A New Algorithm for Error-Tolerant Subgraph Isomorphism Detection

Bruno T. Messmer and Horst Bunke, *Member, IEEE Computer Society*

Abstract—In this paper, we propose a new algorithm for error-correcting subgraph isomorphism detection from a set of model graphs to an unknown input graph. The algorithm is based on a compact representation of the model graphs. This representation is derived from the set of model graphs in an off-line preprocessing step. The main advantage of the proposed representation is that common subgraphs of different model graphs are represented only once. Therefore, at run time, given an unknown input graph, the computational effort of matching the common subgraphs for each model graph onto the input graph is done only once. Consequently, the new algorithm is only sublinearly dependent on the number of model graphs. Furthermore, the new algorithm can be combined with a future cost estimation method that greatly improves its run-time performance.

Index Terms—Graphs, subgraph isomorphism, graph matching, error-correcting graph matching, search, graph algorithms, graph decomposition.

1 Introduction

Due to their representational power, attributed graphs are widely used in various domains of computer science. Particularly in computer vision and pattern recognition, they have been used to represent complex structures such as Chinese characters [11], hand-drawn symbols [10], aerial road network images [3], 3D-objects [24], [4], and others. In many applications, these complex structures must be classified, detected, or compared to each other by means of some matching scheme. By using attributed graphs for the representation, the matching process can be formulated as a search for graph or subgraph isomorphisms. However, real world objects are often affected by noise such that the graph representations of identical objects may not exactly match. Thus, it is necessary to integrate the concept of error correction into the matching process. Depending on the problem domain, correspondences between the models and the input graph can then be established by searching for either error-correcting graph isomorphisms between the models and the input, error-correcting subgraph isomorphisms from the models to the input, or error-correcting subgraph isomorphisms from the input to the models. Because error-correcting graph isomorphism is a special case of error-correcting subgraph isomorphism, and because input graphs are often larger than model graphs, we are particularly interested in the problem of error-correcting subgraph isomorphism detection from a set of models to an input graph.

It is well known that subgraph isomorphism detection is an NP-complete problem [8]. For example, the number of computational steps required to detect all subgraph isomorphism from one graph to another is exponential in the size of the underlying graphs. Consequently, error-correcting subgraph isomorphism detection is also in NP and generally harder than exact subgraph isomorphism detection.

In the past, various approaches to error-correcting subgraph isomorphism detection have been proposed. The most common approach is based on tree search with the A^* algorithm [15]. The search space of the A^* algorithm can be greatly reduced by applying heuristic error estimation functions. In the domain of computer vision, numerous heuristics have been proposed [24], [22], [5], [17], [18], [1]. All of these methods are guaranteed to find the optimal solution but require exponential time in the worst case. Random methods, on the other hand, are polynomially bounded in the number of computation steps but may fail to find the optimal solution. For example, in [9], [3] a probabilistic relaxation scheme is described, which works well for large graphs, but may miss the optimal match in some cases. Other approaches are based on neural networks such as the Hopfield network [6] or the Kohonen map [26]. However, all of these random methods may get trapped in local minima and miss the optimal solution.

An additional problem in many applications is that there is not only one, but several a priori known model graphs that must be matched onto a single input graph. The methods for error-tolerant graph matching mentioned so far work on only two graphs at a time. Thus, for databases which contain more than one model graph it is necessary to apply the graph matching method to each model-input pair, resulting in a linear dependency on the size of the database. This may be prohibitive for large databases. In order to overcome this problem, some authors proposed to organize the database of graphs such that the number of error-correcting subgraph isomorphism searches can be reduced. For example, in [19], [21], [20], a hierarchical organi-

- *B.T. Messmer is with Corporate Technology, Swisscom AG, Ostermundigenstr. 93, CH-3000 Bern 29, Switzerland. E-mail: Bruno.Messmer@swisscom.com.*
- *H. Bunke is with Institut für Informatik und angewandte Mathematik, University of Bern, Neubrückstr. 10, CH-3012 Bern, Switzerland. E-mail: bunke@iam.unibe.ch.*

Manuscript received 7 Dec. 1995; revised 22 Dec. 1997. Recommended for acceptance by K. Boyer.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 106430.

zation of the database was proposed, where the hierarchy is determined by clustering the model graphs into similarity classes. For a given input graph, the hierarchy is traversed by first matching the input graph onto the root of the hierarchy and then choosing the branch that represents the class of model graphs which are most similar to the input. The indexed class of model graphs is then again clustered into similarity classes or, if only few models are left, each of them is directly matched with the input graph. Another hierarchical organization was proposed in [16], where at the root of the hierarchy a supergraph, consisting of different distinct subgraphs of the model graphs is placed and matched against the input. The disadvantage of this scheme, however, is that the root graph may become much larger than the individual model graphs and thus the first matching process may be more time consuming than the sum of each individual graph match between a model and the input.

In this paper, we present a new approach to the problem of error-correcting subgraph isomorphism detection between a database of model graphs and an unknown input graph. The approach is based on a compact representation of the model graphs. This representation is derived from the model graphs in an off-line preprocessing step. In this preprocessing step, the model graphs are decomposed into smaller subgraphs and represented in terms of these subgraphs. If a subgraph appears multiple times within the same model graph or in different models, it will be represented only once. Hence, the resulting representation of the models is very compact. At run time, this compact representation is used to efficiently detect error-tolerant subgraph isomorphisms from the model graphs to the input. Common subgraphs that are part of different model graphs are matched only once with the input. Consequently, the complexity of the new algorithm is only sublinearly dependent on the size of the database. Furthermore, the algorithm can be combined with a very efficient future cost estimation technique.

The rest of this paper is organized as follows. In Section 2 the main definitions and notation that will be used throughout the paper are given. A description of the new algorithm is given in Section 3. In Section 4, a number of practical experiments are described. Finally, in Section 5 a discussion and conclusions are provided.

2 Definitions and Notation

The algorithms presented in this paper work on labeled graphs. Let L_V and L_E denote the set of vertex and edge labels, respectively.

DEFINITION 1. *A graph G is a four-tuple $G = (V, E, \mu, \xi)$, 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,
- $\xi: E \rightarrow L_E$ is a function assigning labels to the edges.

In this definition, the edges are directed, i.e., there is an edge from v_1 to v_2 if $(v_1, v_2) \in E$. For graphs with undirected edges, we require that $(v_2, v_1) \in E$ for any edge $(v_1, v_2) \in E$.

DEFINITION 2. *Given a graph $G = (V, E, \mu, \xi)$, a subgraph of G is a graph $S = (V_s, E_s, \mu_s, \xi_s)$ such that*

1) $V_s \subseteq V$
2) $E_s = E \cap (V_s \times V_s)$
3) *μ_s and ξ_s are the restrictions of μ and ξ to V_s and E_s, respectively, i.e.,*

$$\mu_s(v) = \begin{cases} \mu(v) & \text{if } v \in V_s \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\xi_s(e) = \begin{cases} \xi(e) & \text{if } e \in E_s \\ \text{undefined} & \text{otherwise} \end{cases}$$

From this definition it is easy to see that, given a graph G, any subset of its vertices uniquely defines a subgraph of G. We use the notation $S \subseteq G$ to indicate that S is a subgraph of G.

DEFINITION 3. *Given a graph $G = (V, E, \mu, \xi)$ and a subgraph $S = (V_s, E_s, \mu_s, \xi_s)$ of G, the* difference *of G and S is the subgraph of G that is defined by the set of vertices $V - V_s$.*

The difference of a graph G and a subgraph S of G is denoted by $G - S$.

DEFINITION 4. *Given two graphs $G_1 = (V_1, E_1, \mu_1, \xi_1)$, $G_2 = (V_2, E_2, \mu_2, \xi_2)$, where $V_1 \cap V_2 = \emptyset$, and a set of edges $E' \subseteq (V_1 \times V_2) \cup (V_2 \times V_1)$ with a labeling function $\xi': E' \rightarrow L_E$, the* union *of G_1 and G_2 with respect to E' is the graph $G = (V, E, \mu, \xi)$ such that*

1) $V = V_1 \cup V_2$
2) $E = E_1 \cup E_2 \cup E'$
3) $\mu(v) = \begin{cases} \mu_1(v) & \text{if } v \in V_1 \\ \mu_2(v) & \text{if } v \in V_2 \end{cases}$
4) $\xi(e) = \begin{cases} \xi_1(e) & \text{if } e \in E_1 \\ \xi_2(e) & \text{if } e \in E_2 \\ \xi'(e) & \text{if } e \in E' \end{cases}$

The union of two graphs G_1 and G_2 with respect to a set of edges E' according to Definition 4 will be denoted by $G_1 \cup_{E'} G_2$.

