# Fast Error-correcting Graph Isomorphism Based on Model Precompilation

B.T. Messmer and H. Bunke
Institut für Informatik und angewandte Mathematik,
Universität Bern, Neubrückstr. 10, Bern, Schweiz

IAM-96-012

September 1996

# Fast Error-correcting Graph Isomorphism Based on Model Precompilation

B.T. Messmer and H. Bunke
Institut für Informatik und angewandte Mathematik,
Universität Bern, Neubrückstr. 10, Bern, Schweiz

September 1996

**Abstract**

In this paper we present a fast algorithm for the computation of error-correcting graph isomorphisms. The new algorithm is an extension of a method for exact subgraph isomorphism detection from an input graph to a set of a priori known model graphs, which was previously developed by the authors. Similarly to the original algorithm, the new method is based on the idea of creating a decision tree from the model graphs. This decision tree is compiled off-line in a preprocessing step. At run time, it is used to find all error-correcting graph isomorphisms from an input graph to any of the model graphs up to a certain degree of distortion. The main advantage of the new algorithm is that error-correcting graph isomorphism detection is guaranteed to require time that is only polynomial in terms of the size of the input graph. Furthermore, the time complexity is completely independent of the number of model graphs and the number of edges in each model graph. However, the size of the decision tree is exponential in the size of the model graphs. Nevertheless, practical experiments have indicated that the method can be applied to graphs containing up to 16 vertices.

**CR Categories and Subject Descriptors:** G2.2 [Discrete Mathematics]: Graph Algorithms; I.5 Pattern Recognition; I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods and Search.

**General Terms:** Algorithms

**Additional Key Words:** subgraph Isomorphism, Indexing

# Fast Error-correcting Graph Isomorphism Based on Model Precompilation

B.T. Messmer and H. Bunke
Institut für Informatik und angewandte Mathematik,
Universität Bern, Neubrückstr. 10, Bern, Schweiz

September 1996

**Abstract**

In this paper we present a fast algorithm for the computation of error-correcting graph isomorphisms. The new algorithm is an extension of a method for exact subgraph isomorphism detection from an input graph to a set of a priori known model graphs, which was previously developed by the authors. Similarly to the original algorithm, the new method is based on the idea of creating a decision tree from the model graphs. This decision tree is compiled off-line in a preprocessing step. At run time, it is used to find all error-correcting graph isomorphisms from an input graph to any of the model graphs up to a certain degree of distortion. The main advantage of the new algorithm is that error-correcting graph isomorphism detection is guaranteed to require time that is only polynomial in terms of the size of the input graph. Furthermore, the time complexity is completely independent of the number of model graphs and the number of edges in each model graph. However, the size of the decision tree is exponential in the size of the model graphs. Nevertheless, practical experiments have indicated that the method can be applied to graphs containing up to 16 vertices.

## 1 Introduction

Graph structures is a powerful and universal tool with applications in various subfields of science and engineering. In pattern recognition and image analysis, graphs are often used for the representation of structured objects. For example, if the problem is to recognize instances of known objects in an image, then often models of the known objects are represented by means of model graphs and stored in a database. The unknown objects in the input image are extracted by means of suitable preprocessing and segmentation algorithms, and represented by input graphs. Thus the problem of object recognition can be solved by searching for graph or subgraph isomorphisms between the models and the input graph. In a real world application, however, there is usually a certain amount of noise and distortion present in an input graph. Therefore, perfect correspondences between the models and the input do frequently not exist. Hence, it is necessary to provide means for error-tolerant matching. In the past, different methods for finding exact and error-tolerant graph and subgraph isomorphisms have been proposed, such as heuristic search[BA83, SH81, Ull76], probabilistic relaxation[KCP92], simulated annealing[HHVN90] or linear programming[AD93]. One of the major problems of error-correcting graph or subgraph isomorphism detection is its exponential time complexity, which is due to the fact that the problem is NP-complete. Combinatorial search methods such as $A^*$ are guaranteed to always find the optimal solution. However, they require exponential time in the worst case. Stochastic optimization methods such as relaxation or simulated annealing, on the other hand, have only polynomial time complexity, but they are not guaranteed to always yield the correct solution. Another problem with graph matching arises if the number of models in the database is large. In case of many models, it may become impossible to sequentially match each model against the input graph.

In this paper, we propose a method which is capable of finding all error-correcting graph isomorphisms between an input and a set of model graphs in time that is only polynomial in the number of vertices of the input graph. In particular, the time complexity of the new method is completely independent of the number of model graphs in the database. The new algorithm is an extension of the method for exact subgraph isomorphism detection that was previously presented by the authors in [MB95b, MB95c]. It is based on the idea of generating all possible adjacency matrices of a model graph off-line and organizing them in a decision tree. At run time, the decision tree is used to classify the adjacency matrix of an unknown input graph. In the case of exact graph isomorphism detection, the classification process can be done in quadratic time. In this paper, we describe two approaches to the incorporation of error correction in the decision tree method.

In the first approach, error correction is considered at the time of the creation of the decision tree. That is, for each model graph a set of distorted copies are created and compiled into the decision tree. The number of distorted copies depends on the maximal admissible error. At run time, the decision tree is used to classify the unknown input graph in the same way as in case of exact subgraph isomorphism detection. The time complexity of this procedure is only quadratic. However, the size of the decision tree is exponential in the number of vertices of the model graphs and in the degree of distortion that is to be considered.

In the second approach, the error corrections are considered at run time only. That is, the decision tree for a set of model graphs does not incorporate any information about possible errors. Hence, the decision tree compilation step is identical to the original preprocessing step presented in [MB95b, MB95c] and, consequently, the size of the decision tree is exponential only in the size of the model graphs. At run time, a set of distorted copies of the input graph are constructed such that all possible error corrections up to a certain error threshold are considered. Each graph in this set is then classified by the decision tree.

The remainder of this paper is organized as follows. In Section 2, the concepts of graph isomorphism and error-correction are formally defined and some notations are introduced. Next, in Section 3, the basic idea of the decision tree approach for exact graph isomorphism detection is outlined. The application of the decision tree approach to error-correcting graph isomorphism and the computational complexity of the new method are described in Section 4. In Section 5, the results of a number of experiments with randomly generated graphs are presented and, finally, in Section 6, some conclusion are drawn.

## 2 Definitions and Notations

**Definition 1:** A *labeled graph* $G$ is a 4-tuple, $G = (V, E, \mu, \nu)$, where

- $V$ is the set of vertices,
- $E \subseteq V \times V$ is the set of edges,
- $\mu : V \rightarrow L_V$ is a function assigning labels to the vertices,
- $\nu : E \rightarrow L_E$ is a function assigning labels to the edges.

$\square$

We assume that $L_V$ and $L_E$ are finite sets of symbolic labels. Note that the above definition corresponds to the case of directed graphs. Undirected graphs are obtained if we require for each edge $(v_1, v_2)$ the existence of an edge $(v_2, v_1)$ in the opposite direction with the same label.

Let $G = (V, E, \mu, \nu)$ be a graph with $V = \{v_1, v_2, \ldots, v_n\}$. Then $G$ can also be represented by its adjacency matrix $M = (m_{ij}), i, j = 1, \ldots, n$, where $m_{ii} = \mu(v_i)$ and $m_{ij} = \nu((v_i, v_j))$ for $i \neq j$. Apparently, the adjacency matrix representation of a graph doesn't take loops at a vertex into account. However, this isn't a real restriction as loops can be represented by means of an extended set of vertex labels.

Clearly, the matrix $M$ is not unique for a graph $G$. If $M$ represents $G$, then any permutation of $M$ is also a valid representation of $G$.

