Analysis of Time Series of Graphs: Prediction of Node Presence by Means of Decision Tree Learning

Horst Bunke[1], Peter Dickinson[2], Christophe Irniger[1], Miro Kraetzl[2]

[1] Department of Computer Science, University of Bern
Neubrückstrasse 10, CH-3012 Bern, SWITZERLAND
{bunke,irniger}@iam.unibe.ch
[2] Intelligence, Surveillance and Reconnaissance Division, Defence Science and Technology Organisation, Edinburgh SA 5111, Australia
{Peter.Dickinson,Miro.Kraetzl}@dsto.defence.gov.au

Abstract. This paper is concerned with time series of graphs and proposes a novel scheme that is able to predict the presence or absence of nodes in a graph. The proposed scheme is based on decision trees that are induced from a training set of sample graphs. The work is motivated by applications in computer network monitoring. However, the proposed prediction method is generic and can be used in other applications as well. Experimental results with graphs derived from real computer networks indicate that a correct prediction rate of up to 97% can be achieved.

1 Introduction

Time series, or sequence, data are encountered in many applications, such as financial engineering, audio and video databases, biological and medical research, and weather forecast. Consequently, the analysis of time series has become an important area of research [1]. Particular attention has been paid to problems such as time series segmentation [2], retrieval of sequences or partial sequences [3], indexing [4], classification of time series [5], detection of frequent subsequences [6], periodicity detection [7] and prediction [8–10].

Typically a time series is given in terms of symbols, numbers, or vectors [1]. In the current paper we go one step further and consider time series of graphs. A time series of graphs is a sequence, $s = g_1, \ldots, g_n$, where each g_i is a graph. In a recent survey it has been pointed out that graphs are a very suitable and powerful data structure for many operations needed in data mining in intelligent information processing [11]. As a matter of fact, traditional data structures, such as sequences of symbols, numbers, or vectors, can all be regarded as a special case of sequences of graphs.

The work presented in this paper is motivated by one particular application, which is computer network monitoring [12, 13]. In this application, graphs play an important role [14]. The basic idea is to represent a computer network by

a graph, where the clients and servers are modelled by nodes, and physical connections correspond to edges. If the state of the network is captured at regular points in time and represented as a graph, a sequence, or time series, of graphs is obtained that formally represents the network. Given such a sequence of graphs, abnormal network events can be detected by measuring the dissimilarity, or distance, between a pair of graphs that represent the network at two consecutive points in time. Typically an abnormal event manifests itself through a large graph distance [14].

In the current paper we address a different problem, viz. the recovery of incomplete network knowledge. Due to various reasons it may happen that the state of a network node or a network link can't be properly captured during network monitoring. This means that it is not known whether a certain node or edge is actually present or not in the graph sequence at a certain point in time. In this paper we describe a procedure that is able to recover missing information of this kind. This procedure is capable to make a decision as to the presence or absence of such a network node or edge. An information recovery procedure of this kind can also be used to predict, at time t, whether a certain computer or a certain link will be present, i.e. active, in the network at the next point in time, $t + 1$. Such procedures are useful in computer network monitoring in situations where one or more network probes have failed. Here the presence, or absence, of certain nodes and edges is not known. In these instances, the network management system would be unable to compute an accurate measurement of network change. The techniques described in this paper can be used to determine the likely status of this missing data and hence reduce false alarms of abnormal change.

Although the motivation of our work is in computer network monitoring, the methods described in this paper are fairly general and can be applied in other domains as well. Our proposed recovery scheme is based on decision tree learning as described in [15]. Basically we cast the recovery and prediction task into a classification framework, where one wants to decide whether a node is present in the network at a certain point in time or not. Clearly such a decision can be understood as a two-class classification problem, with one class, Ω_0, indicating the absence of the node in question, and another class, Ω_1, representing its presence in the network at a given point in time.

The rest of this paper is organized as follows. Basic terminology and notation will be introduced in the next section. Then, in Section 3, we will describe our novel information recovery and prediction scheme. Experimental results with this new scheme will be presented in Section 4 and conclusions drawn in Section 5.