DEFINITION 5. *A bijective function $f: V \rightarrow V'$ is a* graph isomorphism *from a graph $G = (V, E, \mu, \xi)$ to a graph $G' = (V', E', \mu', \xi')$ if:*

1) *$\mu(v) = \mu'(f(v))$ for all $v \in V$.*
2) *For any edge $e = (v_1, v_2) \in E$ there exists an edge $e' = (f(v_1), f(v_2)) \in E'$ such that $\xi(e) = \xi'(e')$, and for any $e' = (v_1', v_2') \in E'$ there exists an edge $e = (f^{-1}(v_1'), f^{-1}(v_2')) \in E$ such that $\xi(e) = \xi'(e')$.*

DEFINITION 6. *An injective function $f: V \rightarrow V'$ is a* subgraph isomorphism *from G to G' if there exists a subgraph $S \subseteq G'$ such that f is a graph isomorphism from G to S.*

Apparently, graph isomorphism is a special case of subgraph isomorphism. For the remainder of this paper, we will assume that there is a number of a priori known graphs, the so-called *model graphs*, and an *input graph*, which is given on-line. The input and model graphs will be also referred to as input and models, for short.

Real-world objects which are represented by graphs may be affected by noise and distortions. In order to compare the undistorted model graphs to a distorted input graph and decide which of the models is most similar to the input, it is necessary to define a distance measure for graphs. Similar to the string matching problem where edit operations are used to define the string edit distance [23], we define a subgraph edit distance that is based on the idea of compensating the distortions in the input graph by means of edit operations that are applied to the model graphs [1]. That is, the graph edit operations are used to alter the model graphs until there exist subgraph isomorphisms to the input graph. To each of the graph edit operations, a certain cost is assigned. The subgraph 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 compensation of the distortions in the input graph. It can be concluded that the smaller the subgraph distance between a model and an input graph is, the more similar they are. We consider the following distortions in a graph: vertex and edge label distortions, missing vertices, and missing or extraneous edges. Notice that it is not necessary to consider extraneous vertices in the input graph as we are searching for subgraph isomorphisms from the model to the input graph. Thus, extraneous vertices in the input graph are automatically ignored. For each type of distortion given above, a corresponding graph edit operation is defined.

DEFINITION 7. *Given a graph $G = (V, E, \mu, \xi)$, a* graph edit operation δ *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).*
- $\xi(e) \to l'$, $e \in E$, $l' \in L_E$: *substituting the label $\xi(e)$ of edge e by l′ (for the correction of edge label distortions).*
- $v \to \$$, $v \in V$: *deleting the vertex v from G (for the correction of missing vertices).*[1]
- $e \to \$$, $e \in E$: *deleting the edge e from G (for the correction of missing edges).*
- $\$ \to e = (v_1, v_2)$, $v_1, v_2 \in V$: *inserting an edge between two existing vertices v_1, v_2 of G (for the correction of extraneous edges).*

In order to model the fact that certain distortions are more likely than others, each graph edit operations δ is assigned a certain cost $C(\delta)$. The costs of the graph edit operations are strongly application dependent and must be defined on the basis of heuristic knowledge.

The five edit operations in Definition 7 are powerful enough to transform any graph G into a subgraph of any other graph G'. Therefore, it is always possible to find a sequence of edit operations that transform a model graph G into another graph G' such that a subgraph isomorphism f from G' to a distorted input graph G_I exists.

DEFINITION 8. *Given a graph $G = (V, E, \mu, \xi)$ and an edit operation δ, the edited graph, $\delta(G)$, is a graph $\delta(G) = (V_\delta, E_\delta, \mu_\delta, \xi_\delta)$ with*

$$1)\ V_\delta = \begin{cases} V - \{v\} & \text{if } \delta = (v \to \$) \\ V & \text{otherwise} \end{cases}$$

$$2)\ E_\delta = \begin{cases} E \cup \{e\} & \text{if } \delta = (\$ \to e) \\ E - \{e\} & \text{if } \delta = (e \to \$) \\ E \cap (V_\delta \times V_\delta) & \text{otherwise} \end{cases}$$

$$3)\ \mu_\delta(v) = \begin{cases} l & \text{if } \delta = (\mu(v) \to l) \\ \mu(v) & \text{otherwise} \end{cases}$$

$$4)\ \xi_\delta(e) = \begin{cases} l & \text{if } \delta = (\xi(e) \to l) \\ \xi(e) & \text{otherwise} \end{cases}$$

DEFINITION 9. *Given a graph $G = (V, E, \mu, \xi)$ and a sequence of edit operations $\Delta = (\delta_1, \delta_2, \ldots, \delta_k)$, 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 $C(\Delta) = \sum_{i=1}^{k} C(\delta_i)$. We can now combine the concepts of edited graph and subgraph isomorphism into the concept of error-correcting subgraph isomorphism.

DEFINITION 10. *Given two graphs G and G′, an* error-correcting (ec) subgraph isomorphism *f from G to G′ is a two-tuple $f = (\Delta, f_\Delta)$ where*

1) *Δ is a sequence of edit operations such that there exists a subgraph isomorphism from $\Delta(G)$ to G′,*
2) *f_Δ is a subgraph isomorphism from $\Delta(G)$ to G′.*

The cost of an *ec* subgraph isomorphism $f = (\Delta, f_\Delta)$ is the cost of the edit operations Δ, i.e., $C(f) = C(\Delta)$. It is easy to see that there is usually more than one sequence of edit operations Δ such that a subgraph isomorphism from $\Delta(G)$ to G' exists and, consequently, there is usually more than one error-correcting subgraph isomorphism from G to G'. For our distance measure, we are particularly interested in the error-correcting subgraph isomorphism with minimum cost.

DEFINITION 11. *Let G and G′ be two graphs. The* subgraph distance *from G to G′, d(G, G′), is given by the minimum cost taken over all error-correcting subgraph isomorphisms f from G to G′*:

$$d(G, G') = MIN_\Delta\{C(\Delta) \mid \textit{there is an ec subgraph isomorphism } f = (\Delta, f_\Delta) \textit{ from } G \textit{ to } G'\}$$

The *ec* subgraph isomorphism f associated with $d(G, G')$ is called the optimal error-correcting (*oec*) subgraph isomorphism from G to G'. Notice that the subgraph distance according to Definition 11 is in general not symmetric, i.e., $d(G, G') \neq d(G', G)$. Therefore, it is important to note that the proposed algorithm is designed to compute the graph distance from the model graph G to the input graph G_I, $d(G, G_I)$, and not vice versa.

3 A New Algorithm for Error-Correcting Subgraph Isomorphism Detection

3.1 Overview

In the previous section it was shown that, given a set of models $G_1, \ldots, G_N$ and an input graph G_I, the traditional algorithm can only match a single model graph, G_i, at a

1. Note that all edges in G that are incident with the vertex v are deleted, too.

time to the input graph G_I. We now propose a new method in which the model graphs are not treated individually but integrated in a common data structure. This data structure is then directly used to find the *oec* subgraph isomorphisms from any of the models to the input.

The basic idea of the new method is to recursively decompose the model graphs off-line into smaller subgraphs. In particular, each model graph is decomposed into two subgraphs and each of these subgraphs is again decomposed until the remaining subgraphs consist of only one vertex. The subgraphs resulting from this decomposition process are recorded and the model graphs are represented in terms of these subgraphs. The main advantage of this scheme is that subgraphs which appear in different model graphs are represented only once and can be reused for each of the model graphs. Hence, this representation of the model graphs is fairly compact. Notice that this representational scheme is similar to the RETE network used in forward-chaining production systems [7], [12]. At run time, given an unknown input graph, the search for *ec* subgraph isomorphisms is first performed for the smallest subgraphs of the model graphs. Next, the *ec* subgraph isomorphisms with the least edit costs are recursively combined to form *ec* subgraph isomorphisms for the complete model graphs. Due to the fact that common subgraphs of different model graphs are represented only once, it is sufficient to match these common subgraphs only once onto the input graph. Consequently, the new method is only sublinearly dependent on the number of model graphs. Furthermore, the efficiency of the method can be greatly enhanced by the addition of a future cost estimation technique.