**Definition 2:** A $n \times n$-matrix $P = (p_{ij})$ is called a *permutation matrix* if

1. $p_{ij} \in \{0, 1\}$ for $i, j = 1, \ldots, n$
2. $\sum_{i=1}^{n} p_{ij} = 1$ for $j = 1, \ldots, n$
3. $\sum_{j=1}^{n} p_{ij} = 1$ for $i = 1, \ldots, n$

$\square$

If a graph $G$ is represented by an $n \times n$ adjacency matrix $M$ and $P$ is an $n \times n$ permutation matrix, then the $n \times n$ matrix

$$M' = PMP^T \tag{1}$$

where $P^T$ denotes the transpose of $P$, is also an adjacency matrix of $G$. If $p_{ij} = 1$ then the $j$-th vertex in $M$ becomes the $i$-th vertex in $M'$.

**Definition 3:** Let $G_1$ and $G_2$ be two graphs and $M_1$ and $M_2$ their corresponding adjacency matrices. $G_1$ and $G_2$ are *isomorphic* if there exists a permutation matrix $P$ such that

$$M_2 = PM_1P^T \tag{2}$$

□

Notice that the matrix $P$ can be understood as a bijective function $f$ that maps the vertices of $G_1$ to $G_2$, and vice versa. That is, $f(v_j) = v_i$ iff $p_{ij} = 1$. We will call both $P$ and $f$ a *graph isomorphism* between $G_1$ and $G_2$. Thus, the problem of finding a graph isomorphism between two graphs $G_1$ and $G_2$ is equivalent to finding a permutation matrix $P$ for which Eq.(2) holds true.

In order to integrate the concept of error correction into graph matching, we define a distance measure for graphs. Similarly to the string matching problem where edit operations are used to define the string edit distance[WF74], we define a graph edit distance, which is based on the idea of correcting distortions in an input graph by means of edit operations [BA83]. The graph edit operations are used to modify either the model or the input graph until there exists a graph isomorphism between the model and the input. In order to model the fact that certain distortions, i. e. edit operations, are more likely than others, each graph edit operations $\delta$ is assigned a cost $C(\delta) \geq 0$. The costs of the graph edit operations are application dependent and must be defined on the basis of heuristic knowledge. The graph distance from a model to an input graph is then defined to be the minimum cost taken over all sequences of edit operations that are necessary for the correction of the distortions in the input graph. It can be concluded that the smaller the edit distance between a model and an input graph is, the more similar they are. In this paper, we consider the following distortions in a graph: vertex and edge label substitution, and missing and extraneous edges. For each type of distortion, a corresponding graph edit operation is defined.

**Definition 4:** Given a graph $G = (V, E, \mu, \nu)$, a *graph edit operation* $\delta$ on $G$ is any of the following:

- $\mu(v) \to l$, $v \in V$, $l \in L_V$: substituting the label $\mu(v)$ of vertex $v$ by $l$ (for the correction of vertex label distortions).
- $\nu(e) \to l'$, $e \in E$, $l' \in L_E$: substituting the label $\nu(e)$ of edge $e$ by $l'$ (for the correction of edge label distortions).
- $e \to \$$, $e \in E$: deleting the edge $e$ from $G$ (for the correction of missing edges).

- $\$ \to e = (v_1, v_2),\ v_1, v_2 \in V, (v_1, v_2) \notin E$: inserting an edge between two existing vertices $v_1$,$v_2$ of $G$ (for the correction of extraneous edges).

□

The four edit operations in Def. 4 are powerful enough to transform any graph $G$ into any other graph $G'$, provided that $G$ and $G'$ have an equal number of vertices. It is possible, of course, to extend this set of edit operations by including vertex insertions and deletions, too. Then any graph $G$ can be transformed into any other graph $G'$, even if $G$ and $G'$ have a different number of vertices.

**Definition 5:** Given a graph $G = (V, E, \mu, \nu)$ and an edit operation $\delta$, the edited graph, $\delta(G)$, is a graph $\delta(G) = (V_\delta, E_\delta, \mu_\delta, \nu_\delta)$ with

1. $V_\delta = V$
2. $E_\delta = \begin{cases} E \cup \{e\} & \text{if } \delta = (\$ \to e) \\ E - \{e\} & \text{if } \delta = (e \to \$) \\ E & \text{otherwise} \end{cases}$
3. $\mu_\delta(v) = \begin{cases} l & \text{if } \delta = (\mu(v) \to l) \\ \mu(v) & \text{otherwise} \end{cases}$
4. $\nu_\delta(e) = \begin{cases} l' & \text{if } \delta = (\nu(e) \to l') \\ \nu(e) & \text{otherwise} \end{cases}$

□

**Definition 6:** Given a graph $G = (V, E, \mu, \nu)$ and a sequence of edit operations $\Delta = (\delta_1, \delta_2, \ldots, \delta_k),\quad k \geq 1$, the edited graph, $\Delta(G)$, is a graph $\Delta(G) = \delta_k(\ldots \delta_2(\delta_1(G))\ldots)$. The total cost of the transformation of $G$ into $\Delta(G)$ is given by $C(\Delta) = \sum_{i=1}^{k} C(\delta_i)$. □

**Definition 7:** Given two graphs $G$ and $G'$, an *error-correcting (ec) graph isomorphism* from $G$ to $G'$ is a 2-tuple $(\Delta, P)$ where $\Delta$ is a sequence of edit operations and $P$ is a graph isomorphism from $\Delta(G)$ to $G'$. The cost of an *ec* graph isomorphism $(\Delta, P)$ is the cost $C(\Delta)$. □

It follows from this definition that, if there is an *ec* graph isomorphism $(\Delta, P)$ from a graph $G$ to a graph $G'$, then $G' = PM_{\Delta(G)}P'$ where $M_{\Delta(G)}$ is the adjacency matrix of the graph $\Delta(G)$. It is also easy to see that the permutation matrix $P$ is implied by $G$, $G'$ and $\Delta$.

Usually, there is more than one sequence $\Delta$ of edit operations such that a graph isomorphism from $\Delta(G)$ to $G'$ exists. Consequently, there is usually more than one *ec* graph isomorphism from $G$ to $G'$. For our graph distance measure, we are particularly interested in the *ec* graph isomorphism with minimum cost.

**Definition 8:** Let $G$ and $G'$ be two graphs. The *graph distance* from $G$ to $G'$, $d(G, G')$, is given by the minimum cost taken over all *ec* graph isomorphisms from $G$ to $G'$:

$$d(G, G') = MIN_{\Delta}\{C(\Delta) \mid (\Delta, P) \text{ is an } ec \text{ graph isomorphism from } G \text{ to } G'\}$$

□

The *ec* subgraph isomorphism $(\Delta, P)$ associated with $d(G, G')$ is called the *optimal* error-correcting (*oec*) graph isomorphism from $G$ to $G'$.(Notice that the permutation matrix $P$ is implied for given $G$, $G'$ and $\Delta$.) It is easy to see that the graph distance according to Def. 8 is in general not symmetric, i.e., $d(G, G') \neq d(G', G)$. However, in the rest of this paper, we assume that the costs for substituting labels and for inserting and deleting edges are symmetric, i.e., $C(l_1 \to l_2) = C(l_2 \to l_1)$ for all labels $l_1, l_2$ and $C(e \to \$) = C(\$ \to e)$ for all edges $e$[1]. It is easy to see that this assumption guarantees the symmetry of the graph distance according to Def. 8.

## 3 Graph Isomorphism by Decision Tree - A Brief Overview

For the purpose of self-containedness, the approach presented in [MB95b, MB95c] is briefly reviewed in this section. It works for both graph and subgraph isomorphism, but in the context of the present paper we consider only graph isomorphism. We assume that there is a set of model graphs that are known a priori, while the input graph becomes accessible at run time only. For each model graph we compute, in an off-line step, all possible permutations of the adjacency matrix and transform these adjacency matrices into a decision tree. At run time, the matrix of the input graph is used to find those adjacency matrices in the decision tree that are identical to it. The permutation matrices that correspond to these adjacency matrices represent the graph isomorphisms that we are looking for.

