Decision Tree Structures for Graph Database Filtering

Christophe Irniger and Horst Bunke

Department of Computer Science, University of Bern
Neubrückstrasse 10, CH-3012 Bern, Switzerland
{bunke,irniger}@iam.unibe.ch

Abstract. In structural pattern recognition it is often required to match an unknown sample against a database of candidate patterns in order to find the most similar prototype. If the patterns are represented using graphs, the sample's graph is matched against a database of model graphs and the pattern recognition problem is turned into a graph matching problem. Graph matching is a powerful yet computationally expensive procedure. If the unknown sample is matched against a whole database of prototypes, the size of the database is introduced as an additional factor into the overall complexity of the matching process. To reduce the influence of that factor an approach based on machine learning techniques is proposed in this paper. The graphs are represented using feature vectors. Based on these vectors a decision tree is built to index the database. The decision tree allows at runtime to eliminate a number of graphs from the database as possible matching candidates. Experimental results are reported demonstrating the efficiency of the proposed filtering procedure. The work presented in this paper extends previous studies from the case of graph-isomorphism to the problem of subgraph-isomorphism.

1 Introduction

Graphs play an important role in structural pattern recognition. Besides comparing two given patterns, it is often required to match an input pattern against a database of known patterns or sub-patterns. If graphs are used to represent structural data, the task of matching patterns is turned into a problem of graph matching. Graph matching is used in a variety of applications, for example document processing [1], image analysis [2, 3], biometric identification [4] and video analysis [5]. Despite being a computationally expensive approach, graph matching is attractive for pattern recognition problems since graphs are a universal representation formalism. If databases of model graphs are used, an additional factor proportional to the size of the database is introduced in the overall complexity of the matching process. A variety of mechanisms have been proposed to reduce the complexity of graph matching when large databases are involved [6–10]. In this paper we propose an approach based on machine learning techniques.

The presented approach proposes to characterize graphs by features which can efficiently be extracted (e.g. the number of nodes or edges in a graph or

A. Fred et al. (Eds.): SSPR&SPR 2004, LNCS 3138, pp. 66–75, 2004.

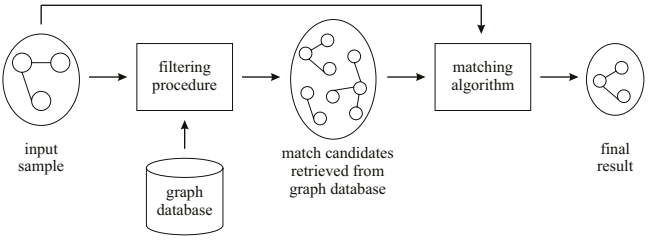

Fig. 1. Illustration of the filtering procedure in the database matching process.

the number of nodes or edges with a certain label). These features are used to perform a filtering on the database. A filtering procedure is a method which performs a quick and inexpensive reduction of the initial graph database size with respect to a given input graph. The aim of database filtering is to reduce the number of graph candidates in the database that need to undergo an expensive, full fledged graph matching process. A graphical illustration is shown in Figure 1.

The presented work extends previous studies on graph matching performance and graph database filtering (see [11, 12]). In contrast with the work reported in [11], which was restricted to the case of graph isomorphism, the present paper deals with the problem of subgraph-isomorphism. Database filtering in conjunction with subgraph-isomorphism search was also studied in [13]. However, the decision trees used in [13] were identical to the decision trees used in [11, 12] for graph-isomorphism, and the case of subgraph-isomorphism was dealt with by means of an extended decision tree traversal procedure. By contrast, a generalized decision tree induction procedure is proposed in the present paper. In the next section, we briefly introduce terminology and graph features used in this study. Then, we show how the concept of graph representation by means of vectors can be combined with the decision tree filtering approach. Experimental results will be presented in Section 4, and conclusions drawn in Section 5.