2 Basic Concepts and Notation

A labeled graph is a 4-tuple, $g = (V, E, \alpha, \beta)$, where V is the finite set of nodes, $E \subseteq V \times V$ is the set of edges, $\alpha : V \rightarrow L$ is the node labeling function, and $\beta : E \rightarrow L^{'}$ is the edge labeling function, with L and $L^{'}$ being the set of node and edge labels, respectively. In this paper we focus our attention on a special

class of graphs that are characterized by unique node labels. That is, for any two nodes, $x, y \in V$, if $x \neq y$ then $\alpha(x) \neq \alpha(y)$. Properties of this class of graphs have been studied in [16]. In particular it has been shown that problems such as graph isomorphism, subgraph isomorphism, maximum common subgraph, and graph edit distance computation can be solved in time that is only quadratic in the number of nodes of the larger of the two graphs involved.

To represent graphs with unique node labels in a convenient way, we drop set V and define each node in terms of its unique label. Hence a graph with unique node labels can be represented by a 3-tuple, $g = (L, E, \beta)$ where L is the set of node labels occurring in g, $E \subseteq L \times L$ is the set of edges, and $\beta : E \rightarrow L^{'}$ is the edge labeling function [16]. The terms "node label" and "node" will be used synonymously in the remainder of this paper.

In this paper we will consider time series of graphs, i.e. graph sequences, $s = g_1, g_2, \ldots, g_N$. The notation $g_i = (L_i, E_i, \beta_i)$ will be used to represent individual graph g_i in sequence s; $i = 1, \ldots, N$. Motivated by the computer network analysis application considered in this paper, we assume the existence of a universal set of node labels, or nodes, $\mathcal{L}$, from which all node labels that occur in a sequence s are drawn. That is, $L_i \subseteq \mathcal{L}$ for $i = 1, \ldots, N$ and $\mathcal{L} = \bigcup_{i=1}^{N} L_i$.[1]

Given a time series of graphs, $s = g_1, g_2, \ldots, g_N$, and its corresponding universal set of node labels, $\mathcal{L}$, we can represent each graph, $g_i = (L_i, E_i, \beta_i)$, in this series as a 3-tuple $(\gamma_i, \delta_i, \widehat{\beta}_i)$ where

- $\gamma_i : \mathcal{L} \rightarrow \{0, 1\}$ is a mapping that indicates whether node l is present in g_i or not. If l is present in g_i, then $\gamma_i(l) = 1$; otherwise $\gamma_i(l) = 0$.[2]
- $\delta_i : \mathcal{L}^{'} \times \mathcal{L}^{'} \rightarrow \{0, 1\}$ is a mapping that indicates whether edge (l_1, l_2) is present in g_i or not; here we choose $\mathcal{L}^{'} = \{l \mid \gamma_i(l) = 1\}$, i.e. $\mathcal{L}^{'}$ is the set of nodes that are actually present in g_i.
- $\widehat{\beta}_i : \mathcal{L}^{'} \times \mathcal{L}^{'} \rightarrow L^{'}$ is a mapping that is defined as follows:
  $\widehat{\beta}_i(e) = \begin{cases} \beta_i(e), & \text{if } e \in \{(l_1, l_2) \mid \delta_i(l_1, l_2) = 1\} \\ \text{undefined}, & \text{otherwise} \end{cases}$

The definition of $\widehat{\beta}_i(e)$ means that each edge e that is present in g_i will have label $\beta_i(e)$. The 3-tuple $(\gamma_i, \delta_i, \widehat{\beta}_i)$ that is constructed from $g_i = (L_i, E_i, \beta_i)$ will be called the *characteristic representation* of g_i, and denoted by $\chi(g_i)$. Clearly, for any given graph sequence $s = g_1, g_2, \ldots, g_N$ the corresponding sequence $\chi(s) = \chi(g_1), \chi(g_2), \ldots, \chi(g_N)$ can be easily constructed and is uniquely defined. Conversely, given $\chi(s) = \chi(g_1), \chi(g_2), \ldots, \chi(g_N)$ we can uniquely reconstruct $s = g_1, g_2, \ldots, g_N$.