In the following, we introduce the representation of the model graphs and the new *oec* subgraph isomorphism detection algorithm in detail.

3.2 The Data Structures

The new algorithm works on a set $B = \{G_1, \ldots, G_N\}$ of model graphs, that are known a priori, and an input graph that becomes available at run time only. In an off-line preprocessing step, the model graphs are compiled into a compact representation. At run time, this representation is used to efficiently detect the *oec* subgraph isomorphism f_i from a model graph G_i to the input graph such that the corresponding graph distance $d(G_i, G_I)$ is minimal over all model graphs, i.e.,

$$C(f_i) = d(G_i, G_I) = MIN_j\{d(G_j, G_I) \mid j = 1, \ldots, N\} \qquad (1)$$

The compilation of the model graphs in the preprocessing step is based on the idea of decomposing each model graph into two distinct subgraphs. Then, each of these subgraphs is again decomposed into smaller subgraphs until all subgraphs consist of single vertices and cannot be decomposed any further.

DEFINITION 12. *Let $B = \{G_1, \ldots, G_N\}$ be a set of model graphs. A* decomposition *of B, $D(B)$, is a finite set of four-tuples (G, G', G'', E), where*

1) G, G', and G'' are graphs with $G' \subset G$ and $G'' \subset G$
2) E is a set of edges such that $G = G' \cup_E G''$
3) For each G_i there exists a four-tuple $(G, G', G'', E) \in D(B)$ with $G = G_i$; $i = 1, \ldots, N$.
4) For each four-tuple $(G, G', G'', E) \in D(B)$ there exists no other four-tuple $(G_1, G_1', G_1'', E_1) \in D(B)$ with $G = G_1$.
5) For each four-tuple $(G, G', G'', E_1) \in D(B)$
   a) if G' consist of more than one vertex then there exists a four-tuple $(G_1, G_1', G_1'', E_1) \in D(B)$ such that $G' = G_1$
   b) if G'' consists of more than one vertex then there exists a four-tuple $(G_2, G_2', G_2'', E_2) \in D(B)$ such that $G'' = G_2$
   c) if G' consists of one vertex then there exists no four-tuple $(G_3, G_3', G_3'', E_3) \in D(B)$ such that $G' = G_3$
   d) if G'' consists of one vertex then there exists no four-tuple $(G_4, G_4', G_4'', E_4) \in D(B)$ such that $G'' = G_4$.

Informally speaking, a decomposition is a recursive partitioning of graphs into smaller subgraphs, starting with complete models and terminating at the level of single vertices. The first component in a four-tuple (G, G', G'', E) is the graph to be decomposed, G' and G'' are its two parts, and E are the edges in G between G' and G'' (see Conditions 1 and 2 in Definition 12). Condition 3 in Definition 12 makes sure that every model in B is decomposed, and Condition 4 implies that a decomposition is unique. By means of Condition 5 it is guaranteed that a decomposition is complete, i.e., the process of partitioning a graph into two parts is continued until individual vertices are reached. If several models G_i, G_j, … have a common subgraph G, or if G occurs multiple times in one model G_i, it is sufficient to represent G just by one four-tuple (G, G', G'', E) in $D(B)$. This property not only leads to a compact representation of a set of models, B, by means of the decomposition $D(B)$, but it also is the key to an efficient matching procedure at run time.

In Fig. 1, an example of the representation of two graphs g_1, g_2 in a decomposition D_{12} is given. The decomposition D_{12} consists of the four four-tuples $(g_2, s_1, s_2, \{e_5\})$, $(g_1, s_2, s_5, \{e_3, e_4\})$, $(s_1, s_3, s_2, \{e_2\})$, and $(s_2, s_4, s_5, \{e_1\})$. Notice that D_{12} is displayed as a hierarchical network. In this graphical representation, each subgraph s_1, s_2, s_3, s_4, s_5 and each graph g_1 and g_2 is represented by a network node and for each tuple $(S, S', S'', E) \in D_{12}$, there is a network edge from the node for S' to the node for S and a network edge from the node for S'' to the node for S. The network nodes are ordered from top to bottom according to the size of the subgraph they represent. At the top of the network, the nodes representing the subgraphs s_3, s_4, and s_5 are displayed. On the next level, the subgraph s_2 which consists of the subgraphs s_4 and s_5 and the connecting edge e_1 is given, followed by the subgraph s_1 and the model graph g_1. Finally, at the bottom of the network, the model graph g_2 which is decomposed into the subgraphs s_1, s_2 and the edge e_5 is displayed. Notice that the subgraph s_2 appears twice in the model graph g_2 and once in the model graph g_1, but is represented only once in the decomposition.

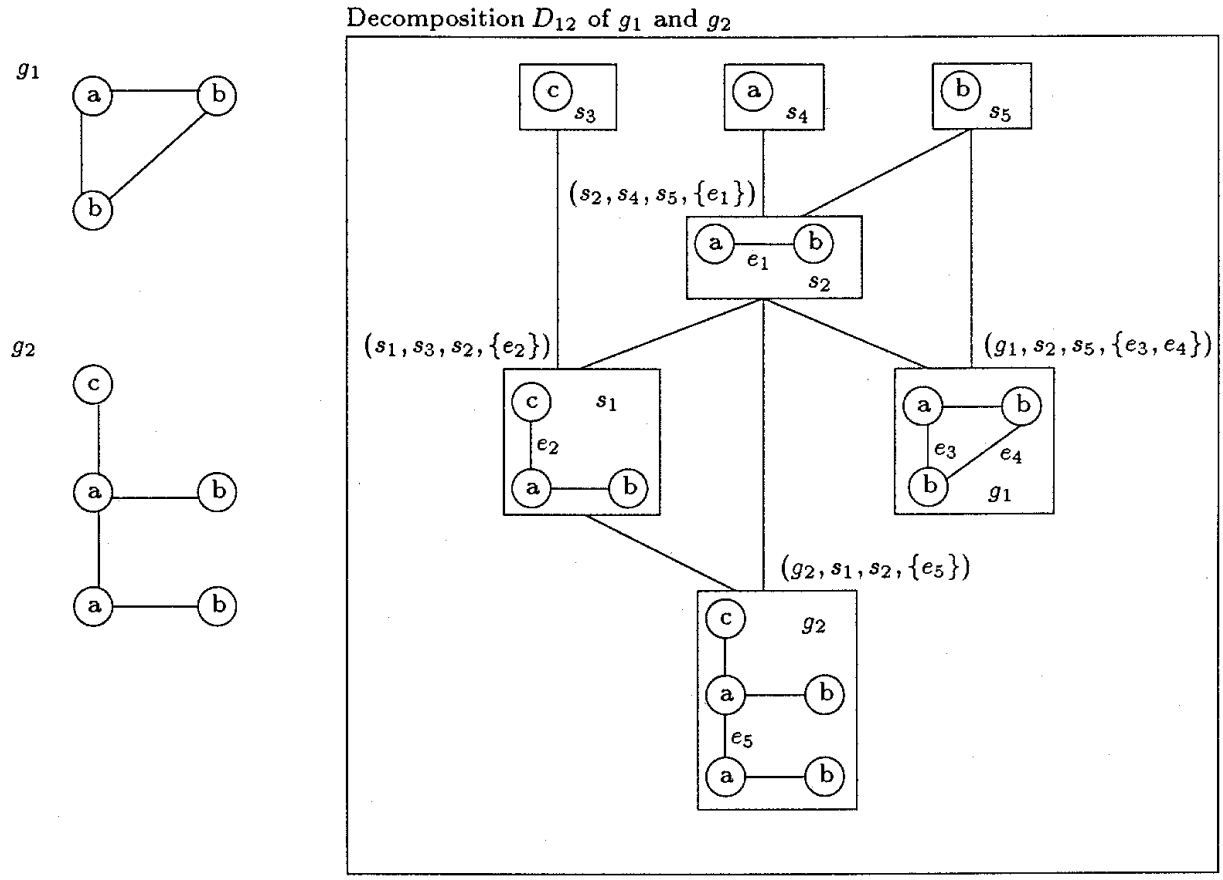


Fig. 1. Two model graph g_1 and g_2 and a graphical representation of the corresponding decomposition D_{12}.