Let $G = (V, E, \mu, \nu)$ be a model graph and $M$ its corresponding $n \times n$-adjacency matrix. Furthermore, let $A(G)$ denote the set of all permuted adjacency matrices of $G$, i.e.,

$$A(G) = \{M_P | M_P = PMP^T \text{ where } P \text{ is an } n \times n \text{ permutation matrix}\} \quad (3)$$

The graph isomorphism problem can now be restated in terms of the sets introduced above. For a model graph $G$ with $n \times n$-adjacency matrix $M$ and an input graph

[1]This assumption is not essential, but it simplifies the description of the proposed method.

$G_I$ with $n \times n$-adjacency matrix $M_I$, we determine whether there exists a matrix $M_P \in A(G)$ such that $M_I = M_P$. If such a matrix $M_P$ exists, the permutation matrix $P$ associated with $M_P$ describes a graph isomorphism from $G_I$ to $G$, i.e. $M_I = M_P = PMP^T$.

In [MB95b, MB95c] it was proposed to organize the set $A(G)$ in a decision tree such that each matrix in $A(G)$ is classified by the tree. The features that will be used for the classification process are the individual elements in the adjacency matrices. We say that an $n \times n$ matrix consists of an array of so-called *row-column* elements $a_i$, where each $a_i$ is a vector of the form $a_i = (m_{1i}, m_{2i}, \ldots, m_{ii}, m_{i(i-1)}, \ldots, m_{i1})$. The matrix can then be written as $M = (a_1, a_2, \ldots, a_n)$; $i = 1, \ldots, n$. The decision tree is built according to the row-column elements of each adjacency matrix $M_P \in A(G)$. At the top of the decision tree there is a single root node. The direct successor nodes of the root node constitute the first level of the decision tree. On the first level, the classification of the matrices in $A(G)$ is done according to the first row-column element $a_1$ of each matrix $M_P \in A(G)$. The element $a_1 = (m_{11})$ represents the label of the first vertex in each matrix in $A(G)$, $m_{11} \in L_V$ (see Section 2). Each branch that leads to a direct successor node of the root is associated with a specific value of the row-column element $a_1$. Consequently, the matrices in $A(G)$ are classified at the first level of the decision tree according to their first vertex label. Next, on the second level of the decision tree, the second row-column element $a_2$ of each matrix is used for the classification. In general, the matrices that are represented by some node on the level $k$ are divided into classes according to the element $a_k$. At the bottom of the decision tree, there are the leaf nodes. Each leaf node represents a class of identical matrices $M_P \in A(G)$.

In Fig. 1, a graph, $g_1$, and its corresponding decision tree are shown. The nodes of the decision tree are represented by shaded circles. Each directed branch from one node to another has a row-column element associated with it. At the top of Fig. 1 the set $A(g_1)$ of permuted adjacency matrices of $g_1$ is listed. Eeach leaf node in the decision tree represents exactly one adjacency matrix.

Until now the construction of a decision tree for a single model graph has been explained. However, it is easy to see that different model graphs can be integrated into one decision tree. The classification of an adjacency matrix by the decision tree is independent of the model graph to which the adjacency matrix corresponds. Consequently, a set of different model graphs $G_1, \ldots, G_L$ and their corresponding sets of adjacency matrices $A(G_1), \ldots, A(G_L)$ can be represented by the same decision tree.

In Fig. 2 the decision tree for the graph $g_1$ in Fig. 1 and another graph, $g_2$, is displayed. The adjacency matrices of $A(g_2)$ are given at the top of Fig. 2. Only two nodes have to be added to the decision tree shown in Fig. 1. (Notice that for $g_2$ some of the permuted adjacency matrices are identical with each other.)

At run time, the decision tree is used in order to classify the $n \times n$ adjacency matrix $M_I$ of an unknown input graph $G_I$. The matrix $M_I$ is classified on the first

$$A\;\begin{array}{|c|c|c|}\multicolumn{1}{c}{1}&\multicolumn{1}{c}{2}&\multicolumn{1}{c}{3}\\\hline b&1&1\\\hline 0&a&1\\\hline 0&0&a\\\hline\end{array}\quad B\;\begin{array}{|c|c|c|}\multicolumn{1}{c}{1}&\multicolumn{1}{c}{3}&\multicolumn{1}{c}{2}\\\hline b&1&1\\\hline 0&a&0\\\hline 0&1&a\\\hline\end{array}\quad C\;\begin{array}{|c|c|c|}\multicolumn{1}{c}{2}&\multicolumn{1}{c}{1}&\multicolumn{1}{c}{3}\\\hline a&0&1\\\hline 1&b&1\\\hline 0&0&a\\\hline\end{array}\quad D\;\begin{array}{|c|c|c|}\multicolumn{1}{c}{2}&\multicolumn{1}{c}{3}&\multicolumn{1}{c}{1}\\\hline a&1&0\\\hline 0&a&0\\\hline 1&1&b\\\hline\end{array}\quad E\;\begin{array}{|c|c|c|}\multicolumn{1}{c}{3}&\multicolumn{1}{c}{1}&\multicolumn{1}{c}{2}\\\hline a&0&0\\\hline 1&b&1\\\hline 1&0&a\\\hline\end{array}\quad F\;\begin{array}{|c|c|c|}\multicolumn{1}{c}{3}&\multicolumn{1}{c}{2}&\multicolumn{1}{c}{1}\\\hline a&0&0\\\hline 1&a&0\\\hline 1&1&b\\\hline\end{array}$$

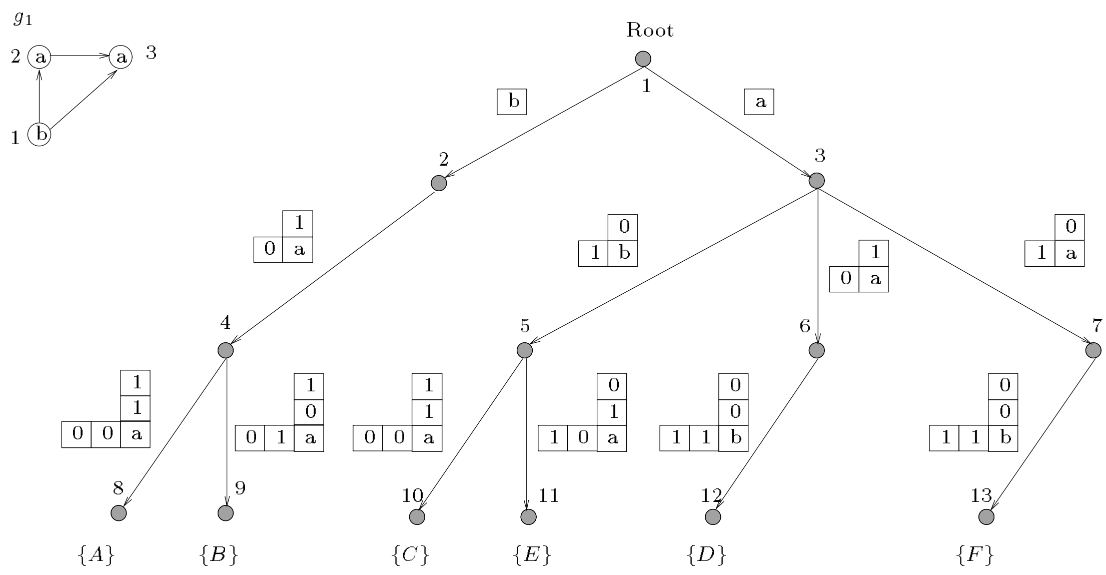

Figure 1: Decision tree for the classification of the adjacency matrices $A, \ldots, F$ of the graph $g_1$.

level according to its row-column element $a_{1_I}$. If there is some branch from the root node to a successor node whose associated element matches $a_{1_I}$, the algorithm continues with that successor node on the second level, and so on. If at some point no classification is possible, then the input graph $G_I$ is not isomorphic to any of the model graphs. If a leaf node has been reached then a graph isomorphism between the input graph and one of the models has been found. The permutation matrix $P$ corresponding to the leaf node represents the graph isomorphism between $G$ and $G_I$, i.e. $PMP^T = M_I$. For a more detailed description of this algorithm see [MB95b, MB95c].