2 Terminology and Graph Features

In this work, structural data or patterns are represented as graphs. A graph is defined as a four-tuple $g = (V, E, \alpha, \beta)$, where V denotes a finite set of nodes, $E \subseteq V \times V$ is a finite set of edges, $\alpha : V \rightarrow L_V$ is a node labelling function, and $\beta : E \rightarrow L_E$ is an edge labelling function. L_V and L_E are finite or infinite sets of node and edge labels, respectively. In this work only directed graphs are considered. However, the same ideas can be applied to undirected graphs as well. A subgraph $g_s = (V_s, E_s, \alpha_s, \beta_s)$ of a graph g is a subset of its nodes and edges, such that $V_s \subseteq V$, $E_s = E \cap (V_s \times V_s)$, $\alpha_s(v) = \alpha(v)$ and $\beta_s(e) = \beta(e)$. Two graphs g and g' are isomorphic to each other if there exists a bijective mapping u from the nodes of g to the nodes of g', such that the structure of the edges as well as all node and edge labels are preserved under u. Similarly, an isomorphism between a graph g and a subgraph g'_s of a graph g' is called a subgraph-isomorphism from g to g'.

In this study, feature vectors are used to represent graphs. Since the vectors are used for the purpose of database filtering, the features must meet the following two requirements. First, their extraction from a sample graph must be fast (compared to performing a graph matching) and second, they should possess a high degree of saliency (they should be able to differentiate between as many graphs as possible). Fast extraction of the features from sample graphs directly influences the filtering performance whereas high saliency ensures a precise candidate retrieval (which means that fewer graphs will be subject to the full-fledged graph matching procedure after filtering). In this work, the following features are used:

1. the total number of vertices in the graph
2. the total number of vertices with a given label in the graph
3. the total number of incoming (outgoing) edges per vertex label in the graph
4. the total number of vertices with a given number of incoming (outgoing) edges in the graph
5. the total number of vertices with a given label and a given number of incoming (outgoing) edges in the graph

Previous studies have shown that the selection of these features is very effective for database filtering. In [11] it can be seen that when searching for graph-isomorphism candidates, these features are completely sufficient to distinguish between all graphs in the database. Hence in this study, the same features have been used.

In case of graph-isomorphism [11], a necessary condition for two graphs g and g' being isomorphic is that they have identical feature values. Hence, given the feature vector $f = (f_1, \ldots, f_m)$ extracted from g and $f' = (f'_1, \ldots, f'_m)$ extracted from g', g and g' can immediately be ruled out to be isomorphic if a feature j is discovered such that $f_j \neq f'_j$. However, for subgraph-isomorphism, which is considered in this paper, the relation is not that simple. Looking at the list of features presented above, we can see that there are two basic types of features:

- features not containing edge information (features 1 and 2)
- features containing edge information (features 3 to 5)

Extending the feature vector comparison from graph-isomorphism to subgraph-isomorphism is straightforward for features not containing edge information. In order for a graph g' possibly being isomorphic to a subgraph of a graph g, the value of such a feature f'_j must be smaller than, or equal to, the feature value f_j in g. Therefore, the relation $f'_j = f_j$ for graph-isomorphism needs to be replaced by $f'_j \leq f_j$ for subgraph-isomorphism. (Note that this is only a necessary, but not a sufficient condition for g' being a subgraph of g.) The difficulty for features 3 to 5 (features containing edge information) is that in subgraph-isomorphism, nodes of lower vertex degrees in the subgraph may be mapped onto nodes with a higher degree in the supergraph. (This is contrary to graph-isomorphism, where nodes can only be mapped onto nodes of equal degree.) A simple example would be a star-graph as the (original) graph and

the same graph with its center node removed as the subgraph. In the original graph, all nodes except the center node are of degree 1 (the center node is of degree $n-1$, where n is the number of nodes in the graph). In the subgraph, since the center node has been removed, the surrounding nodes are now of degree 0. Hence, it can happen that a subgraph has a larger number of nodes with a certain degree than its supergraph. In order to overcome this problem, the features used in our approach must be able to associate the nodes of degree 0 in the subgraph with the nodes of degree 1 in the original graph. In order to properly compare these feature values, one must not only consider the value for the current node's degree but also include the values for the nodes of a lower degree. This can be done by summing the feature vector's values according to node degree order. Applying this technique allows us to use the same features for both graph-isomorphism and subgraph-isomorphism (but the way the features are handled by the subgraph isomorphism filtering procedure is different from filtering in case of graph isomorphism).