In the current paper we'll pay particular attention to graph sequences with missing information. There are two possible cases of interest. First it may not

[1] In the computer network analysis application $\mathcal{L}$ will be, for example, the set of all unique IP host addresses in the network. Note that in one particular graph, g_i, usually only a subset is actually present. In general, $\mathcal{L}$ may be any finite or infinite set.

[2] One can easily verify that $\{l \mid \gamma_i(l) = 1\} = L_i$.

be known whether node l is present in graph g_i or not. In other words, in $\chi(g_i)$ it is not known whether $\gamma_i(l) = 1$ or $\gamma_i(l) = 0$. Secondly, it may not be known whether edge (l_1, l_2) is present in g_i, which is equivalent to not knowing, in $\chi(g_i)$, whether $\delta_i(l_1, l_2) = 1$ or $\delta_i(l_1, l_2) = 0$. In this paper, we focus on the case of missing node information. To cope with the problem of missing information and in order to make our notation more convenient, we extend function γ in the characteristic representation, $\chi(g)$, of graph $g = (L, E, \beta)$ by including the special symbol ? in the range of values of each function to indicate the case of missing information. That is, we write $\gamma(l) =?$ if it is unknown whether node l is present in g or not.

3 Recovery of Missing Information Using Decision Trees

Our goal is to construct a function that computes $\gamma_t(l)$ for a node, l, given some data extracted from time series $g_1, g_2, \ldots, g_t$ as input. In the approach proposed in this paper the function that computes $\gamma_t(l)$ will actually use only information extracted from g_t. However, graphs $g_1, \ldots, g_{t-1}$ will be used as a training set, i.e. they are used to learn this function.

The approach proposed in this paper is based on decision trees. Decision tree classifiers have often been used for the purpose of object classification. An object, $\mathbf{x}$, is given in terms of the values of d different features and represented by means of a d-dimensional vector, i.e. $\mathbf{x} = (x_1, \ldots, x_d)$. The feature values, x_i, $1 \leq i \leq d$, can be numerical or non-numerical. It is possible that one or several feature values are unknown. To classify an object means to assign it to a class, Ω_i, out of a number of given classes, $\Omega_1, \ldots, \Omega_c$. For all further technical details we refer the reader to [15].

Assume we want to make a decision as to $\gamma_t(l) = 0$ or $\gamma_t(l) = 1$, given $\gamma_t(l) =?$. Actually, this decision problem can be transformed into a classification problem as follows. The network at time t, g_t, corresponds to the unknown object to be classified. Network g_t is described by means of a feature vector, $\mathbf{x} = (x_1, \ldots, x_d)$, and the decision as to $\gamma_t(l) = 0$ or $\gamma_t(l) = 1$ can be interpreted as a two-class classification problem, where $\gamma_t(l) = 0$ corresponds to class Ω_0 and $\gamma_t(l) = 1$ corresponds to class Ω_1. As features $x_1, \ldots, x_d$ that represent the unknown object $\mathbf{x}$, i.e. graph g_t, one can use, in principle, any quantity that is extractable from graphs $g_1, \ldots, g_t$. In this paper we consider the case where these features are extracted from graph g_t exclusively. Assume that the universal set of node labels is given by $\mathcal{L} = \{l_0, l_1, \ldots, l_D\}$, and assume furthermore that it is node label l_0 for which we want to make a decision as to $\gamma_t(l_0) = 0$ or $\gamma_t(l_0) = 1$, given $\gamma_t(l_0) =?$. Then we set $d = D$ and use the D-dimensional binary feature vector $(\gamma_t(l_1), \ldots, \gamma_t(l_D))$ to represent graph g_t. In other words, $\mathbf{x} = (\gamma_t(l_1), \ldots, \gamma_t(l_D))$. This feature vector is to be classified as either belonging to class Ω_0 or Ω_1. The former case correspond to deciding $\gamma_t(l_0) = 0$, and the latter to $\gamma_t(l_0) = 1$. Intuitively, using $(\gamma_t(l_1), \ldots, \gamma_t(l_D))$ as a feature vector for the classification of g_t means we make a decision as to the presence or absence

of l_0 in g_t depending on the presence or absence of all other nodes from $\mathcal{L}$ in g_t. [3]