A decomposition algorithm is described in [14]. Notice that model graphs can be decompressed in various ways, in general.

3.3 The Algorithm

The new on-line algorithm for *oec* subgraph isomorphism detection is based on the decomposition of the model graphs. Given a set of models $B = \{G_1, \ldots, G_N\}$, a decomposition $D(B)$ and an input graph G_I, the new algorithm first searches for *ec* subgraph isomorphisms for the subgraphs occurring in $D(B)$. The *ec* subgraph isomorphisms that are found are then combined to form *ec* subgraph isomorphisms for the full models $G_1, \ldots, G_N$. For example, in Fig. 1, the algorithm first matches the subgraphs s_3, s_4, and s_5 onto the graph G_I. Next, the algorithm tries to combine *ec* subgraph isomorphisms that were found for s_4 and s_5 in order to form *ec* subgraph isomorphisms for s_2 and so on. There are three basic problems that must be solved by the algorithm:

Problem (I) For each subgraph S in the decomposition that consists of a single vertex only, find all *ec* subgraph isomorphisms from S to G_I.

Problem (II) For each subgraph S in the decomposition that consists of more than one vertex and is decomposed into subgraphs S_1 and S_2, i.e., $(S, S_1, S_2, E) \in D(B)$, combine *ec* subgraph isomorphisms that are found for S_1 and S_2 such that *ec* subgraph isomorphisms from S to G_I result.

Problem (III) The sequence in which *ec* subgraph isomorphisms for subgraphs in the decomposition are combined must be controlled intelligently in order to avoid combinatorial explosion.

In the following, we examine each of the above problems individually and then formally describe the new algorithm for *oec* subgraph isomorphism detection.

Problem (I) Given a subgraph S that consist of only one vertex v and an input graph G_I, the *ec* subgraph isomorphisms from S to G_I can be generated by mapping v onto each of the vertices in G_I and registering the necessary substitution operation. Additionally, the possible deletion of v must be taken into account as well. In Fig. 2, the procedure *vertex_matching* is displayed. It takes as arguments the single vertex v, its label l and an input graph G_I, and returns the set F of all *ec* subgraph isomorphisms from v to G_I. In the beginning, F is initialized as empty. In step 2, for each v_I in V_I an *ec* subgraph isomorphism $f = (\Delta, f_\Delta)$ is generated where Δ consist of the substitution of l by $\mu_I(v_I)$, i.e., $\Delta = ((l \rightarrow \mu(v_I)))$, and $f_\Delta(v) = v_I$. Each of these *ec* subgraph isomorphisms is collected in F. Finally, in order to account for the possible deletion of a vertex from the model graph, the *ec* subgraph isomorphism

VERTEX_MATCHING(v, l, G_I)

1. let $F = \emptyset$ and $G_I = (V_I, E_I, \mu_I, \nu_I)$
2. for all $v_I \in V_I$
   a) generate an *ec* subgraph isomorphism $f = (\Delta, f_\Delta)$
   b) $F = F \cup \{f\}$
3. create $f' = (\Delta', f_{\Delta'})$ with $\Delta' = (v \rightarrow \$)$ and $f_{\Delta'} = \emptyset$ and set $F = F \cup \{f'\}$
4. output F

Fig. 2. Algorithm *vertex_matching*.

$f' = \left(\Delta', f_{\Delta'}\right)$, where $\Delta' = ((v \rightarrow \$))$ denotes the deletion of v and $f_{\Delta'}$ is the null function, is generated and added to F. The set F is then output.

Problem (II) Given is a graph S that is decomposed into S_1 and S_2, i.e., $(S, S_1, S_2, E) \in D$. Moreover, let $f_1 = \left(\Delta_1, f_{\Delta_1}\right)$ and $f_2 = \left(\Delta_2, f_{\Delta_2}\right)$ be two *ec* subgraph isomorphisms from S_1 and S_2 to G_I, respectively. The problem is to find an *ec* subgraph isomorphism f from $S = S_1 \cup_E S_2$ to G_I that is based on f_1 and f_2. Clearly, f_1 and f_2 can only be combined if no two vertices in $\Delta_1(S_1)$ and $\Delta_2(S_2)$ are mapped onto the same input vertex. More precisely, the intersection of the images of f_{Δ_1} and f_{Δ_2} must be empty. If this is the case, then f_1 and f_2 can be combined into an *ec* subgraph isomorphism f from S to G_I. In Fig. 3, the procedure *combine* for the combination of *ec* subgraph isomorphisms is displayed. The input to *combine* consists of two graphs S_1, S_2, a set of edges E connecting S_1 and S_2, an input graph G_I, and two *ec* subgraph isomorphisms f_1, f_2 for S_1 and S_2, respectively. In the beginning of the procedure, it is tested whether the intersection $f_{\Delta_1}(V_1) \cap f_{\Delta_2}(V_2)$ is empty. If it is not empty, then the procedure exits immediately. Otherwise, an *ec* subgraph isomorphism f is constructed from f_1 and f_2. The construction of an *ec* subgraph isomorphism $f = (\Delta, f_\Delta)$ from f_1 and f_2 requires that a set of edit operations Δ and a subgraph isomorphism f_Δ are generated on the basis of Δ_1, Δ_2 and f_{Δ_1}, f_{Δ_2}, respectively, such that f_Δ is a subgraph isomorphism from $\Delta(S_1 \cup_E S_2)$ to G_I. Due to the fact that the edit operations on the subgraphs S_1 and S_2 are contained in Δ_1 and Δ_2, it is easy to see that Δ is a concatenation of Δ_1, Δ_2 and the set Δ_E of edge operations on E, i.e.,

$$\Delta = \Delta_1 + \Delta_2 + \Delta_E \tag{2}$$

For the construction of Δ_E, three possible edge edit operations must be considered for each edge specified in E. First, for each edge $e = (v_i, w_j)$ in E there must be an edge $e_I = \left(f_{\Delta_1}(v_i), f_{\Delta_2}(w_j)\right) \in E_I$. If e_I exists and $\nu(e) \neq \nu_I(e_I)$ then, depending on the respective costs, either the label $\nu(e)$ is substituted by the label $\nu_I(e_I)$ or e is deleted and e_I is subsequently inserted. That is, if $C(\nu(e) \rightarrow \nu_I(e_I)) \leq C(e \rightarrow \$) + C(\$ \rightarrow e_I)$ then the substitution of $\nu(e)$ by $\nu_I(e_I)$ must be added to Δ_E. Otherwise, the deletion of e and the insertion of e_I must be added to Δ_E. Secondly, if e_I does not exist then the deletion of $e = (v_i, w_j)$ from the graph $S_1 \cup_E S_2$ must be added to Δ_E. Thirdly, if there is an edge $e_I = \left(f_{\Delta_1}(v_i), f_{\Delta_2}(w_j)\right)$ in the input but no corresponding edge $e = (v_i, w_j)$ is specified in E, then the insertion of e into $S_1 \cup_E S_2$ must be added to Δ_E. With this, $\Delta = \Delta_1 + \Delta_2 + \Delta_E$ contains all edit operations that are necessary such that there exists a subgraph isomorphism f_Δ from $\Delta(S_1 \cup_E S_2)$ to G_I. The subgraph isomorphism f_Δ is then defined as

$$f_\Delta(v) = \begin{cases} f_{\Delta_1}(v) & \text{if } v \in V_{\Delta_1} \\ f_{\Delta_2}(v) & \text{if } v \in V_{\Delta_2} \end{cases} \tag{3}$$

for all $v \in V_{\Delta_1} \cup V_{\Delta_2}$. Finally, in step 4, the *ec* subgraph isomorphism $f = (\Delta, f_\Delta)$ is output.