The most important aspect of decision tree based graph isomorphism detection is that no backtracking is necessary at any point in the decision tree traversal. On each level $k$, the process of deciding which successor node to follow can be performed in $O(k)$ steps. This is due to the fact that the row-column elements that are associated to the successor nodes of a node $N$ can be stored in a dictionary. This dictionary is organized as a $2k-1$ index structure in which each element $m_{ij}$ of a row-column element $a_k = (m_{1k}, \ldots, m_{kk}, \ldots, m_{k1})$ is used as an index. Consequently, at run time, the dictionary which is attached to the node $N$, is used

$$A' \begin{array}{|c|c|c|} \multicolumn{1}{c}{1} & \multicolumn{1}{c}{2} & \multicolumn{1}{c}{3} \\ \hline a & 1 & 0 \\ \hline 0 & a & 1 \\ \hline 1 & 0 & a \\ \hline \end{array} \quad B' \begin{array}{|c|c|c|} \multicolumn{1}{c}{1} & \multicolumn{1}{c}{3} & \multicolumn{1}{c}{2} \\ \hline a & 0 & 1 \\ \hline 1 & a & 0 \\ \hline 0 & 1 & a \\ \hline \end{array} \quad C' \begin{array}{|c|c|c|} \multicolumn{1}{c}{2} & \multicolumn{1}{c}{1} & \multicolumn{1}{c}{3} \\ \hline a & 0 & 1 \\ \hline 1 & a & 0 \\ \hline 0 & 1 & a \\ \hline \end{array} \quad D' \begin{array}{|c|c|c|} \multicolumn{1}{c}{2} & \multicolumn{1}{c}{3} & \multicolumn{1}{c}{1} \\ \hline a & 1 & 0 \\ \hline 0 & a & 1 \\ \hline 1 & 0 & a \\ \hline \end{array} \quad E' \begin{array}{|c|c|c|} \multicolumn{1}{c}{3} & \multicolumn{1}{c}{1} & \multicolumn{1}{c}{2} \\ \hline a & 1 & 0 \\ \hline 0 & a & 1 \\ \hline 1 & 0 & a \\ \hline \end{array} \quad F' \begin{array}{|c|c|c|} \multicolumn{1}{c}{3} & \multicolumn{1}{c}{2} & \multicolumn{1}{c}{1} \\ \hline a & 0 & 1 \\ \hline 1 & a & 0 \\ \hline 0 & 1 & a \\ \hline \end{array}$$

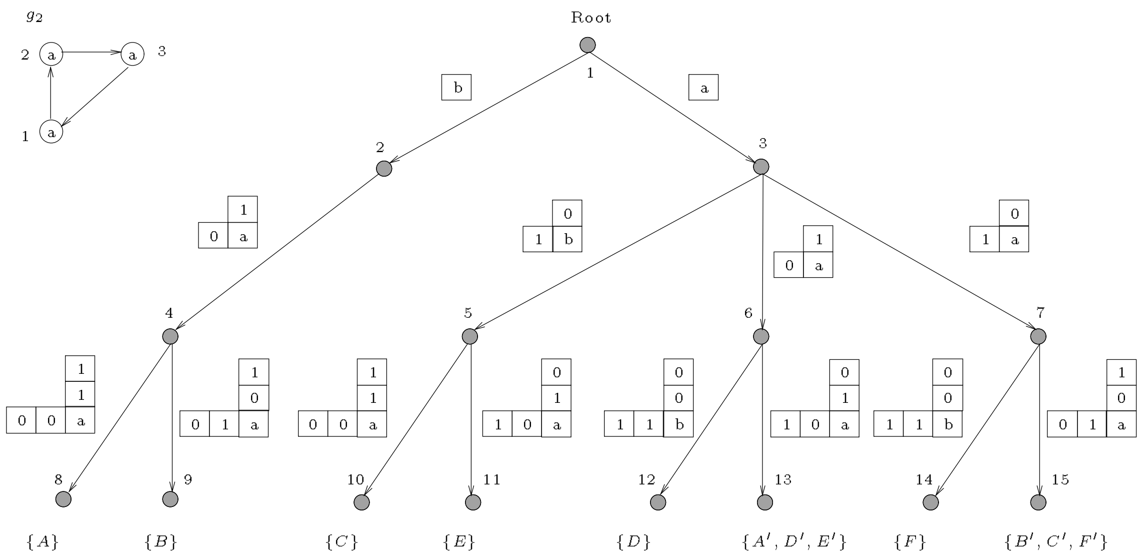

Figure 2: Decision tree for the graph $g_1$ in Fig 1 and another graph $g_2$.

to look up the existence of a row-column element in $O(k)$ steps. In particular, the computation of the next successor node is independent of the number of successor nodes and therefore independent of the number of model graphs that are represented by the decision tree. We can conclude that given an input graph with $n$ vertices and a decision tree representing a set of model graphs $G_1, \ldots, G_L$, the theoretical run time complexity of the new algorithm is $O(n^2)$ as there are $O(n)$ levels in the decision tree and on each of these levels $O(n)$ steps for choosing the next successor node must be performed. Notice that there is no best or worst case for the new algorithm. Independent of the input, its computation time is always quadratic, while for conventional algorithms such as Ullman's algorithm the computation time becomes exponential in the worst case.

The disadvantage of this method is the size of the decision tree. It is easy to see that for a graph with $n$ vertices, there are $n!$ different permuted adjacency matrices and, in the worst case, $n!$ leaf nodes in the corresponding decision tree. Hence, the size of the decision tree is bounded by $O(n^n)$ for a single model graph and by $O(Ln^n)$ for $L$ model graphs. In [MB95c] various pruning techniques have been described which reduce the size of the decision tree to $O(L3^n)$. It has been shown in a number of practical experiments documented in [MB95b, MB95c] that the decision tree approach to exact graph and subgraph isomorphism detection is applicable to single graphs with up to 22 vertices or, alternatively, to databases consisting of 100 graphs, each comprising up to 11 vertices. In the following section, we propose an extension of the decision tree approach to error-correcting graph isomorphism detection.

## 4 Error-correcting Graph Isomorphism by Decision Tree

Given a set of model graphs $G_1, \dots, G_L$ and an input graph $G_I$ we want to find the *oec* graph isomorphism $(\Delta^i, P^i)$ (see Def. 8) between $G_i$ and $G_I$ such that the cost $C(\Delta^i)$ is minimal over all model graphs, i.e. $C(\Delta^i) = \min\{C(\Delta^j); j = 1, \dots, L\}$. Traditionally, this problem is solved by applying an $A^*$-based algorithm to each model-input graph pair [SH81, SF83, Won90, CYS+96]. In such an algorithm, the vertices of the model graph are tentatively mapped to the vertices of the input graph one after the other, and for each such mapping the necessary edit operations are recorded and the corresponding edit costs are accumulated. At each stage of the algorithm, it is always the mapping with the least cost that is further expanded by mapping an additional vertex of the model graph onto vertices of the input graph. In this manner, it is guaranteed that for each model-input pair the *oec* graph isomorphism is found. There are two major disadvantages of this method. First, the number of expanded mappings may grow exponentially, resulting in an exponential time complexity in the worst case. Secondly, the method must be applied individually to each model-input pair. Consequently, the time complexity is also linearly dependent on the number of model graphs. In the case of exact graph isomorphism detection, both of these disadvantages can be avoided by using the decision tree approach described in the previous section.

