# Efficient Subgraph Isomorphism Detection: A Decomposition Approach

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

**Abstract**—Graphs are a powerful and universal data structure useful in various subfields of science and engineering. In this paper, we propose a new algorithm for subgraph isomorphism detection from a set of a priori known model graphs to an input graph that is given online. The new approach is based on a compact representation of the model graphs that is computed offline. Subgraphs that appear multiple times within the same or within different model graphs are represented only once, thus reducing the computational effort to detect them in an input graph. In the extreme case where all model graphs are highly similar, the run-time of the new algorithm becomes independent of the number of model graphs. Both a theoretical complexity analysis and practical experiments characterizing the performance of the new approach will be given.

**Index Terms**—Graph matching, graph isomorphism, subgraph isomorphism, preprocessing.

## 1 Introduction

Graphs are a powerful and universal data structure useful in various subfields of science and engineering. When graphs are used for the representation of objects in some domain, the problem of comparing different objects to each other can be formulated as the search for correspondences between the attributed graphs representing the objects. Such correspondences can be established by isomorphism or subgraph isomorphism detection.

The concept of subgraph isomorphism detection has been applied in a variety of fields. For example, it has been used in chemical documentation systems for the comparison of molecular structures [1], in case-based-reasoning for the retrieval of cases from the case base [2], [3], in semantic networks in combination with graph grammars [4], or in machine learning, where it is used in order to learn common substructures of different concepts [5], [6], [7], [8]. In pattern recognition, subgraph isomorphism detection was applied to Chinese character recognition [9], the interpretation of schematic diagrams [10], [11], and seal verification [12]. It was also combined with evidence-based systems for shape analysis [13]. In computer vision, subgraph isomorphism detection was used for the localization of 3D objects in images [14], [15], [16], [17], [18] and in robot vision [19].

The main problem with subgraph isomorphism detection is the fact that it is an NP-complete problem [20]. In other words, the time to detect a subgraph isomorphism between two graphs is in the worst case exponential in the number of vertices of these graphs. In the following, we give a short review of methods for subgraph isomorphism detection that have been proposed in the past.

The most common technique to establish a subgraph isomorphism is based on backtracking in a search tree. In order to prevent the search tree from growing unnecessarily large, different refinement procedures such as the one by Ullman [21], forward-checking and looking-ahead [22], or discrete relaxation [23] have been proposed. These techniques are fairly stable and perform well in most cases. Another approach is taken in [24], [25], where each possible mapping of a vertex of the first graph onto a vertex of the second graph is recorded in an association graph and the detection of a possible graph match is performed by maximal clique detection. Finally, in [26], it is proposed to partition the graphs according to lattice theory in order to reduce the computational complexity of the subgraph isomorphism problem. All these methods provide an optimal solution to the graph matching problem, but may in the worst case become computationally intractable.

Continuous optimizations methods, on the other hand, aim at providing a solution within a reasonable time. However, they may not always find the optimal solution, i.e., the mapping of model graph vertices to input graph vertices found by a continuous optimization method not necessarily represents a subgraph isomorphism. In [27], the advantages and disadvantages of continuous optimization methods such as neural networks compared to the optimal backtracking methods are examined. Other continuous optimization approaches include the application of simulated annealing [28], genetic algorithms [29], [30], [31], and probabilistic relaxation [32].

The methods for subgraph isomorphism detection mentioned so far work on only two graphs at a time. However, in many applications there is more than one model graph that must be matched with the input graph. Consequently, it is necessary to apply the subgraph isomorphism algorithm to each model-input pair, resulting in a computation time that is linearly dependent on the size of the database. This dependency may become prohibitive if the number of model graphs is large. Hence, some form of organization or indexing on the database of model graphs is needed. In [33], [34], [35], [36], a hierarchical organization of the database is proposed which clusters the model graphs into similarity classes. The given input graph is then used to

• *The authors are with the Institut für Informatik und angewandte Mathematik, University of Bern, Neubrückstr. 10, CH-3012 Bern, Switzerland. E-mail: {messmer, bunke}@iam.unibe.ch.*