For the implementation of the classification procedure described in the last paragraph, we need a training set. For the training set we can use all previous graphs in the given time series, i.e. $g_1, \ldots, g_{t-1}$. From each graph, g_i, we extract the D-dimensional feature vector

$$\mathbf{x_i} = (\gamma_i(l_1), \ldots, \gamma_i(l_D)) \tag{3.1}$$

So our training set becomes $L = \{\mathbf{x_1}, \ldots, \mathbf{x_{t-1}}\}$. We do need to assign the proper class to each element of the training set. This can be easily accomplished by assigning class Ω_0 to $\mathbf{x_i}$ if $\gamma_i(l_0) = 0$; otherwise, if $\gamma_i(l_0) = 1$, we assign class Ω_1 to $\mathbf{x}_i$; $i = 1, \ldots, t-1$.

Given such a training set, constructed from $g_1, \ldots, g_{t-1}$, we can now apply any known procedure to infer a decision tree from training set L. In the experiments described in Section 4, we have used C4.5 [15]. Once the decision tree has been produced, it is straightforward to classify feature vector $\mathbf{x_t}$ (see Eq. (3.1)), which describes g_t, as belonging to Ω_0 or Ω_1.

Decision tree classifiers are able to deal with unknown attribute values. This is important in our application because we must expect that not only information about node l_0 in g_t is missing, but also about other nodes, l_i, in g_t, where $i \in \{1, \ldots, D\}$. Similarly, when building the decision tree from training set $L = \{\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}\}$, there may be graphs, g_i, $i \in \{1, \ldots, t-1\}$ where it is not known for some nodes whether they are present or not in g_i. Hence some of the $\gamma_i(l_j)$ may be unknown. Fortunately, decision tree induction methods are able to cope with such cases of missing data [15].

The procedure described in this section is based on two assumptions. The first assumption is that there is some kind of correlation between the occurrence of a node, l, in graph g_t, and the occurrence of some (or all) other nodes in the same graph. In other words, we assume that the behaviour of node l is dependent, in some way, on the behaviour of the other nodes. Note, however, that we don't need to make any assumptions as to the mathematical nature of this dependency. Our second assumption is that there is some stationarity in the dependency between l and the other nodes. Using graphs $g_1, \ldots, g_{t-1}$ as a training set to derive a classifier that makes a decision pertaining to graph g_t will work well only if the dependency between l and the other nodes in g_t is of the same nature as in $g_1, \ldots, g_{t-1}$.

In a practical setting it may be computationally too demanding to infer a decision tree at each point of time, t. Hence it may be preferable to do an update of the actual decision tree only after a certain period of time has elapsed. Moreover, in the decision tree updating process it is possible to use only part of the network history. This means that for the construction of the decision tree for g_t, we don't use $g_1, \ldots, g_{t-1}$, but focus on only the M most recent

[3] Note that in principle also information about edges could be incorporated in the feature vector. However such an extension would increase the space complexity from $O(D)$ to $O(D^2)$.

	S_1	S_2	S_3	S_4
Number of graphs in sequence	102	292	202	99
Size of smallest graph in sequence	38	85	15	572
Size of largest graph in sequence	94	154	329	10704
Average size of graphs in sequence	69.7	118.5	103.9	5657.8

Table 1. Characterisation of the graph sequences used in the experiments

graphs $g_{t-M}, \ldots, g_{t-1}$. This is particularly advisable if there is evidence that the behaviour of the network is not perfectly stationary, but changing over time.

4 Experimental Results

The method described in Section 3 of this paper has been implemented and experimentally evaluated on real network data. For the experiments four time series of graphs, S_1, S_2, S_3 and S_4, acquired from existing computer networks have been used. Characteristics of these graph sequences are shown in Table 1, where the size of a graph is defined as the number of its nodes. All four series represent logical communications on the network. Series S_1, S_2 and S_4 were derived from data collected from a large enterprise data network, while S_3 was collected from a wireless LAN used by delegates during the World Congress for Information Technology (WCIT2002). The nodes in each graph of S_1 and S_2 represent business domains in the network, while in S_3 and S_4 they represent individual IP addresses. Note that all graph sequences are complete, i.e. there are no missing nodes and edges in these sequences.