3 Decision Trees

In this section, we will show how the concept of feature-vector comparison for graph-/subgraph-isomorphism can be used in combination with decision trees. The general idea is that in a preprocessing step a decision tree is built to classify graphs according to their feature vectors. Depending on the given matching task, two types of decision trees can be induced. In order to filter the database for graph-isomorphism candidates, a graph-isomorphism tree is used and analogously for subgraph-isomorphism filtering, a subgraph-isomorphism tree is induced. The tree induction algorithm ensures that out of the entire feature set only the most salient features are used for filtering. At runtime, all features needed are extracted from the sample graph and then, the decision tree is traversed to retrieve suitable graph candidates from the database. This approach minimizes the number of features to be tested in order to eliminate the maximum number of non-candidates from the database. The decision tree procedure itself is analogous to standard decision tree methods (see [14], for example) with the difference that, whereas ordinary decision tree methods try to generalize from a training set of objects, the approach presented here tries to 'overfit' the data in the sense that all leaf nodes in the tree include just a single graph. In general, the smaller the number of graphs in a leaf node is, the smaller is the number of full-fledged graph matchings that need to be computed.

In this section a brief explanation will be given on how to induce decision trees useful for subgraph-isomorphism filtering. For the purpose of completeness we also describe how to induce trees for graph-isomorphism. Then, we will focus on the problem of using the trees to create two filter types, one for graph-isomorphism and one for subgraph-isomorphism.

3.1 Graph-Isomorphism Decision Tree Induction

As mentioned in Section 2, a necessary condition for two graphs g and g' being isomorphic is that they have identical feature vectors. The decision tree induction

algorithm, which is derived from [14], classifies the graphs in the database based on their feature values. As an initializing step, the tree's root node is constructed and it is assigned the entire graph set of the database. Then, each available feature is tested and its suitability is evaluated according to a given split criterion (see [14]). Amongst all features, the best one is chosen and the current root's graph set is split into subsets according to the best feature. For each feature value, a son node is created and the node is assigned the subset of graphs that correspond to that feature value. The induction procedure is recursively continued with the son nodes until one of the following termination conditions holds: a) the graph set in a node contains only one graph; b) no features are left to divide a subset; c) the features left cannot distinguish the remaining graphs in the set. Cases b) and c) correspond to the situation where, later in the decision traversal phase, multiple graphs are returned by the filtering procedure, while case a) reflects the 'ideal' situation where only one candidate graph remains to undergo the full-fledged graph matching procedure.

3.2 Subgraph-Isomorphism Decision Tree Induction

Before extending the approach presented above to the problem of filtering for subgraph-isomorphism candidates, one must first define the search task to be considered. There are two possible search scenarios:

- supergraph-search: the input sample is considered to be isomorphic to subgraphs of the graphs in the database.
- subgraph-search: the database contains graphs possibly isomorphic to a subgraph of the input sample.

In the following explanations, we will focus on the second scenario where the input sample is a supergraph and the database graphs are subgraphs (subgraph-search). Note that the adaption of this traversal to the task of supergraph-search is straightforward.

The decision tree structure as presented in the graph-isomorphism case needs two major adaptations before it can be used for subgraph-filtering purposes. The first adaption concerns the assignment of graph-subsets to son nodes. In the isomorphism case, the father node's graph set is split into disjoint subsets according to the best feature's values. For subgraph-isomorphism trees however, these subsets are not disjoint anymore. Consider the case where a feature occurs n times in the input sample. In that case, all graphs in the database where the same feature occurs $n' < n$ times are possible subgraph-isomorphism candidates and need to be assigned to the son-node representing feature value n. As an additional consequence, the son-node with the maximal feature value at any level in the tree is assigned the entire graph set of its father (hence the depth of the decision tree is only limited by the number of features evaluated). Depending on the graph type, the number of possible features is in general quite large. In case a son node containing its father's graph set is unlikely to be reached during traversal, it should be induced after other nodes that are more likely to be

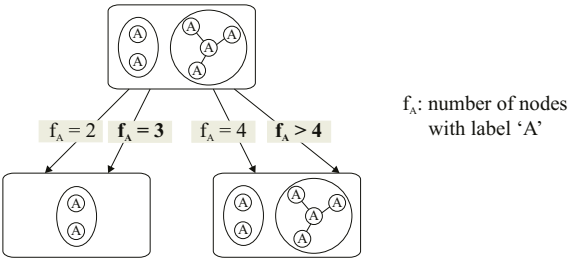

Fig. 2. Example of a decision tree split.

reached. Hence, while constructing the tree, the nodes ready for expansion must be ordered so that these candidates are not (or very late in the process) expanded. An intuitively plausible ordering is given by the probability with which a node is likely to be reached during tree traversal. This probability is given by the number of candidates only appearing in the node under consideration compared to the number of candidates in the father node.