COMBINE(S_1, S_2, E, G_I, f_1, f_2)

1. let $f_1 = \left(\Delta_1, f_{\Delta_1}\right), f_2 = \left(\Delta_2, f_{\Delta_2}\right), \Delta_1(S_1) = \left(V_{\Delta_1}, E_{\Delta_1}, \mu_{\Delta_1}, \nu_{\Delta_1}\right), \Delta_2(S_2) = \left(V_{\Delta_2}, E_{\Delta_2}, \mu_{\Delta_2}, \nu_{\Delta_2}\right)$
2. If $f_{\Delta_1}(V_{\Delta_1}) \cap f_{\Delta_2}(V_{\Delta_3}) \neq \emptyset$ then exit
3. combine f_1 and f_2 to form an *ec* subgraph isomorphism $f = (\Delta, f_\Delta)$ from $S_1 \cup_E S_2$ to G_I, with
$$f_\Delta(v) = \begin{cases} f_{\Delta_1}(v) & \text{if } v \in V_{\Delta_1} \\ f_{\Delta_2}(v) & \text{if } v \in V_{\Delta_2} \end{cases}$$
for all $v \in V_{\Delta_1} \cup V_{\Delta_2}$ and
$$\Delta = \Delta_1 + \Delta_2 + \Delta_E$$
where Δ_E is the set of edit operations on the edges in E connecting S_1 and S_2 (see text).
4. output f

Fig. 3. Algorithm *combine*.

Problem (III) Given a decomposition D and an input graph G_I, *ec* subgraph isomorphisms for the models in D can be found by applying the solutions to the Problems I and II described above. Namely, for all subgraphs S in D that consist of a single vertex, all *ec* subgraph isomorphisms are first generated by calling the procedure *vertex_matching*. Next, all of these *ec* subgraph isomorphisms are combined to form *ec* subgraph iso-

morphisms for the larger subgraphs by calling the procedure *combine* until *ec* subgraph isomorphisms for the model graphs emerge. However, if all *ec* subgraph isomorphisms that are found are immediately used for combination regardless of their edit costs, their number will explode combinatorially. Thus, given a graph S that is decomposed into S_1, S_2, we propose to store the *ec* subgraph isomorphisms that are found for S_1 and S_2 locally, and then choose the *ec* subgraph isomorphisms with the least cost for combination. For this purpose, we introduce two ordered lists, *open* and *closed*, that are attached to each subgraph in D. Informally speaking, the list *closed* holds all *ec* subgraph isomorphisms that have already been used for combination. Furthermore, only *ec* subgraph isomorphisms that are stored in the list *closed* of the subgraph S_1 may be used for combination with *ec* subgraph isomorphisms for the corresponding subgraph S_2, and vice versa. The list *open*, on the other hand, holds all *ec* subgraph isomorphisms that have not yet been used for combination due to their costs. Thus, an *ec* subgraph isomorphism f_1 that is found for some subgraph S_1 is stored in $open(S_1)$ before it is used for further combinations. Let

$$C\left[open(S_1)\right] = \begin{cases} MIN_f\left\{C(f) \middle| f \in open(S_1)\right\} & \text{if } open(S_1) \text{ is not empty} \\ \infty & \text{otherwise} \end{cases} \quad (4)$$

denote the least cost of any *ec* subgraph isomorphisms in $open(S_1)$. Now, the next *ec* subgraph isomorphism that is to be used for combination is selected according to $C[open(S_1)]$. If $C(f_1) = C[open(S_1)]$ is minimal over all subgraphs S_i in D then f_1 is removed from $open(S_1)$ and stored in the list $closed(S_1)$. Analogously to $C[open(S_1)]$, let

$$C\left[closed(S_1)\right] = \begin{cases} MIN_f\left\{C(f) \middle| f \in closed(S_1)\right\} & \text{if } closed(S_1) \text{ is not empty} \\ \infty & \text{otherwise} \end{cases} \quad (5)$$

denote the least cost of any *ec* subgraph isomorphism in $closed(S_1)$. After f_1 has been removed from $open(S_1)$ and stored in $closed(S_1)$ it is combined with all *ec* subgraph isomorphisms that were found for S_2. More precisely, f_1 is combined with all *ec* subgraph isomorphisms that are already stored in $closed(S_2)$ (but not with those stored in $open(S_2)$). This process guarantees that new *ec* subgraph isomorphisms are always built from *ec* subgraph isomorphisms with minimal cost.

Based on the procedures *vertex_matching* and *combine* and the lists *open* and *closed*, the new method for *oec* subgraph isomorphism detection is now formally described in Fig. 4. The input to the algorithm *NSG* consists of the decomposition D which represents the set of model graphs G_1, …, G_N and the input graph G_I. In the beginning, $open(S)$ and $closed(S)$ are empty for each S in D. In step 1 all *ec* subgraph

NSG(D, G_I)

1. for all S in D with $S = (V_s, E_s, \mu_s, \nu_s)$ and $|V_s| = 1$, $\{v\} = V_s$.
   a) F = vertex_matching(v, $\mu_s(v)$, G_I)
   b) $open(S) = F$
2. choose S_1 such that $C[open(S_1)] + h(S_1) \le T$ and minimal in D. If no such S_1 exists, goto 7.
3. choose $f_1 \in open(S_1)$ such that $C(f_1)$ is minimal in $open(S_1)$; remove f_1 from $open(S_1)$ and set $closed(S_1) = closed(S_1) \cup \{f_1\}$.
4. if S_1 is a model graph G_i then reset the acceptance threshold to $T = C(f_1)$.
5. for all $(S, S_1, S_2, E) \in D$ or $(S, S_2, S_1, E) \in D$ do
   a) for all $f_2 \in closed(S_2)$
      i) f = combine(S_1, S_2, E, G_I, f_1, f_2)
      ii) if f is not empty then add f to $open(S)$, i.e., $open(S) = open(S) \cup \{f\}$
6. goto 2.
7. for each model G_i in D, $i = 1 \ldots, N$ output $closed(G_i)$.

Fig. 4. Algorithm *NSG*.

isomorphisms for subgraphs S consisting of a single vertex are generated and stored in $open(S)$ by calling the procedure *vertex_matching*. Next, in step 2, a subgraph S_1 is selected such that $C[open(S_1)] + h(S_1)$ is smaller than a given acceptance threshold T and minimal over all subgraphs in D. At first, the acceptance threshold T is set to the maximal cost that any *ec* subgraph isomorphism from the models to the input may have. Note that this maximal cost is strongly application dependent. It may be set to infinity. Later, in step 4, it will be reset to the cost of some *oec* subgraph isomorphism actually found for any of the models. In addition to the actual costs $C[open(S_1)]$ there is also an evaluation function $h(S_1)$ taken into account. For the moment, we assume that $h(S_1)$ is constant and set to zero for all S, i.e., $h(S) = 0$ for all S in D. (For details of $h(S_1)$ see [14]). After selecting a subgraph S_1 for which the condition in step 2 holds, the *ec* subgraph isomorphism $f \in open(S_1)$ with costs $C(f_1) = C[open(S_1)]$ is removed from $open(S_1)$ and added to the list $closed(S_1)$ in step 3. Next, in step 4 it is checked whether S_1 is one of the model graphs G_i. If this is the case, f_1 represents the *oec* subgraph isomorphism from G_i to G_I. This follows directly from the selection process in steps 2 and 3. By always selecting the *ec* subgraph isomorphism f_1 with the minimal costs, it is guaranteed that no other *ec* subgraph isomorphism for any of the graphs or subgraphs in the decomposition D has cost less than $C(f_1)$. Hence, the *ec* subgraph isomorphism f_1 for the model graph G_i is optimal. For some applications, it may be sufficient to find only one *oec* subgraph isomorphism. In that case, the algorithm can be terminated if the condition in step 4 is satisfied. Otherwise, if all *oec* subgraph isomorphisms with equal cost are to be found (as it is assumed here), the acceptance threshold T can be reset to $C(f_1)$ in step 4. This guarantees that the

TABLE 1
PARAMETERS OF THE EXPERIMENTS

Experiments	Parameters for the graph generation						
Nr.	Vertices	Edges	Labels	Error	Database	C_s	C_d,C_e
1	10 - 36	15 - 48	10	1 - 10	1	1	5
2[a]	20	30	10	10	1 - 10	1	5
3	20	30	3 - 10	1 - 7	1	1	5
4[b]	20 - 40	30 - 60	10	1 - 10	1	1	5
5	10 - 100	15 - 150	20	5	1	1	5
6	20	30	10	5	1 - 100	1	5

a. The size of the common subgraph of the models was increased from five to 18 vertices.