For the experiments described in this section each time series is divided into two disjoined sets of graphs. The first set, G_1, consists of all graphs g_i with index i being an odd number (i.e. graphs $g_1, g_3, g_5, \ldots$), while the other set, G_2, includes all graphs with an even index i (i.e. graphs $g_2, g_4, \ldots$). First, set G_1 is used as the training set for decision tree induction and G_2 serves as the test set. Then G_1 and G_2 change their role, i.e. G_1 becomes the test and G_2 the training set. In the learning phase an individual decision tree is build for each node, i.e. for each label, l, belonging to the universal set of node labels, $\mathcal{L}$, as described in Section 3. Once all decision trees have been learned, testing takes place by assuming, for each graph g_i from the test set and each node label $l \in L$, that $\gamma_i(l) =?$. The decision tree learned for label l is used to decide either $\gamma_i(l) = 0$ or $\gamma_i(l) = 1$. Then the predicted value is compared to the real value. For each graph, g_i, in the test set we count the number of nodes that have been correctly predicted and divide this number by the total number of nodes in g. Splitting the considered time series of graph, S, into disjoined sets, G_1 and G_2, is equivalent to performing a two-fold cross-validation, where each graph in time series S serves one time as a training and one time as a test sample. Clearly, a number of alternative scenarios for testing the proposed method are feasible.

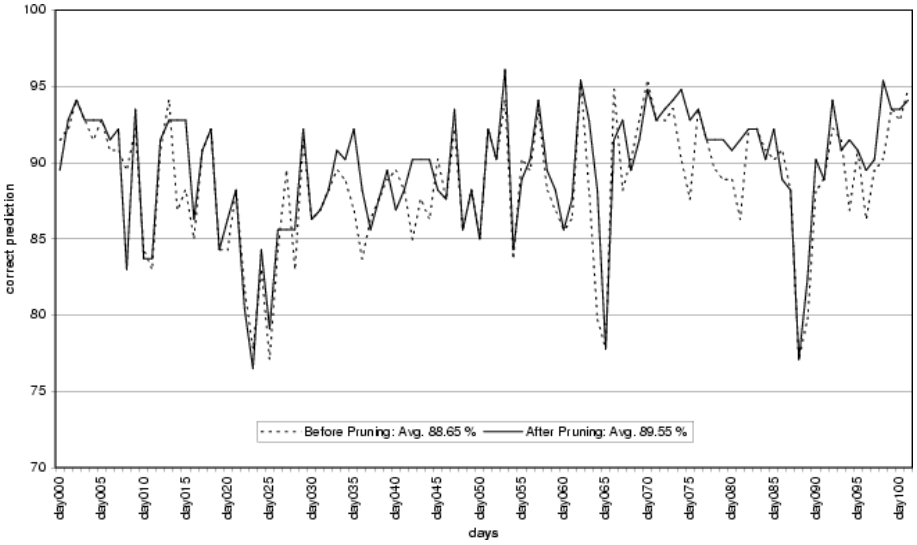

Fig. 1. Percentage of correctly predicted nodes in sequence S_1, using both pruned and non-pruned decision trees

For example, instead of performing just a two-fold cross-validation, one could split the dataset into $n > 2$ disjoint subsets and do an n-fold cross-validation.

Fig. 1 shows the correct prediction rate for each graph in time series S_1. Two different versions of the decision tree induction procedure were applied. The first version generates a tree without any pruning, while the second version applies pruning, as described in [15].

For each of the two versions the percentage of correctly predicted nodes in the corresponding graph is shown. We observe that the correct prediction rate is around 89% on the average. This is a remarkably high value taking into consideration that for a two-class classification problem, such as the one considered here, random guessing would give us an expected performance of only 50%. There are some obvious drops in prediction performance in Fig. 1, for example between time 20 and time 25, and around time 65. These drops correspond to abnormal events in the underlying computer network where major changes in network topology take place. Obviously these abnormal events don't follow the normal network behaviour represented in the training set. But the correct prediction rate is still quite high (more than 75% in any case).

