day. The total number of the URLs involved in this commercial WWW site is about 19,400 and there are a large number of links between them.

As the log file consists of the sequence of the access records, they are initially sorted by the IP addresses, and each subsequence having an identical IP address corresponding to the browsing access history in a session of an individual client is extracted. Then, we transformed the subsequence of each client's visiting history into a graph structured data (total 150,000 nodes). By using all subgraphs transformed in this way as the input data, the proposed method extracted typical patterns in the data. In other words, after removing all kinds of error such as a server error from the access records and sorting the data in order of IP address, we make graph structured data for each IP address (client), and append them into a large table.

In the implemented program, we use the simple "frequency" of pairs as the evaluation function for stepwise pair expansion. We terminated the chunking when we finish finding all chunk patterns which consist of more than a certain number of nodes. We use the frequency threshold 0.1%, 0.05%, 0.025% of total nodes in the graph.

4.3 Experimental Results

We executed the implemented program using the experimental method described in the previous section. Table 2 shows the number of extracted chunk patterns, the number of pairwise chunking and the computation time for each frequency threshold.

Figure 9 indicates the relationship between the threshold and the number of derived patterns consisting of more than 3 nodes. The number of nodes in chunk patterns increases with the decrease of threshold because the larger chunk patterns are derived for the lower threshold where additional nodes are appended to the chunk patterns which have been already extracted in the higher threshold. Chunk patterns derived in the higher threshold is a subset of chunk patterns extracted in the lower threshold.

IP address of a client	Δ	Time stamp of the access	Δ	URL address

Δ:space character

Fig. 8. Basic format of an access record.

Table 2. Experimental result

Threshold (%)	0.10	0.05	0.025
No. of Extracted Patterns	33	106	278
No. of Pairwise Chunking	9455	17443	26672
Computation Time ($Sec.$)	1374	1734	2187

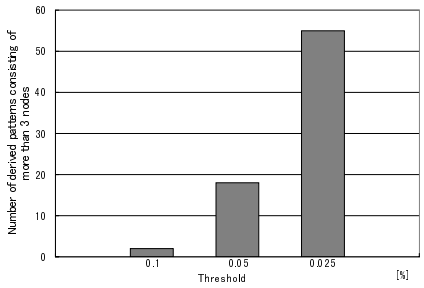

Fig. 9. Relationship between the threshold and the number of derived patterns consisting of more than 3 nodes

Several examples of the extracted chunk patterns are shown below.

$a)$ $/NAVI/CATEGORY/Sub/s03/ss13/d111.html$
$\rightarrow /NAVI/CATEGORY/Sub/s03/ss13/d112.html$
$\rightarrow /NAVI/CATEGORY/Sub/s03/ss13/d113.html$
$b)$ $/NAVI/CATEGORY/Sub/s01.html$
$\rightarrow /NAVI/CATEGORY/Sub/s01/ss06.html$
$\rightarrow /NAVI/CATEGORY/Sub/s01/ss06/d07.html$
$c)$ $/NAVI/mscategory/Sub/s12.html$
$\rightarrow /NAVI/mscategory/Sub/s08.html$
$\rightarrow /NAVI/mscategory/Sub/s02.html$
$\rightarrow /service/shank/NAVI/COOL/cool2.html$

Browsing pattern $a)$ is an example that clients follow some URLs in the same directory and the number of this pattern was 152 ($i.e.$, 152 clients followed this pattern). Browsing pattern $b)$ is an example that clients go deeper into directories step by step and the number of this pattern was 144. Browsing pattern $c)$ is an example that clients jump to the URLs in a different directory after following some URLs in the same directory and the number of this pattern was 72.

It is natural that many patterns similar to $a)$ or $b)$ have been extracted because of the structure of this WWW site. However, it is more interesting to note that some patterns such as $c)$ also have been extracted if the contents of each URL were available to us.