The second adaption is caused by the fact that feature values not occurring in the graph database may occur in the input sample and must therefore be considered during tree construction. This can be done by introducing additional edges from the father to the appropriate son node (the node representing the next smaller feature value) for each non-occurring value of the considered feature.

Figure 2 illustrates the described modifications, namely overlapping graph sets in the son nodes as well as feature values not occurring in database graphs. In the father node, there are two graphs with two and four nodes with label 'A', respectively. Hence, the decision tree consists of two son nodes (and their corresponding edges), one for each feature value $f_A = 2$ and $f_A = 4$. Furthermore, since a sample graph may contain 3 nodes with label 'A', an additional edge with label $f_A = 3$ must be introduced, also pointing to the node for graphs with $f_A = 2$. Naturally, all samples where $f_A > 4$ must be directed to the node where $f_A = 4$ (branch to the very right) and all samples where $f_A < 2$ are not possible supergraphs to the graphs in the database (therefore no branch is provided in the tree for this case). Consider the case where, at runtime, the decision tree is traversed for a sample graph with $f_A = 5$. While traversing the tree, the leaf with $f_A > 4$ is reached. Hence, the input sample is a possible supergraph to both graphs in the illustration which makes sense since the sample consists of at least 5 nodes with label 'A'.

3.3 Decision Tree Traversal

Decision trees induced as described above can be used to retrieve possible graph-isomorphism or subgraph-isomorphism candidates of a given sample graph in the following way. First, the same features that were extracted from the database graphs and used to induce the decision tree are extracted from the input graph. Then, the traversal algorithm follows the tree branch whose feature values are

equal to the values extracted from the sample graph. There are only two possible outcomes of the decision tree traversal procedure. The first outcome is that a leaf node is reached. In this case, the graphs associated with the leaf node are possible matches to the input graph. Each of these graphs must then be tested against the input graph for (sub-)graph-isomorphism using a conventional algorithm as described in [15–17], for example. The second outcome is that no leaf node is reached in which cases there are no graphs in the database that can be (sub-)graph-isomorphic to the input graph.

4 Experimental Results

To demonstrate the efficiency of the approach, we tested it on several different types of graphs described below (see [18]).

- Random Graphs: The random graph database consists of connected graphs with different node and edge label alphabet size.
- Bounded Valence Graphs (regular, irregular): This databases consists of graphs with a fixed valence per node (regular bounded valence graphs) or a fixed valence over the entire graph (irregular bounded valence graphs).
- Meshes / Hyper-Cuboids (regular, irregular): The database consists of (Hyper-) Cuboids of varying dimension $n = 2$ (meshes), $n = 3$ (cuboids), $n = 4, 5$ (hypercuboids).

For each graph type and parameter setting a database of 1,000 graphs was created. During creation of the database, it was made sure that each graph was isomorphic only to itself.

The primary objective of the proposed filtering method is to reduce the number of candidates which have to undergo a full-fledged isomorphism or subgraph-isomorphism matching. Hence, the quality of the approach can be expressed by measuring the number of graphs that are assigned to the leaf nodes in the decision tree. (We will also refer to this number as the *cluster size* of the decision tree.) The cluster size determines the number of full-fledged matchings that need to be executed. The other important measure is the average number of nodes visited during traversal. This value directly affects traversal and therefore filter time.

Decision trees suitable for graph-isomorphism retrieval can be fully induced with no problems. Decision trees suitable for subgraph-isomorphism on the other hand can grow, due to their special structure, quite large and therefore need to be limited in size. To control the subgraph-isomorphism decision tree growth, the trees were limited in size to the same number of nodes as the corresponding graph-isomorphism trees (from hundreds up to several thousand nodes).