The basic idea of the decision tree approach to error-correcting graph isomorphism detection is to separate the graph isomorphism search from the error-correction process. That is, given two graphs $G$ and $G'$, we propose to generate all distorted copies of $G$ such that the graph distance from each copy to $G$ is not larger than a certain threshold $\vartheta$. Each of the distorted copies of $G$ is then separately matched with the graph $G'$. Clearly, if the graph distance of $G$ and $G'$ is not larger than $\vartheta$, there exists a distorted copy of $G$ in the generated set that is isomorphic to $G'$. Hence, this copy will be detected by the exact graph isomorphism process. Formally, let

$$D(G, \vartheta) = \{\Delta(G) | \Delta \text{ is a sequence of edit operations with } C(\Delta) \leq \vartheta\} \qquad (4)$$

denote the set of all edited copies of $G$ with cost less than or equal to $\vartheta$. Clearly, for any graph $G'$, if $d(G, G') \leq \vartheta$, then $G' \in D(G, \vartheta)$. An example is shown in Fig. 3. A graph $g_1$ and the set $D(g_1, \vartheta)$ for $\vartheta = 1$ are displayed. As there are no labels the

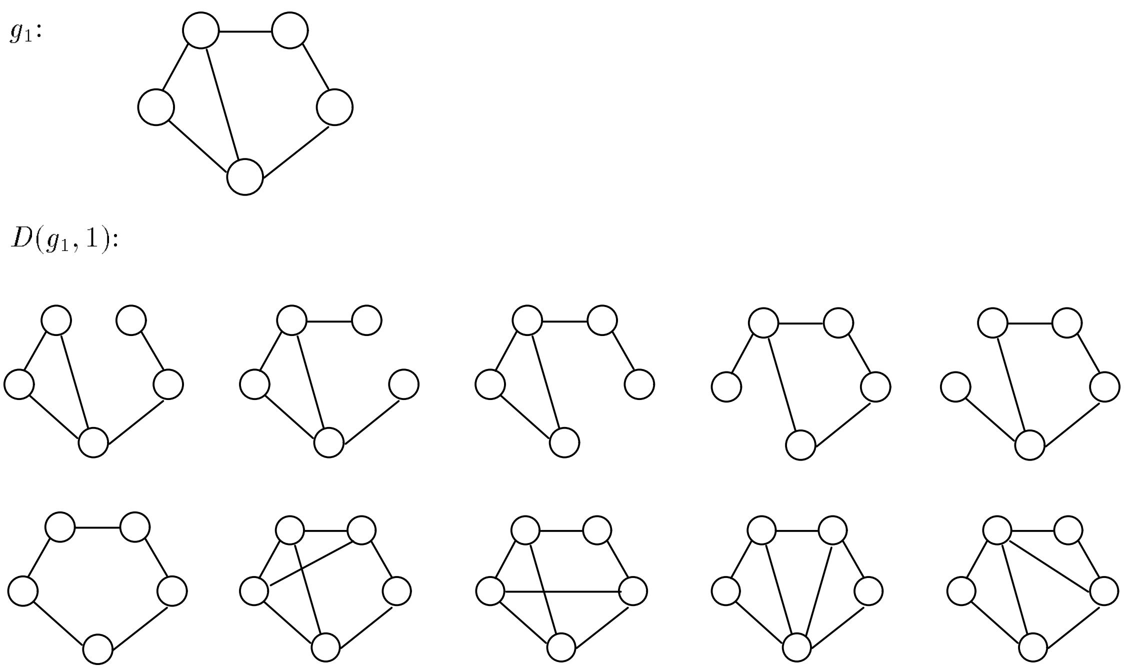

Figure 3: The graph $g_1$ and the set $D(g_1, \vartheta)$ for $\vartheta = 1$.

only possible edit operations are the insertion and the deletion of an edge. Each such operation is assigned a cost equal to 1. Hence, $D(g_1, 1)$ consists of exactly ten edited copies of $g_1$.

After $D(G, \vartheta)$ has been computed, the optimal error-correcting graph isomorphism (if it exists) can be determined by testing each graph $G'' \in D(G, \vartheta)$ with $G'$ for graph isomorphism. Clearly, such a scheme is only efficient if the exact graph isomorphism process is computationally inexpensive. From the previous section, we know that the decision tree approach to graph isomorphism detection has only quadratic time complexity. Furthermore, it is completely independent of the number of model graphs in the database. Thus, we propose to use the decision tree method for the detection of graph isomorphisms between the graphs in $D(G, \vartheta)$ and $G'$.

There are two possible implementations of the scheme described above. Given a model graph $G$ and an input graph $G_I$, we can either compute the set $D(G, \vartheta)$ or the set $D(G_I, \vartheta)$. In the first case, $D(G, \vartheta)$ can be computed off-line and at run time it is tested if $G_I \in D(G, \vartheta)$. In the second case, due to the fact that $G_I$ becomes available at run time only, $D(G_I, \vartheta)$ must be computed on-line before the condition $G \in D(G_I, \vartheta)$ can be tested. In the following, both cases are studied in greater detail.

```
COMPILE_ERROR-CORRECTING_TREE(GRAPHS $G_1, \ldots, G_L$, THRESHOLD $\vartheta$)

1. for each $G_i$ with $i = 1, \ldots, L$
    (a) generate $D(G_i, \vartheta)$ by applying all edit operations and combinations of
        edit operations to $G_i$
    (b) for each $G' \in D(G_i, \vartheta)$
        compile $G'_i$ into the decision tree and register the cost $d(G_i, G'_i)$
```

Figure 4: Algorithm *compile_error-correcting_tree*.

## 4.1 Off-line Computation of Error Corrections

Given a set of model graphs $G_1, \ldots, G_L$, an input graph $G_I$ and a maximal error threshold $\vartheta$, we compute off-line for each model graph $G_i$ the set $D(G_i, \vartheta)$, $i = 1, \ldots, L$, and generate a decision tree for this set. At run time, the decision tree is used to find out whether the input graph $G_I$ (with adjacency matrix $M_I$) is isomorphic to any of the distorted copies $G'_i = \Delta_i(G_i)$ of one of the model graphs $G_i$ (with adjacency matrix $M_i$). That is, if it is detected that $G_I$ is not isomorphic to any of $G_1, \ldots, G_L$ but is isomorphic to $G'_i = \Delta_i(G_i)$ and $P$ is the permutation matrix corresponding to this isomorphism then $(\Delta_i, P)$ denotes an *oec* graph isomorphism between $G_i$ and $G_I$. In Fig. 4 the compilation of the so-called *error-correcting* decision tree for a set of model graphs is outlined. Actually, the result is a decision tree for the set $\cup_{i=1}^{L} D(G_i, \vartheta)$.

The advantage of computing the sets $D(G_i, \vartheta)$ with $i = 1, \ldots, L$ off-line and compiling a decision tree that incorporates all of the generated graphs is that, at run time, the condition $G_I \in \cup_{i=1}^{L} D(G_i, \vartheta)$ can be tested in quadratic time only. That is, given $L$ model graphs each consisting of $n$ vertices, an error threshold $\vartheta$, and an input graph with $n$ vertices, the *oec* graph isomorphism search based on the error-correcting decision tree has a time complexity of only