b. Only the size of the input graph was varied, while the size of the model graph was fixed at 20 vertices.

algorithm only searches for *ec* subgraph isomorphisms with costs equal to the cost of the *oec* subgraph isomorphism already found. In step 5, new *ec* subgraph isomorphisms are generated by means of combination. All graphs S which are decomposed into S_1 and some subgraph S_2, i.e., (S, S_1, S_2, E) or (S, S_2, S_1, E) $\in D$, are selected. Then, each *ec* subgraph isomorphism f_2 in *closed*(S_2) is combined with f_1 in order to form an *ec* subgraph isomorphism f from S to G_I. For this purpose, the procedure *combine* is called with S_1, S_2, the connecting edges E, the input graph G_I and the *ec* subgraph isomorphisms f_1, f_2 as arguments. If *combine* is successful, an *ec* subgraph isomorphism f from S to G_I is returned and stored in *open*(S). When all *ec* subgraph isomorphisms f_2 in *closed*(S_2) have been combined with f_1, the process continues in step 2. Finally, when there are no more subgraphs S_1 with $C[open(S_1)] + h(S_1) \leq T$, all *oec* subgraph isomorphisms that were found for the models $G_1, \ldots, G_N$ are output in step 7 and the algorithm terminates.

For details of the lookahead procedure see [14].

4 EXPERIMENTAL RESULTS

For a theoretical computational complexity analysis of the new algorithm see [14]. In order to study the behavior of the new algorithm in practice, we performed a number of experiments with randomly generated graphs. The new algorithm (NSG) was implemented in C++ and run on a SunSparc Workstation. For comparison reasons, we also implemented a traditional A^*-based algorithm (TA) with the lookahead procedure described in [25]. The graphs used in the experiments were randomly generated on the basis of the following quantities:

- the number of vertices,
- the number of edges,
- the number of different vertex labels (all edges were unlabeled),
- the number of graphs in the database,
- the degree of label distortion, and
- the number of missing vertices.

For each experiment, a set of labeled model graphs consisting of a specified number of vertices and edges was randomly generated. Each vertex in a model graph was randomly assigned a label out of a finite subset of the integers. From each model graph a corresponding input graph was derived by copying and permuting the vertices and edges. Then, the input graph was subject to label distortion by randomly changing the labels of a specified number of vertices. Finally, a specified number of vertices were deleted from the input graph along with their incident edges. As mentioned before, an important factor in any practical application of *ec* subgraph isomorphism algorithms is the definition of the costs of the edit operations. In all our experiments, the cost for substituting a label l_1 with a label l_2 was set to $C_s|l_1 - l_2|$, where $|l_1 - l_2|$ denotes the absolute difference between l_1 and l_2, and C_s is a constant. The cost of inserting a vertex and inserting or deleting an edge was constantly set to C_d and C_e, respectively.

The objective of each experiment was to measure and compare the computation times needed by NSG and TA in detecting the optimal error-correcting subgraph isomorphism from any of the model graphs to the input graph. Each measurement was repeated 20 times with different graphs and the average time was taken as the final result in order to account for the random nature of the graph generation process. In Table 1, an overview of the experiments is given. The cost C_s of substituting labels was set to one and the cost of inserting and deleting edges or vertices was set to five.

In the first experiment, we examined the influence of the size of the model graphs and the number of label distortions on the run time performance of NSG and TA. In the beginning, a single model graph consisting of 10 vertices and 15 edges was randomly generated. From this model graph an input graph was derived by copying its vertices and edges and distorting its vertex labels according to the specified number of label errors. Note that the model and the input graph were structurally isomorphic and only the labels were distorted. The number of label errors was gradually increased from one to 10 and for each setting a new input graph was generated. The number of vertices in the model and input graph was also increased from 10 to 36 along with the number of edges that was increased from 15 to 48. There were a total of 10 vertex and no edge labels used in this experiment. Hence, in a graph of 30 vertices, there were on average three vertices with identical labels. The result of the first experiment is displayed in Fig. 5. The lower (upper) plane in the right corner denotes the times of NSG (TA). Clearly, NSG outperforms TA for a growing model graph size and also for a growing number of label errors in the input graph.

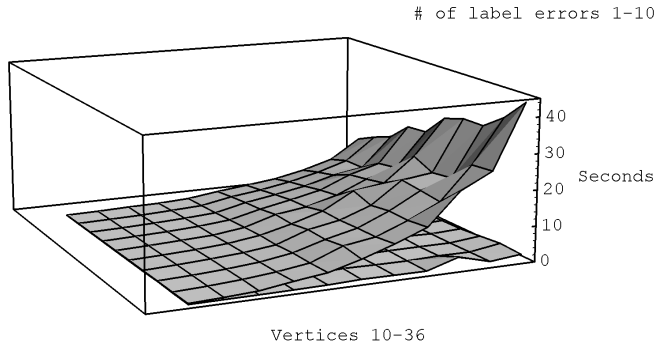


Fig. 5. Time in seconds for an increasing number of vertices in the model graphs and an increasing number of label errors in the input graph (lower plane in the right corner denotes NSG, upper plane denotes TA).

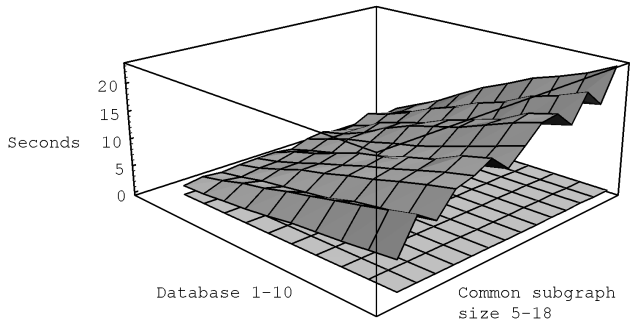


Fig. 6. Time in seconds for a growing number of model graphs in the database and a growing common subgraph of the model graphs (lower plane in the right corner denotes the time of NSG, upper plane denotes the time of TA).

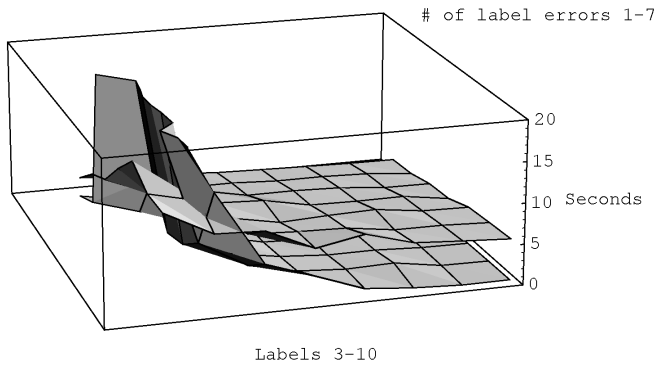


Fig. 7. Time in seconds for a growing number of labels and a growing number of label errors (lower plane in the right corner denotes the time of NSG, upper plane in the right corner denotes the time of TA).

In the experiment described above, there was only one model graph in the database. Consequently, NSG's ability to share common substructures among different model graphs could not be studied. Therefore, in the second experiment, we varied both the number of model graphs in the database and the size of the subgraph that was common to all the model graphs. Each of the model graphs consisted of 20 vertices and 30 edges. The size of the database was gradually increased from one to 10 model graphs. For each size of the database, we also varied the size of the common

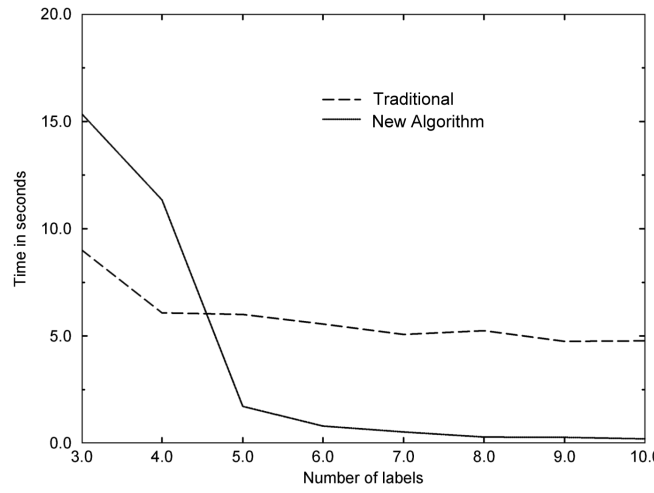


Fig. 8. Cut through Fig. 7 along the label axis with a constant label error of four.

subgraph starting with five vertices and ending with 18 vertices. Thus for a database with 10 model graphs and a common subgraph of size 18, each model graph consisted of an identical subgraph with 18 vertices and 27 edges and a unique subgraph with two vertices and a single edge. The number of distorted labels was set to 10. In Fig. 6, the average computation times for the second experiment are given. TA is linearly dependent on the size of the database, while NSG has only sublinear dependency.