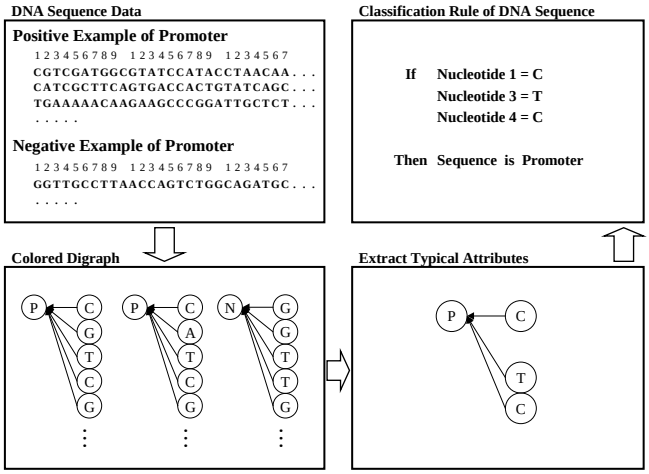

Fig. 10. Extraction of classification rules from DNA sequence data

5 Extracting Classification Rules from DNA Data

5.1 Application to Promoter DNA Sequence Data

In this section, the real-world task of recognizing biological concepts in DNA sequences is investigated. In particular, the task is to recognize *promoters* in strings that represent nucleotides (one of A, G, T, or C). A promoter is a genetic region which initiates the first step in the expression of an adjacent gene (*transcription*). This data set is one of the UCI Machine Learning Repository [Blake98]. The input features are 57 sequential DNA nucleotides and the total number of instances is 106 including 53 positive instances (sample promoter sequences) and 53 negative instances (non-promoter sequences).

Figure 10 illustrates the process of mapping the problem into a colored directed graph, using *GBI* method to extract patterns and interpreting them as classification rules. In mapping the cases in the data set into the graph structure, we construct one subgraph for each sequence in the data set. The subgraph consists of a root node and a set of leaf nodes. The color of the root node of the subgraph specifies whether the corresponding sequence represents a promoter sequence or not, which means the class information (positive or negative). The color of the i-th leaf specifies the nucleotide (one of A, G, T, or C).

In case of the classification problem, we interpret the root node as a class node and the links attached to it as the primary attributes. The node at the other end of each link is the value of the attribute, which can have secondary attributes, although this is not the case for this simple DNA problem. Thus, each attribute can have its own attributes recursively, and the graph (*i.e.*, each instance of the

Table 3. Experimental results

Learning Method	ID3	C4.5	*GBI*
No. of Errors /106	19	18	16

data) becomes a directed tree. Here, we have to select the attribute and its value pair that best characterizes the class.

The chunk patterns derived by *GBI* are tried to match for the test cases in the following way. The chunk patterns which have lower evaluation function value (frequency) are tried to match first. If the frequency of the chunk patterns is same, those which have more nodes in the pattern are tried to match first. That is, more specific rules are tried to match with higher priority.

Table 3 shows the experimental results (number of errors in total 106 cases) in comparison with other learning methods such as ID3, C4.5, which are evaluated by leaving-one-out. From this table, it is found that the error rate of *GBI* is lower than the standard tree-induction program ID3 and C4.5.

5.2 Application to Splice DNA Sequence Data

Splice junctions are the points on DNA sequence at which "superfluous" DNA is removed during the process of protein creation. The problem is to recognize the boundaries between *exons* (the parts of the DNA sequence retained after splicing) and *introns* (the parts of the DNA sequence that are spliced out) in a given sequence of DNA. This problem consists of two subtasks: recognizing exon/intron boundaries (referred to as *E/I*), and recognizing intron/exon boundaries (referred to as *I/E*).

This data set contains 3190 cases, of which 25% are *I/E*, 25% are *E/I* and the remaining 50% are *Neither*. Each example consists of a 60 nucleotide DNA sequence categorized according to the type of boundary at the center of the sequence.

In mapping the cases in the data set into the graph structure, we constructed one subgraph for each sequence in the data set, in the same way as in the Promoter DNA data. The subgraph consists of a root node and a set of leaf nodes. The color of the root node of the subgraph specifies whether the corresponding sequence represents one of the *I/E*, *E/I* and *Neither*. The color of the i-th leaf specifies the nucleotide (one of A, G, T, or C).

The chunk patterns derived by *GBI* are tried to match for the test cases in the same way as mentioned in the previous subsection.

Table 4 shows the experimental results (error rate) in comparison with other learning methods such as ID3, C4.5, which are evaluated by 10-fold cross-validation. From this table, it is found that the error rate of *GBI* is lower than ID3 and is almost the same as the standard tree-induction program C4.5.

Table 4. Experimental results

Learning Method	ID3	C4.5	*GBI*
Error Rate (%)	10.6	8.0	8.8

6 Related Work

Most of the current methods for extracting knowledge from databases have difficulties in handling the growing amount of structural data that express relationships among data objects. However, there are some research work for discovering knowledge in structural data, especially graph structured data.