$$O(n^2) \tag{5}$$

In particular, the time complexity is completely independent of the number $L$ of model graphs and the size of the sets $D(G_i, \vartheta)$; $i = 1, \ldots, L$. Note that the conventional $A^*$-based algorithm has in the best case a computational complexity of $O(Ln^4)$ and in the worst case of $O(Ln^{2n})$ [MB95a]. Hence, the decision tree approach outperforms the conventional algorithm at least by a factor of $O(Ln^2)$. However, the disadvantage of this method is the size of the decision tree. In Section 3, it was mentioned that the size of a decision tree for $L$ models, each consisting of $n$ vertices, is bounded by $O(L3^n)$. Clearly, in the case of an error-correcting decision tree, for each model $G_i$ all edited graphs in $D(G_i, \vartheta)$ must be compiled into the decision tree as well. Hence, the total size of the decision tree strongly depends on the size of $D(G_i, \vartheta)$ for $i = 1, \ldots, L$. The size of $D(G, \vartheta)$ for a graph $G$ depends on the set of edit operations. It can be computed as follows. If we assume that the cost of any edit operation is equal to 1 and that $\vartheta$ assumes integer values only, i.e. $\vartheta \in \{1, 2, \ldots\}$, then the number of edited graphs for a graph $G$ with $n$ vertices depends on the number of possible changes of the elements of its adjacency matrix. For example, with $\vartheta = 1$ each entry in the $n \times n$-adjacency matrix of $G$ may be changed exactly once, resulting in $O(n^2)$ different edited graphs. In the previous example in Fig. 3, the graph $g_1$ is undirected and unlabeled. Hence, the maximal number of edited graph with edit cost 1 is exactly $n(n-1)/2$. For $n = 5$ this number corresponds to the ten edited copies of $g_1$ shown in Fig. 3[2]. In general, the number of edited graphs in $D(G, \vartheta)$ under the assumption that the cost of any edit operation is equal to 1 is bounded by

$$O(\vartheta n^{2\vartheta}) \tag{6}$$

Consequently, the total size of an error-correcting decision tree that incorporates $D(G_i, \vartheta)$ for $i = 1, \ldots, L$ is bounded by

$$O(\vartheta n^{2\vartheta} L 3^n) \tag{7}$$

We conclude that while the error-correcting decision tree allows the detection of error-correcting graph isomorphisms in quadratic time only, it requires an amount of memory that is exponential in both the size of the model graphs and the maximal error that is to be considered. Naturally, the procedure that computes the decision tree is of exponential time complexity.

## 4.2 On-line Computation of Error Corrections

An alternative to the off-line computation of all possible error corrections as it was described in the previous section is to compute $D(G_I, \vartheta)$ for the input graph on-line and test whether there exists a model graph $G_i$ with $G_i \in D(G_I, \vartheta)$. Hence, in the off-line preprocessing step only the model graphs $G_1, \ldots, G_L$ are compiled into a decision tree. At run time, each edited graph in $D(G_I, \vartheta)$ is then tested for isomorphism to any of the model graphs by means of the decision tree. If $G'_I \in D(G_I, \vartheta)$ with adjacency matrix $M'_I$ is isomorphic to the model $G_i$ with adjacency matrix $M_i$, and $P$ is the permutation matrix such that $PM_iP^T = M'_I$ then it is true that $M_i = P^T M'_I P = P^T \Delta'(M_I) P$ where $\Delta'$ is the edit sequence transforming $G_I$ into $G'_I$. In Fig. 5 the on-line error-correcting graph isomorphism algorithm based on a decision tree is outlined, where the error-correcting graph isomorphism between $G_I$ and $G_i$ is denoted by $(P, \Delta')$.

[2]Note that some of these graphs may be isomorphic.

```
ERROR-CORRECTING_GRAPH_ISOMORPHISM(GRAPH $G_I$, $\vartheta$)

1. generate $D(G_I, \vartheta)$ by applying all edit operations and combinations of edit
   operations to $G_I$
2. for each $G'_I \in D(G_I, \vartheta)$
   (a) classify the adjacency matrix of $G'_I$ with the decision tree representing
       the model graphs $G_1, \ldots, G_L$
   (b) if $G'_I$ is successfully classified by the decision tree as being isomorphic to
       the graph $G_i$ and $P$ is the corresponding permutation matrix, then add
       the error-correcting graph isomorphism $(P, \Delta')$ between $G_I$ and $G_i$ (see
       text) to the list $F$.
3. output the error-correcting graph isomorphism $(P, \Delta')$ with the least cost
   $C(\Delta')$ in $F$.
```

Figure 5: Algorithm *error-correcting_graph_isomorphism.*

The main advantage of the on-line computation of the error corrections is that the decision tree incorporates only the model graphs themselves. Hence, for $L$ models and $n$ vertices in each model, its size is bounded by $O(L3^n)$ compared to $O(\vartheta n^{2\vartheta} L3^n)$ in the case of the off-line computation (see Eq. 7). The run time complexity, on the other hand, depends on the size of $D(G_I, \vartheta)$ and the time required for testing whether a graph in $D(G_I, \vartheta)$ is isomorphic to any of the model graphs. Because the model graphs are organized in a decision tree, a single graph isomorphism test for an input graph with $n$ vertices can be performed in $O(n^2)$ steps. Clearly, such a test must be performed for each graph $G'_I$ in $D(G_I, \vartheta)$.From Eq.(6) we know that, for a constant cost of 1 of each edit operation, the size of $D(G_I, \vartheta)$ for a graph $G_I$ with $n$ vertices is bounded by $O(\vartheta n^{2\vartheta})$. Therefore, the time complexity of the on-line *ec* graph isomorphism algorithm based on a decision tree is bounded by

$$O(\vartheta n^{2(\vartheta+1)}) \qquad (8)$$

It is important to note that the time complexity in Eq.(8) is completely independent of the number of model graphs in the database. We conclude that the on-line error-correcting algorithm based on a decision tree is especially efficient if the database of model graphs is large and the maximal degree of distortion to be considered is rather small.

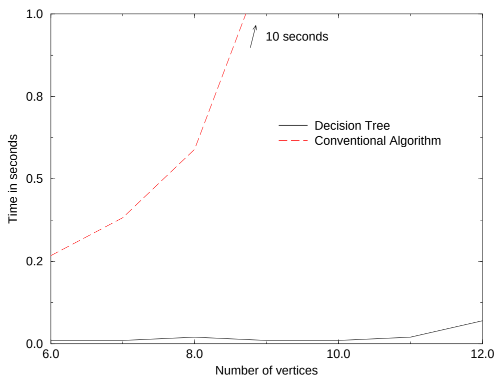

Figure 6: Computation time in seconds for $\vartheta = 1$ and a growing number of vertices (first experiment).

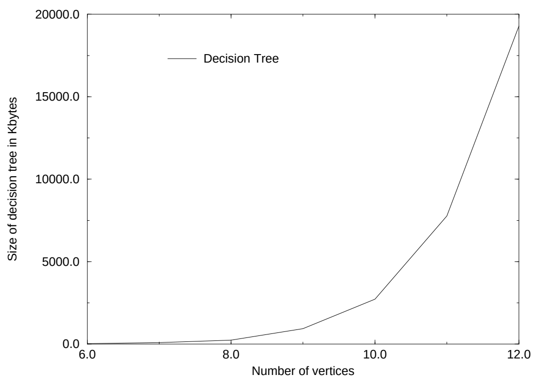

Figure 7: Size of decision tree in Kbytes for $\vartheta = 1$ and a growing number of vertices (first experiment).

## 5 Experimental Results

In order to examine the efficiency of the new algorithms in practice, we have performed a number of experiments with randomly generated graphs. Both the new decision tree algorithms (with on-line and off-line error correction) and a conventional, $A^*$-based algorithm were implemented in C++ and run on a SUN Sparc10 Workstation. For each experiment, we generated one or more model graphs and used these model graphs to create input graphs that were distorted copies of the model graphs. All of the graphs generated for the experiments in this section were undirected and unlabeled. Each experiment was repeated 20 times and the average computation time was recorded. The size of the decision trees in terms of disk space is also given for each experiment. In the first two experiments, we studied the performance of the off-line *ec* algorithm compared to the conventional $A^*$-based method for a varying number of vertices in the graphs and a varying number of models in the database [BA83]. Then, in the last four experiments, the performance of the on-line error-correcting algorithm was tested for varying model and database sizes and varying degrees of distortion.