In order to measure the average cluster size, graphs were randomly picked from the database and were then used as an input sample. Before extracting the features, the sample graph's size was increased and the additional nodes were assigned labels not occurring in the database graphs (this was to ensure that no additional subgraph-isomorphisms were introduced). Then the feature vector

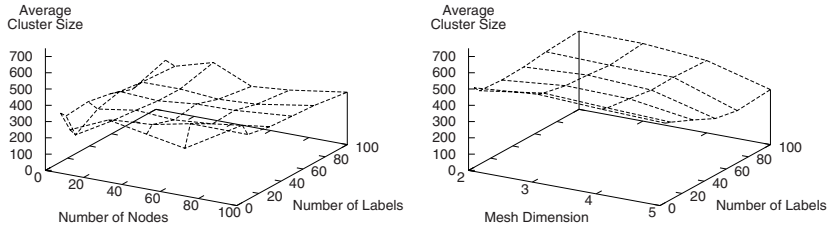

Fig. 3. Cluster size for random and mesh graphs.

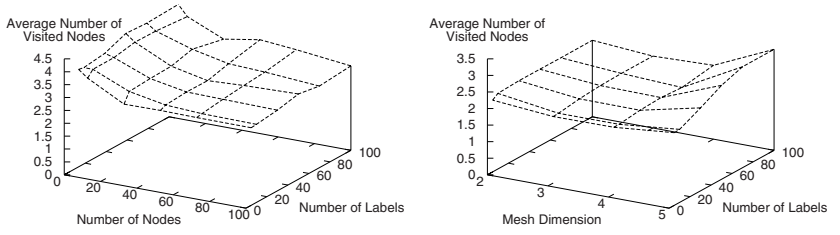

Fig. 4. Average number of visited nodes for random and mesh graphs.

was extracted and the decision tree traversed. In order to get an average value for the cluster size, the procedure was repeated 1,000 times for each database.

Figure 3 shows the cluster size for random and mesh graphs, respectively. The results for bounded valence graphs are similar to the values obtained for random graphs and therefore not depicted. In [11] it has been shown that for graph-isomorphism, the cluster size can be reduced down to only one graph remaining after filtering. For subgraph-isomorphism, naturally, the reduction factor is not that high. However, as is shown in Figure 3 the approach on average reduces the database size to about 300 graphs for random graphs. This means that the initial database size is reduced by about 70%. Due to the much more regular structure of mesh graphs, the filtering effect is not quite that high. However, still approximately 500 graphs can be eliminated from the database which is equal to a reduction factor of 50%.

The second parameter influencing the filter's performance is the average number of nodes visited during tree traversal. Figure 4 shows that for both graph types, random as well as mesh graphs, less than 4 nodes are visited during traversal (again the results obtained for bounded valence graphs are similar to the results of random graphs.) It can be concluded from Figure 4 that the size of the database can be very effectively reduced through a small number of tests. The computation time of these tests is in fact negligible.

5 Conclusions

In this paper an approach to graph database filtering using machine learning techniques has been presented. The method is based on a decision tree data structure. It reduces the number of candidate graphs in a database to be tested for

graph- and subgraph-isomorphism. Depending on the considered graph matching task (graph isomorphism or subgraph isomorphism), special types of decision trees are built that classify the graphs using given features. At runtime, possible matching candidates are retrieved from the database by traversing the decision tree. This paper presented two decision tree induction methods. One method is able to induce trees suitable for graph-isomorphism candidate retrieval, while the other is designed to induce trees suitable for subgraph-isomorphism candidate retrieval.

Considering the complexity of the proposed method we notice that the method is divided into two stages: a) tree induction and b) tree traversal. Tree induction is considered to be an off-line step, hence its complexity is not of primary interest. Tree traversal, however, is the main objective concerning complexity. It is only dependent upon the number of visited nodes during tree traversal. The overall complexity of filtering-based graph matching is determined by the decision tree traversal complexity, the number of final matchings to be performed (this number is identical to the cluster size associated with the leaf nodes in the decision tree) and the complexity of computing the final matchings, which is depending upon graph type and graph matching algorithm. In this paper, a number of experiments investigating cluster size and number of visited nodes have been conducted. The results indicate that the cluster size can be significantly reduced by few tests, resulting in a small number of nodes to be visited as well as a small number of final matchings to be performed.