So far, the number of available labels for the model and the input graphs was constantly kept at 10. Previous research conducted by the authors has shown that the exact version of NSG (where no graph edit operations are allowed) performs poorly when the number of labels is small [14]. Therefore, in the third experiment, we examined the influence of the number of labels on the performance of the algorithms. There was exactly one model graph consisting of 20 vertices and 30 edges in the database. The number of labels was varied between three and 10. Thus, in the beginning, there were on the average approximately seven vertices in the model graph with identical labels. In the end, for 10 labels, there were on the average two vertices with identical labels. The edges were all unlabeled. The number of distorted vertex labels in the input graph was also varied between one and seven. The results of this experiment are documented in Fig. 7. Additionally, in Fig. 8, a cut through Fig. 7 along the axis of the number of labels is given for a constant label error of five. For a decreasing number of labels, the performance of NSG became rapidly worse, i.e., for four labels NSG was already slower than TA.

In all of the experiments described so far, the model and the input graphs were always of equal size. Thus, we were in fact looking for *ec* graph isomorphisms. In the fourth experiment, we explicitly increased the size of the input graph from 20 to 40 vertices along with the edges that were increased from 30 to 60 while the size of the model graph remained constant at 20 vertices and 30 edges. At the same time, the number of errors in the labels of the input graph was varied between one and 10. The

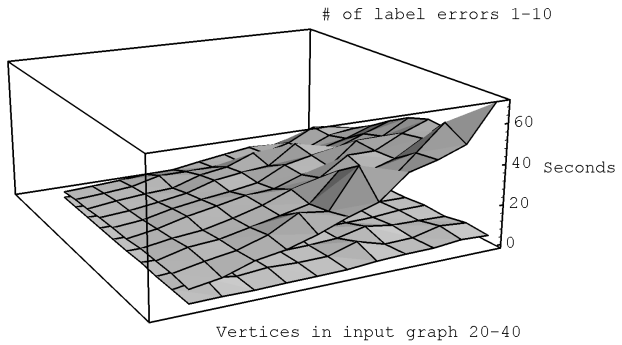


Fig. 9. Time in seconds for a growing number of vertices in the input graph and a growing number of label errors (lower plane in the right corner denotes the time of NSG, upper plane in the right corner denotes the time of TA).

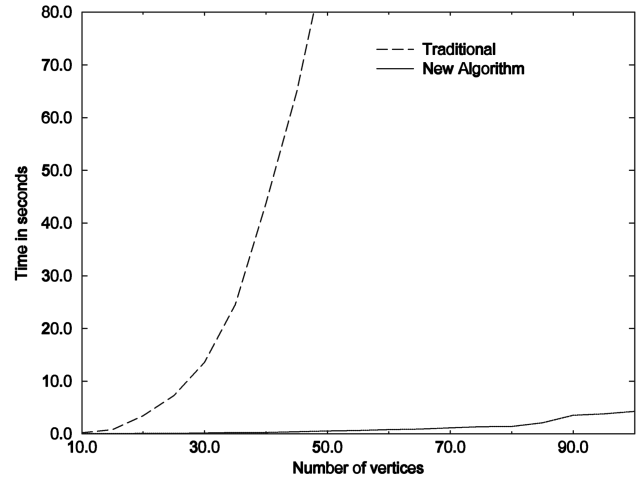


Fig. 10. Time in seconds for an increasing number of vertices in the model graphs and a constant label error of five.

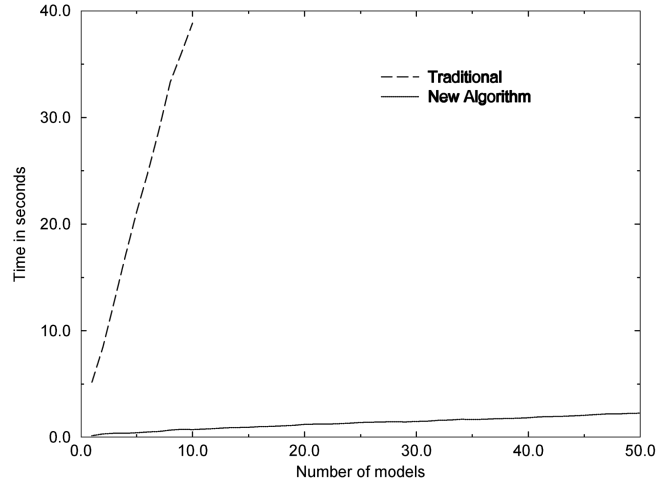


Fig. 11. Time in seconds for an increasing number of models in the database and a constant label error of five.

results of this experiment are given in Fig. 9. We observe that TA again required more time than NSG for both an increasing error and an increasing input graph size.

The previous experiments have shown that NSG is very efficient for graphs with up to 20 vertices and databases containing at most 10 model graphs. In order to explore the limits of NSG with respect to the size of the model graphs and the size of the database, we performed two more experiments. In the fifth experiment, documented in Fig. 10, the size of the model and input graph was increased from 10 to 100 vertices along with the edges that were increased from 15 to 150. There were 20 different vertex labels available and the labels of five input graph vertices were distorted by one unit. Notice that the computation time of TA is only plotted for graphs with less than 50 vertices for which it already required more than 80 seconds while NSG required in the same situation only 0.5 second. In the limit, when the model graph consisted of 100 vertices and 150 edges, NSG required only five seconds on the average.

In the sixth experiment, documented in Fig. 11, we tested the behavior of NSG for large databases of graphs. We generated a growing database of model graphs starting at one and ending at 50 model graphs. Each graph consisted of 20 vertices and 30 edges. Unlike the second experiment, where a common subgraph was explicitly defined, the graphs in this experiment were generated independently of each other. However, the total number of labels was restricted to 10 such that common subgraphs of various sizes naturally evolved. As was to be expected, NSG's performance was only sublinearly dependent on the size of the database. While TA required 40 seconds for a database containing 10 models, NSG finished within three seconds for a database containing 50 models.

Further experiments are described in [14].

5 Discussion and Conclusions

In this paper, we introduced a new method for the computation of optimal error-correcting subgraph isomorphisms from a set of a priori known model graphs to an input graph. The novelty of our approach is the off-line preprocessing of the database of model graphs. In this preprocessing step, the model graphs are compiled into a representation in which common subgraphs of different model graphs are stored only once. At run time, this representation is used to detect the optimal error-correcting subgraph isomorphism from any of the model graphs to the input graph in time that is only sublinearly dependent on the database size. Furthermore, the compact representation of the model graphs allows the implementation of an efficient future cost estimation function.

In the practical experiments, it was confirmed that the new algorithm is more efficient than the conventional algorithm in many cases. In particular, the new algorithm outperforms the conventional algorithm in situations where the model graphs consist of labeled vertices and the cost of the edit operations adequately model the actual label distortions and structural errors. However, the new algorithm is badly fitted for situations where the model graphs are unlabeled or only labeled in the edges, and where unlikely distortions occur. Nonetheless, even in these situations its ability to compactly represent the model graphs remains an advantage such that it eventu-

ally outperforms the conventional algorithm when the database of models grow very large.

The new method was described for unrestricted, general graphs and a set of edit operations that are powerful enough to correct any kind of graph distortion. Consequently, the adaption of the new method to specific applications is straightforward. Furthermore, due to the principle of dividing the model graphs into smaller subgraphs, there is a potential for parallelization inherent in the new method.