In the first experiment, documented in Figs. 6 and 7, the database contained a single model graph. The size of this model graph was steadily increased from 6 vertices and 12 edges to 12 vertices and 24 edges. From each model graph an input graph was derived by first copying the model and then deleting or inserting an edge. Hence, the maximal error to be considered was set to $\vartheta = 1$ for both the decision tree approach and the conventional algorithm. In Fig. 6 the time in seconds required by both algorithms for the detection of the *oec* graph isomorphism between the model and the input graph is displayed. Clearly, the decision tree approach was much more efficient than the conventional algorithm. In the end, the conventional algorithm required for a graph with 12 vertices approximately 10 seconds while the decision tree approach needed less than 0.1 seconds. The drawback of the decision

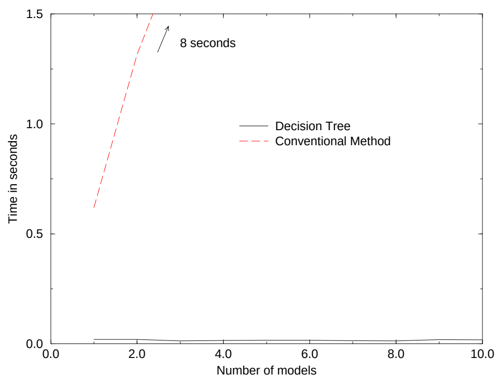

Figure 8: Computation time in seconds for $\vartheta = 1$ and a growing number of models (second experiment).

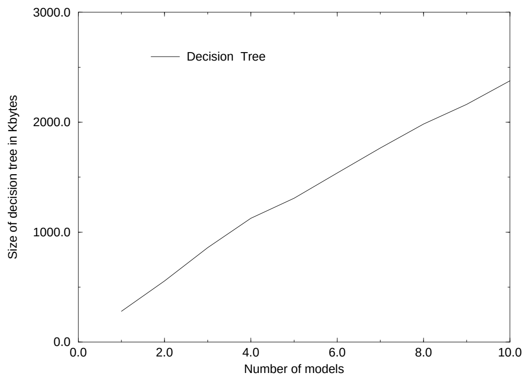

Figure 9: Size of decision tree in Kbytes for $\vartheta = 1$ and a growing number of models (second experiment).

tree approach, however, is the size of the decision tree. In Fig. 7 the disk space in terms of Kbytes that is required for the decision trees in the first experiment is displayed. As expected, the size of the tree grows exponentially. In the end, for a graph consisting of 12 vertices almost 20 Mbytes were required. We conclude that for an error threshold $\vartheta = 1$, model graphs with no more than 12 vertices can be treated by our implementation of the off-line error-correcting method. Due to the theoretical space complexity of $O(\vartheta n^{2\vartheta})$ (see Eq. 6) we did not try this method for error thresholds $\vartheta \geq 2$.

In the second experiment, the number of model graphs in the database was steadily increased from 1 to 10. Each model graph consisted of 8 vertices and 16 edges and the input graphs were again copies of the models with a missing or an extraneous edge. In Fig. 8 the time required by the off-line error-correcting decision tree approach and the conventional method for an error threshold $\vartheta = 1$ are displayed. As it was expected from the theoretical complexity analysis, the decision tree procedure was completely independent of the size of the database. It constantly terminated after 0.05 seconds. The conventional algorithm, on the other hand, was linearly dependent on the number of model graphs and required in the end 8 seconds. The size of the decision tree (see Fig. 9) grew linearly with the number of models. In the end, for 10 model graphs, the decision tree required about 2.4 Mbytes of memory.

The results of the first two experiments indicate that the off-line error-correcting method is very well suited for databases of small graphs and applications where little distortion occurs. However, for problems with larger graphs and higher error thresholds the off-line error computation is no longer possible. Hence, we also performed a series of experiments with the on-line error computation method. In the third experiment, the size of the model graph was increased from 6 vertices and 12 edges to 16 vertices and 32 edges. The error threshold was kept at $\vartheta = 1$ and from each model graph an input was constructed by copying the model and adding or removing an edge. The computation time required by the on-line error-correcting

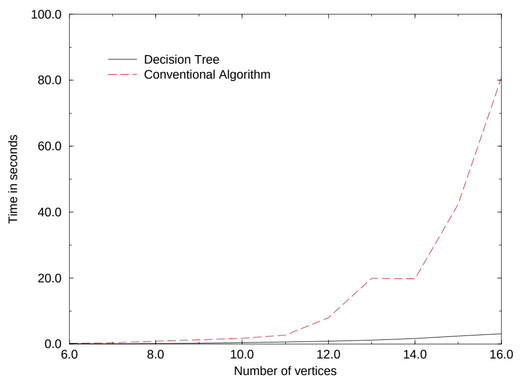

Figure 10: Computation time in seconds for $\vartheta = 1$ and a growing number of vertices (third experiment).

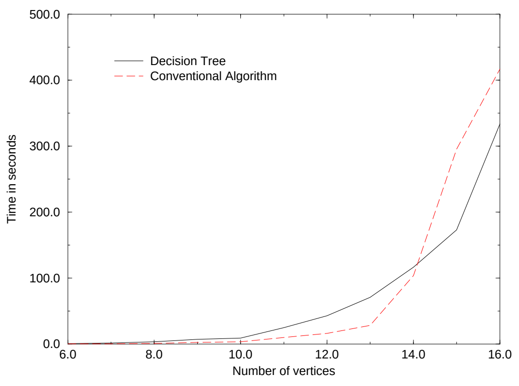

Figure 11: Computation time in seconds for $\vartheta = 2$ and a growing number of vertices (fourth experiment).

algorithm and by the conventional algorithm are displayed in Fig. 10. Similarly to the first experiment, the decision tree method is much faster than the conventional algorithm. This result corresponds to the theoretical complexity analysis in which we showed, that, for $\vartheta = 1$, the $O(n^4)$ complexity of the on-line decision tree approach is always equal to, or smaller than, the complexity of the conventional algorithm (with a best case of $O(n^4)$ and a worst case of $O(n^{2n})$). However, while the off-line error-correcting method required only 0.01 seconds for a graph with 12 vertices (Fig. 6), the on-line error-correcting method took more than 0.9 seconds. The advantage of the on-line method is that, due to the fact that only the model graphs are compiled into the decision tree, graphs with up to 16 vertices can be handled. For the same reason, it is also possible to apply the method to problems with higher error thresholds.

In the fourth experiment, the error threshold was set to $\vartheta = 2$. That is, the input graphs differed from the model graphs by either 2 missing, 2 extraneous, or 1 missing and 1 extraneous edge. The size of the model graphs was again increased from 6 vertices and 12 edges to 16 vertices and 32 edges. The results of the fourth experiment are displayed in Fig. 11. Note that for graphs with less than 14 vertices, the conventional algorithm is faster than the decision tree approach. For larger graphs, however, the decision tree approach outperforms the conventional algorithm[3].

Clearly, the size of the decision tree (Fig. 14) is identical in the third and fourth experiment, because in the on-line error-correcting method the value of the error threshold does not influence the preprocessing step. Note that while the decision tree in the on-line error-correcting method (Fig. 7) required 20 Mbytes of memory, the decision tree for the off-line error-correcting method required only 1 Mbyte of memory for a graph with 12 vertices, and less than 15 Mbytes for a graph with 16

[3]This is mainly due to the fact that the theoretical complexity of the decision approach is guaranteed to be $O(n^6)$ (see Eq. 8 for $\vartheta = 2$) while the conventional algorithm's complexity may vary between $O(n^4)$ and $O(n^{2n})$ where $n$ is the number of vertices in the input and the model graphs.