From Fig. 1 it is hard to tell which of the two decision tree induction methods, including or excluding pruning, gives the better overall result. However if we average the correct prediction rate over all graphs in the time series, we get values of $89,5\%$ and $88,6\%$ for pruned and non-pruned trees, respectively. Hence using decision tree pruning gives us a slightly better performance.

Fig. 2 shows the percentage of correctly predicted nodes in sequence S_2 for pruned and non-pruned trees. We observe again a high correct prediction rate,

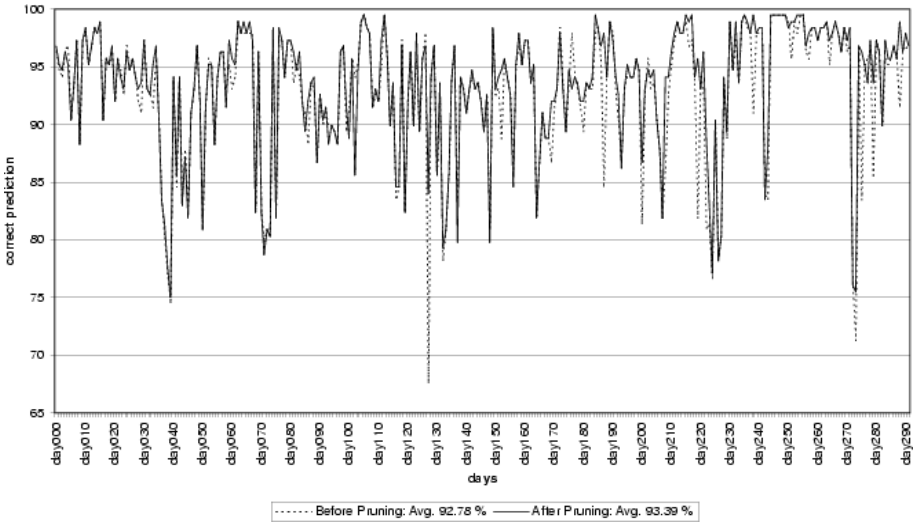

Fig. 2. Percentage of correctly predicted nodes in sequence S_2, using both pruned and non-pruned decision trees

	S_1	S_2	S_3	S_4
Pruned trees	89,5	93,4	96,9	89,4
Non-pruned trees	88,6	92,8	96,3	87,4

Table 2. Summary of correct prediction rates for sequences S_1 to S_4

with the curve showing a bit more jitter than Fig. 1. The correct prediction rate, averaged over all graphs of Sequence S_2, is $93,4$ for pruned and $92,8$ for non-pruned decision trees.

Results for sequences S_3 and S_4 are similar. A summary of the correct prediction rates, averaged over all graphs in a sequence, for both pruned and non-pruned decision trees, is shown in Table 2. From this figure we can conclude that for all time series used in this study quite high prediction rates were achieved. Pruned trees are consistently slightly better than non-pruned trees.

5 Conclusions

The problem of incomplete knowledge recovery and prediction of the behaviour of nodes in time series of graphs is studied in this paper. Formally, this task is formulated as a classification problem where nodes with an unknown status are to be assigned to one of the classes 'present in' (Ω_1) or 'absent from' (Ω_0) the actual graph. A decision tree learning scheme is proposed in order to solve this classification problem. The motivation of this work derives from the field of computer network monitoring. However the proposed framework for graph

sequence analysis is fairly general and can be applied in other domains as well. In computer network monitoring, prediction procedures, as studied in this paper, are important for patching missing network data in instances where one or more network probes have failed. Without such procedures, the network management system would have diminished capability in detecting abnormal change.

The proposed prediction procedure is straightforward to implement using decision tree induction software tools. Excellent performance with correct prediction rates ranging between about 89% and 97% with pruned trees has been achieved, using the proposed method on four independent data sets acquired from real computer networks.

The task in this paper has been cast as a classification problem. Consequently, not only decision trees, but also any other type of classifier can be applied, for example, neural network, Bayes classifier, nearest neighbor, or support vector machine. However, decision trees have at least two advantages. First, they can cope with missing data in both the training and test set. Secondly, it is possible to extract, from a trained decision tree, a number of rules that are interpretable by a human expert. This second aspect has not been stressed in our work yet, but is interesting to be investigated in future research.