A first application in the field of document image analysis, i.e., graphical symbol recognition and automatic symbol acquisition, was presented by the authors in [13]. The drawings and the graphical symbols in this application are represented by attributed relational graphs. Instances of known symbols are detected in the drawing by determining subgraph isomorphisms from these symbol graphs to the graph representing a complete input drawing. Due to the fact that the symbol graphs are represented by the compact decomposition structure described in the present paper, the time complexity of the method is only sublinearly dependent on the number of different symbols. Additionally, the proposed application also contains a learning component. This learning component is closely connected to the graph matching procedure. It makes use of the fact that the database of known symbols can be incrementally updated. For example, new symbols can be added to the database during runtime without the need to recompile the database from scratch.

ACKNOWLEDGMENT

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

REFERENCES

[1] H. Bunke and G. Allerman, "Inexact Graph Matching for Structural Pattern Recognition," *Pattern Recognition Letters*, vol. 1, no. 4, pp. 245-253, 1983.

[2] J. Ben-Arie and A.Z. Meiri, "3D-Object Recognition by Optimal Matching Search of Multinary Relation Graphs," *Computer Vision, Graphics, and Image Processing*, vol. 37, pp. 345-361, 1987.

[3] W.J. Christmas, J. Kittler, and M. Petrou, "Structural Matching in Computer Vision Using Probabilistic Relaxation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 17, no. 8, pp. 749-764, Aug. 1995.

[4] M.S. Costa and L.G. Shapiro, "Analysis of Scenes Containing Multiple Non-Polyhedral 3D Objects," C. Braccini, L. DeFloriani, and G. Vernazza, eds., *Lectures Notes in Computer Science 974: Image Analysis and Processing*. Springer Verlag, 1995.

[5] M.A. Eshera and K.S. Fu, "A Graph Distance Measure for Image Analysis," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 14, no. 3, pp. 398-408, May 1984.

[6] J. Feng, M. Laumy, and M. Dhome, "Inexact Matching Using Neural Networks," E.S. Gelsema and L.N. Kanal, eds., *Pattern Recognition in Practice IV: Multiple Paradigms, Comparative Studies, and Hybrid Systems*, pp. 177-184. North-Holland, 1994.

[7] C.L. Forgy, "Rete, A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *Artificial Intelligence*, vol. 19, pp. 17-37. Elvesier, 1982.

[8] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman and Company, 1979.

[9] J. Kittler, W.J. Christmas, and M. Petrou, "Probabilistic Relaxation for Matching of Symbolic Structures," H. Bunke, ed., *Advances in Structural and Syntactic Pattern Recognition*, pp. 471-480. World Scientific, 1992.

[10] S.W. Lee, J.H. Kim, and F.C.A. Groen, "Translation- Rotation- and Scale Invariant Recognition of Hand-Drawn Symbols in Schematic Diagrams," *Int'l J. Pattern Recognition and Artificial Intelligence*, vol. 4, no. 1, pp. 1-15, 1990.

[11] S.W. Lu, Y. Ren, and C.Y. Suen, "Hierarchical Attributed Graph Representation and Recognition of Handwritten Chinese Characters," *Pattern Recognition*, vol. 24, pp. 617-632, 1991.

[12] H.S. Lee and M.I. Schor, "Match Algorithms for Generalized Rete Networks," *Artificial Intelligence*, pp. 255-270, 1992.

[13] B.T. Messmer and H. Bunke, "Automatic Learning and Recognition of Graphical Symbols in Engineering Drawings," K. Tombre and R. Kasturi, eds., *Graphics Recognition, Lecture Notes in Computer Science*, vol. 1,072, pp. 123-134. Springer Verlag, 1996.

[14] B.T. Messmer, "Efficient Graph Matching Algorithms for Preprocessed Model Graphs," PhD thesis, Institut für Informatik und angewandte Mathematik, Universität Bern, Switzerland, 1995.

[15] N.J. Nilsson, *Principles of Artificial Intelligence*. Palo Alto, Calif.: Tioga, 1980.

[16] K. Sengupta and K.L. Boyer, "Organizing Large Structural ModelBases," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 17, no. 4, Apr. 1995.

[17] A. Sanfeliu and K.S. Fu, "A Distance Measure Between Attributed Relational Graphs for Pattern Recognition," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 13, pp. 353-363, 1983.

[18] L.G. Shapiro and R.M. Haralick, "Structural Descriptions and Inexact Matching," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 3, pp. 504-519, 1981.

[19] L.G. Shapiro and R.M. Haralick, "Organization of Relational Models for Scene Analysis," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 4, pp. 595-602, 1982.

[20] H. Sossa and R. Horaud, "Model Indexing: The Graph-Hashing Approach," *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp. 811-814, 1992.

[21] D.S. Seong, H.S. Kim, and K.H. Park, "Incremental Clustering of Attributed Graphs," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 23, no. 5, pp. 1,399-1,411, 1993.

[22] W.H. Tsai and K.S Fu, "Error-Correcting Isomorphisms of Attributed Relational Graphs for Pattern Recognition," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 9, pp. 757-768, 1979.

[23] R.A. Wagner and M.J. Fischer, "The String-to-String Correction Problem," *J. Assoc. Computing Machinery*, vol. 21, no. 1, pp. 168-173, 1974.

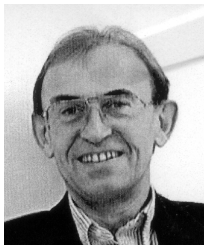
[24] E.K. Wong, "Three-Dimensional Object Recognition by Attributed Graphs," H. Bunke and A. Sanfeliu, eds., *Syntactic and Structural Pattern Recognition- Theory and Applications*, pp. 381-414. World Scientific, 1990.

[25] A.K. Wong, M. You, and A.C. Chan, "An Algorithm for Graph Optimal Monomorphism," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 20, no. 3, pp. 628-636, 1990.

[26] L. Xu and E. Oja, "Improved Simulated Annealing, Boltzmann Machine, and Attributed Graph Matching," L. Almeida, ed., *Lecture Notes in Computer Science*, vol. 412, pp. 151-161. Springer Verlag, 1990.

Bruno T. Messmer received his master's in computer science in 1992 from the University of Berne for his work on applying the RETE algorithm to graph matching. For this work, he won the 1992 IBM Switzerland artificial intelligence prize. After having worked for one year at a software company designing financial real-time applications, he continued his work on graph matching at the University of Berne. In 1995, he received a PhD in computer science for developing new algorithms for efficient graph matching from the University of Berne. Dr. Messmer is an artificial intelligence expert currently working for Swiss Telecom in the domain of network planning, design, optimization, and the development of object-oriented software frameworks. His main interests include graph matching, pattern recognition, case-based reasoning, and search algorithms.

Dr. Messmer has published more than 16 articles. He has also implemented a general graph-matching toolkit in C++ that has been integrated in a number of different applications. He can be contacted via Bruno.Messmer@swisscom.com or messmer@iam.unibe.ch.



Horst Bunke received his MS and PhD degrees in computer science from the University of Erlangen, Germany in 1974 and 1979, respectively. He was a member of the scientific staff at the University of Erlangen from 1974 to 1984. From 1980 to 1981, he was on a postdoctoral leave visiting Purdue University, West Lafayette, Indiana, and, in 1983, he held a temporary appointment at the University of Hamburg, Germany. In 1984, he joined the University of Bern, Switzerland, where he is a full professor in the Computer Science Department. He was department chairman from 199201996. In 1997, he became Dean of the Faculty of Science. Dr. Bunke held visiting positions at the IBM Los Angeles Scientific Center (1989), the University of Szeged, Hungary (1991), the University of South Florida at Tampa (1991 and 1996), the University of Nevada at Las Vegas (1994), and Kagawa University, Takamatsu, Japan (1995).

Dr. Bunke is a fellow the the International Association for Pattern Recognition (IAPR), editor-in-charge of the *International Journal of Pattern Recognition and Artificial Intelligence*, and editor-in-chief of the book series on *Machine Perception and Artificial Intelligence* by World Scientific Publishing Co. He was on the program and organization committee of many conferences and served as a referee for numerous journals and scientific organizations. He has more than 250 publications, including 20 books. He is a member of the AAAI, the IEEE Computer Society, the Pattern Recognition Society, the European Association for Signal Processing, and other scientific organizations. His current interests include pattern recognition, machine vision, and artificial intelligence.