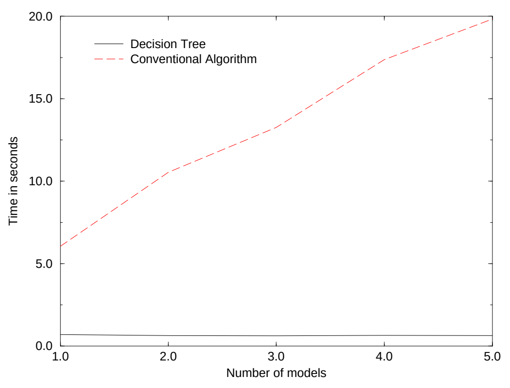

Figure 12: Computation time in seconds for $\vartheta = 1$ and a growing number of models (fifth experiment).

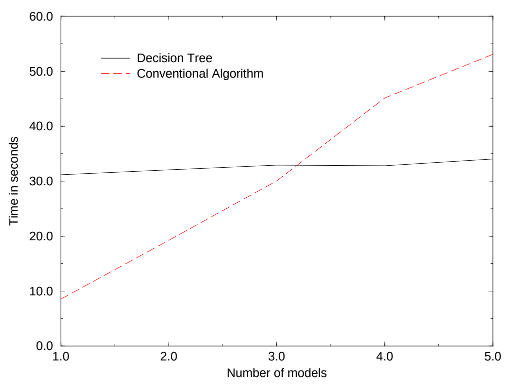

Figure 13: Computation time in seconds for $\vartheta = 2$ and a growing number of models (sixth experiment).

vertices.

Finally, in the fifth and the sixth experiment, we tested the performance of the on-line decision tree approach for a growing number of model graphs. In both experiments, the number of models in the database was gradually increased starting at one and ending at five graphs. Each model graphs consisted of 11 vertices and 22 edges. In the fifth experiment documented in Fig. 12, the error threshold was kept at $\vartheta = 1$ while in the sixth experiment documented in Fig. 13, the error threshold was set to $\vartheta = 2$. Notice that in both experiments, the time required by the decision tree method was independent of the number of model graphs while the conventional method's performance was linearly dependent on the size of the database. We can observe that for $\vartheta = 1$ the decision tree approach is always faster than the conventional algorithm. For $\vartheta = 2$, the decision tree approach is slower than the conventional algorithm when the database is small. But for more than 3 models in the database, the decision tree method becomes superior. The size of the decision tree in the fifth and the sixth experiment is displayed in Fig. 15. Complying with our complexity analysis, it is linearly dependent on the database size.

## 6 Conclusions

We have presented two new algorithms for the problem of error-correcting graph isomorphism detection that are based on the decision tree paradigm. The new algorithms are extensions of an algorithm for exact graph and subgraph isomorphism detection that was developed by the authors and described in [MB95b, MB95c]. The two new algorithms mainly differ in the way they handle error corrections. In the first method, the error-corrections are computed off-line and used to create a decision tree which is capable of detecting error-correcting graph isomorphisms in quadratic time only. In the second method, the error corrections are computed on-line on the basis of the input graph and each edited copied of the input graph is then subjected

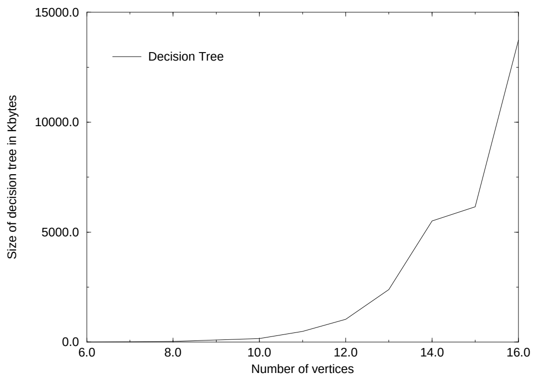

Figure 14: Size of the decision tree for a growing number of vertices (third and fourth experiment).

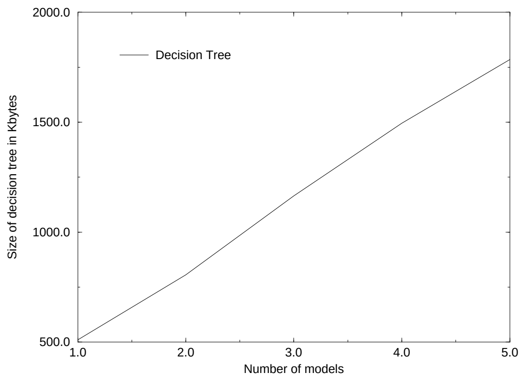

Figure 15: Size of the decision tree for a growing number of models (fifth and sixth experiment).

to a graph isomorphism test by means of a decision tree traversal. This results in a time complexity that is quadratic in the size of the model graphs and exponential in the error threshold that is to be considered. Very important is the fact that both methods are completely independent of the number of model graphs that are represented by the decision tree. On the other hand, a complexity analysis revealed that the size of the decision tree grows exponentially with the size of the model graphs for both new algorithms.

The results of the theoretical complexity analysis have been confirmed in a number of practical experiments with randomly generated graphs. The advantages of the new algorithms in terms of computational performance were demonstrated in these experiments for graphs with up to 12 vertices for the off-line error-correcting method and for graphs with up to 16 vertices for the on-line version.

Despite the exponential space complexity, we believe that there are potential applications of the new graph matching algorithm. It is particularly of interest if the underlying graphs and the degree of distortion are rather small but computation time is critical. We also believe that the general approach of using decision trees for graph matching is worth further investigation. Topics of future research may include the adaption of the method to the general error-correcting subgraph isomorphism problem. In the experiments described in this paper, the edit operations were restricted to insertions and deletions of edges. However, depending on the particular application, other edit operations can be introduced, for example, the substitution, deletion and insertion of nodes. Another interesting aspect will be a possible combination of the off-line and on-line method introduced in this paper.

**Acknowledgment**
This work is part of a project of the Priority Program SPP IF, No: 5003-34285, funded by the Swiss National Science Foundation.

## References

[AD93] H.A. Almohamad and S.O. Duffuaa. A linear programming approach for the weighted graph matching problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI*, 5:522–525, 1993.

[BA83] H. Bunke and G. Allerman. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters 1*, 4:245–253, 1983.

[CYS+96] L. Cinque, D. Yasuda, L.G. Shapiro, S. Tanimoto, and B. Allen. An improved algorithm for relational distance graph matching. *Pattern Recognition*, 29(2):349–359, 1996.

[HHVN90] L. Herault, R. Horaud, F. Veillon, and J.J. Niez. Symbolic image matching by simulated annealing. In *Proc. British Machine Vision Conference*, pages 319–324. Oxford, 1990.

[KCP92] J. Kittler, W. J. Christmas, and M. Petrou. Probabilistic relaxation for matching of symbolic structures. In H. Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition*, pages 471–480. World Scientific, 1992.

[MB95a] B.T. Messmer and H. Bunke. Efficient error-tolerant subgraph isomorphism detection. In D. Dori and A. Bruckstein, editors, *Shape, Structure and Pattern Recognition*, pages 231–240. World Scientific Publ. Company, Singapore, 1995.

[MB95b] B.T. Messmer and H. Bunke. Subgraph isomorphism detection in polynomial time on preprocessed model graphs. In *Proceedings of the Asian Conference on Computer Vision ACCV*, pages 151–155, 1995.

[MB95c] B.T. Messmer and H. Bunke. Subgraph isomorphism in polynomial time. Technical Report IAM-95-003, University of Bern, 1995.

[SF83] A. Sanfeliu and K.S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:353–363, 1983.

[SH81] L.G. Shapiro and R.M. Haralick. Structural descriptions and inexact matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI*, 3:504–519, 1981.

[Ull76] J.R. Ullman. An algorithm for subgraph isomorphism. *Journal of the Association for Computing Machinery*, 23(1):31–42, 1976.

[WF74] R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21(1):168–173, 1974.

[Won90] E. K. Wong. Three-dimensional object recognition by attributed graphs. In H. Bunke and A. Sanfeliu, editors, *Syntactic and Structural Pattern Recognition- Theory and Applications*, pages 381–414. World Scientific, 1990.