[Cook94] proposed the substructure discovery system called SUBDUE which discovers interesting and repetitive subgraphs in a labeled graph representation. Experiments show SUBDUE's applicability to several domains, such as molecular biology, image analysis and computer-aided design. SUBDUE expands one subgraph based on the Minimum Description Length (MDL) principle. Therefore, the number of substructure which is discovered in a graph is always one. On the other hand, *GBI* for general graph structured data which is proposed in this paper can extract multiple patterns based on the evaluation function for stepwise pair expansion.

[Wallace96] presented a Bayesian approach to the discovery of causal models based on Minimum Message Length (MML), which is one way to realize Occam's razor just like MDL. The MML induction approach can recover causal models from sample graph data without incorporating background knowledge. While this approach is towards automated learning of causal model using MML, this applicability to huge graph structured data is not clear so far.

[Inokuchi99] applied Basket Analysis to mine association rules from the graph structured data. The Basket Analysis [Agrawal94] derives frequent itemsets and association rules having *support* and *confidence* levels greater than their thresholds from massive transaction data. In [Inokuchi99], a simple preprocessing of data enabled to use a standard Basket Analysis technique for a graph structured data. However, each node must have a distinct label with this approach.

7 Conclusion

In this paper, we showed how we can expand the capability of the Graph-Based Induction algorithm to handle more general graphs, i.e., directed graphs with 1) multiple inputs/outputs nodes and 2) loop structure (including a self-loop). The algorithm was implemented and verified to work as expected using first artificially generated data and second two kinds of real-world data: World Wide Web browsing data and DNA sequence data. The algorithm runs almost linearly to the graph size and can indeed find interesting patterns.

The followings are in progress: 1) investigating the sensitivity of chunk ordering, 2) using extended chunks as constructed of new features for standard

decision tree algorithm, 3) using statistical index (e.g. Gini Index [Breiman84]) or the description length in stead of the simple "frequency" as the evaluation function for stepwise expansion, 4) introducing a new index which corresponds to the notion of "similarity" of human concept, 5) applying different kinds of graph structured data in the real-world such as chemical structured data.

Acknowledgment

The authors are very grateful to Mr. Kouhei Kumazawa and Mr. Shoei Arai in Recruit Co., Ltd., Japan for providing the access log of the commercial WWW site.

References

[Agrawal94] R. Agrwal and R. Srikant. First Algorithms for Mining Association Rules. *Proc. of the 20th VLDB Conference,* pp. 487–499, 1994.

[Blake98] C. Blake, E. Keogh, and C. J. Merz. UCI Repository of Machine Learning Databases. http://www.ics.uci.edu/ mlearn/MLRepository.html, 1998.

[Breiman84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth & Brooks/Cole Advanced Books & Software, 1984.

[Clark89] P. Clark and T. Niblett. The CN2 Induction Algorithm. *Machine Learning* Vol. 3, pp. 261–283, 1989.

[Cook94] D. J. Cook and L. B. Holder. Substructure Discovery Using Minimum Description Length and Background Knowledge. *Journal of Artificial Intelligence Research*, Vol. 1, pp. 231–255, 1994.

[Inokuchi99] A. Inokuchi, T. Washio and H. Motoda. Basket Analysis for Graph Structured Data, *Proc. of the Third Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'99)*, pp. 420–431, 1999.

[Michalski90] R. S. Michalski. Learning Flexible Concepts: Fundamental Ideas and a Method Based on Two-Tiered Representaion. *In Machine Learning, An Artificial Intelligence Approach*, Vol. III, (eds. Kodrtoff Y. and Michalski T.), pp. 63–102, 1990.

[Muggleton89] S. Muggleton and L. de Raedt. Inductive Logic Programming: Theory and Methods. *Journal of Logic Programming* Vol. 19, No. 20, pp. 629–679, 1994.

[Quinlan86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, Vol. 1, pp. 81–106, 1986.

[Wallace96] C. Wallace, K. B. Korb and H. Dai. Causal Discovery via MML, *Proc. of the 13th International Conference on Machine Learning (ICML'96)*, pp. 516–524, 1996.

[Yoshida95] K. Yoshida and H. Motoda. CLIP: Concept Learning from Inference Pattern, *Artificial Intelligence*, Vol. 75, No. 1, pp. 63–92, 1995.

[Yoshida97] K. Yoshida and H. Motoda. Inductive Inference by Stepwise Pair Extension (in Japanese), *Journal of Japanese Society for Artificial Intelligence*, Vol. 12, No. 1, pp. 58–67, 1997.