The proposed schemes can be extended in a variety of ways. First of all, a prediction scheme, similar to the one proposed in this paper for nodes, can be designed for edges. In both, node and edge prediction, node as well as edge information can be utilized. That is, feature vectors as described in Section 3 can be extended to including information about the presence or absence of edges and they can be used for both node and edge prediction. Dealing with edges, however, introduces a substantially higher cost from the computational complexity point of view, because in a graph with n nodes we may have up to $O(n^2)$ edges, i.e. the complexity of our algorithms is going up from $O(n)$ to $O(n^2)$.

In the scheme proposed in Section 3 we have used within-graph context exclusively, i.e. the only information that is used in order to make a decision as to the presence or absence of a certain node in a graph, g, comes from that graph, g. One could use, however, also context in time. This means that we include information about the past behaviour of a network node in order to decide about its presence in the actual graph. From the conceptual point of view it is straightforward to integrate information of this kind in the proposed decision tree learning procedures. A more detailed investigation of this issue is left to future research.

Acknowledgement

The authors are thankful to J. Marbach and T. Varga for valuable contributions to this paper.

References

1. Last, M., Kandel, A., Bunke, H. (eds.): *Data Mining in Time Series Databases*, World Scientific, 2004

2. Keogh, E. et al: Segmenting time series: A survey and novel approach, in [1], 1-21
3. Kahveci, T., Singh, K.: Optimizing similarity search for arbitrary length time series queries, IEEE Trans. KDE, Vol. 16, No 2, 2004, 418-433
4. Vlachos, M. et al.: Indexing time-series under conditions of noise, in [1], 67-100
5. Zeira, G. et al: Change detection in classification models induced from time series data, in [1], 101-125
6. Tanaka, H., Uehara, K.: Discover motifs in multi-dimensionaltime-series using the principle component analysis and the MDL principle, in Perner, P., Rosenfeld, A. (eds.): Machine Learning and Data Mining in Pattern Recognition, *Proc. 3rd Int. Conference*, Springer LNAI 2734, 2003, 252-265
7. Yang, J., Wang, W., Yu, P.S.: Mining asynchronous periodic patterns in time series data, IEEE Trans. KDE, Vol. 15, No 3, 2003, 613-628
8. Schmidt, R., Gierl, L.: Temporal abstractions and case-based reasoning for medical course data: two prognostic applications, in Perner, P. (ed.): *Machine Learning in Pattern Recognition Proc. 2nd Int. Workshop*, Springer, LNAI 2123, 2001, 23-34
9. Fung, G.P.C.F., Yu, D.X., Lam, W.: News sensitive stock trend prediction, in Chen, M.-S., Yu, P.S., Liu, B. (eds.): *Advances in Knowledge Discovery and Data Mining, Proc. 6th Pacific-Asia Conference, PAKDD*, Springer, LNAI 2336, 2002, 481-493
10. Povinelli, R.F., Feng, X.: A new temporal pattern identification method for characterization and prediction of complex time series events, IEEE Trans. KDE, Vol. 15, No 2, 2003, 339-352
11. Bunke, H.: Graph-based tools for data mining and machine learning, in Perner, P., Rosenfeld, A. (eds.): Machine Learning and Data Mining in Pattern Recognition, *Proc. 3rd Int. Conference*, Springer LNAI 2734, 2003, 7-19
12. Hayes, S.: Analysing network performance management, IEEE Communications Magazine, 31 (5):52-58, May 1993
13. Higginbottom, G.N.: *Performance Evaluation of Communication Networks*, Artech House, Massachusetts, 1998
14. Bunke, H., Kraetzl, M., Shoubridge, P., Wallis, W.: Detection of abnormal change in time series of graphs, *Journal of Interconnection Networks*, Vol. 3, Nos 1,2, 2002, 85-101
15. Quinlan, R.: C4.5: Programs for Machine Learning, Morgen Kaufmann Publ., 1993
16. Dickinson, P., Bunke, H., Dadej, A., Kraetzl, M.: Matching graphs with unique node labels, accepted for publication in *Pattern Analysis and Applications*