The main contribution of the present paper is an extension of the method proposed in [11] from graph-isomorphism to the problem of subgraph-isomorphism by generalization of the decision tree's structure. Future work is planned extending the approach to error tolerant, edit distance based graph matching methods and determining its efficiency on real world data.

Acknowledgment

This research was supported by the Swiss National Science Foundation (Nr. 2100-066700). The authors thank the Foundation for the support.

References

1. Llados, J., Marti, E., Villanueva, J.: Symbol recognition by error-tolerant subgraph matching between region adjacency graphs. In: IEEE Transactions on Pattern Analysis and Machine Intelligence. Volume 23 – 10. (2001) 1137 – 1143
2. Torsella, A., Hancock, E.: Learning stuctural variations in shock trees. In: Proc. of the Joint IAPR International Workshops SSPR and SPR. (2002) 113 – 122
3. Luo, B., Hancock, E.: Structural graph matching using the em algorithm and singular value decomposition. In: IEEE Transactions on Pattern Analysis and Machine Intelligence. Volume 23 – 10. (2001) 1120 – 1136
4. Lumini, A., Maio, D., Maltoni, D.: Inexact graph matching for fingerprint classification. Machine Graphics and Vision, Special Issue on Graph Transformations in Pattern Generation and CAD **8** (1999) 231 – 248

5. Chen, H., Lin, H., Liu, T.: Multi-object tracking using dynamical graph matching. In: Proc. of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (2001) 210 – 217
6. Guigno, R., Sasha, D.: Graphgrep: A fast and universal method for querying graphs. In: Proceeding of the International Conference in Pattern Recognition (ICPR). (2002) 112 – 115
7. Wang, J.l., Sasha, D., Guigno, R.: Algorithmics and applications of tree and graph searching. In: Proceeding of the ACM Symposium on Principles of Database Systems (PODS). (2002)
8. Messmer, B., Bunke, H.: A new algorithm for error–tolerant subgraph isomorphism detection. In: IEEE Trans. Pattern Analysis and Machine Intelligence. Volume 20. (1998) 493 – 505
9. Shapiro, L., Haralick, R.: Organization of relational models for scene analysis. In: IEEE Trans. Pattern Analysis and Machine Intelligence. Volume 3. (1982) 595 – 602
10. Sengupta, K., Boyer, K.: Organizing large structural modelbases. In: IEEE Trans. Pattern Analysis and Machine Intelligence. Volume 17. (1995)
11. Irniger, C., Bunke, H.: Graph matching: Filtering large databases of graphs using decision trees. In Jolion, J.M., Kropatsch, W., Vento, M., eds.: Graph-based Representations in Pattern Recognition, Cuen (2001) 239 – 249
12. Irniger, C., Bunke, H.: Theoretical analysis and experimental comparison of graph matching algorithms for database filtering. In Hancock, E., Vento, M., eds.: Graph-based Representations in Pattern Recognition, Springer Verlag Berlin Heidelberg (2003) 118 – 129
13. Irniger, C., Bunke, H.: Graph database filtering using decision trees. In: Proceedings of the International Conference in Pattern Recognition (ICPR). (2004)
14. Quinlan, J.: C4.5: Programs for Machine Learning. Document Analysis Systems II. Morgan Kaufmann Publishers (1993)
15. Ullmann, J.: An algorithm for subgraph isomorphism. In: JACM. Volume 23. (1976) 31 – 42
16. Cordella, L., Foggia, P., Sansone, C., Vento, M.: An improved algorithm for matching large graphs. In Jolion, J.M., Kropatsch, W., Vento, M., eds.: Graph-based Representations in Pattern Recognition, Cuen (2001) 149 – 159
17. McKay, B.: Practical graph isomorphism. In: Congressus Numerantium. Volume 30. (1981) 45 – 87
18. Foggia, P., Sansone, C., Vento, M.: A database of graphs for isomorphism and subgraph isomorphism. In Jolion, J.M., Kropatsch, W., Vento, M., eds.: Graph-based Representations in Pattern Recognition, Cuen (2001) 176 – 188