*Manuscript received 1 Dec. 1995; revised 30 Jan 1996; accepted 5 Mar. 1998. For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 104345.*

index the database of model graphs by traversing the hierarchy. However, such an indexing can only work if the input graph is isomorphic to one of the model graphs. If the input graph is larger than the models or contains more than one instance of a model graph, the indexing process will not work. Another hierarchical approach which does not have this drawback was proposed in [37], where at the root of the hierarchy there is a supergraph, consisting of different subgraphs of the model graphs. At run-time, the input graph is matched with the supergraph and the hierarchy is traversed according to this initial match. The disadvantage of this method, 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 subgraph isomorphism detection process between a model graph and the input. Finally, some interesting approaches have been applied in the domain of computer vision, where multilevel indexes are computed by precalculating all possible views of an object into a view-description network [38], [39]. This network can then be used to efficiently index the database of model graphs. However, the scheme has not yet been generalized to arbitrary graphs.

Many of the reviewed algorithms above have interesting properties. However, no technique has been described, which solves the problem of subgraph isomorphism detection and the organization of large graph databases at the same time for general labeled graphs. In this paper, we propose a new approach to the problem of subgraph isomorphism detection from a set of model graphs to an input graph. Our algorithm is somewhat similar to the RETE algorithm for forward-chaining, rule-based systems [40]. It is based on a compact representation of the model graphs. The representation is created offline by decomposing the model graphs into a set of subgraphs. These subgraphs are the basic elements of the new representation. If a subgraph in the decomposition appears multiple times in the same or in different model graphs, it is represented only once. At run-time, the new representation is used to efficiently detect subgraph isomorphisms from the models to the input graph in the following manner. First, subgraph isomorphisms for the subgraphs in the representation are detected. These subgraph isomorphisms are then recursively combined to form subgraph isomorphisms for the complete model graphs. Due to the fact that common subgraphs of different models are represented only once, they are matched exactly once with the input graph. Thus, the new algorithm is only sublinearly dependent on the number of the model graphs.

The rest of this paper is organized in the following manner: In Section 2, we provide basic definitions and notations. In Section 3, the new algorithm is described in detail. In Section 4, the computational complexity of the new algorithm is analyzed. The results of the theoretical complexity are confirmed in a number of practical experiments documented in Section 5. Finally, we conclude the paper with a summary and some remarks on the applicability of the new algorithm in various domains. In Appendix A, Ullman's algorithm which is used as a benchmark in this paper is briefly described, along with a computational complexity analysis.

## 2 Definitions and Notations

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 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.*

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 $(v_2, v_1) \in E$ for any edge $(v_1, v_2) \in E$. The empty graph, i.e., the graph with an empty set of vertices, will be denoted by $\emptyset$.

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

1. $V_s \subseteq V$
2. $E_s = E \cap (V_s \times V_s)$
3. *$\mu_s$ and $\nu_s$ are the restrictions of $\mu$ and $\nu$ to $V_s$ and $E_s$, respectively, i.e.,*

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

$$\nu_s(e) = \begin{cases} \nu(e) & \text{if } e \in E_s \\ \text{undefined} & \text{otherwise.} \end{cases}$$

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 $G = (V, E, \mu, \nu)$ and a subgraph $S = (V_s, E_s, \mu_s, \nu_s)$ of $G$, the* difference *of $G$ and $S$ is the subgraph of $G$ that is defined by the set of vertices $V - V_s$. We denote the difference of $G$ and $S$ by $G - S$.*

**Definition 4.** *Given two graphs $G_1 = (V_1, E_1, \mu_1, \nu_1)$, $G_2 = (V_2, E_2, \mu_2, \nu_2)$, 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 $\nu : E' \rightarrow L_E$, the* union *of $G_1$ and $G_2$ with respect to $E'$ is the graph $G = (V, E, \mu, \nu)$ 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. $$\nu(e) = \begin{cases} \nu_1(e) & \text{if } e \in E_1 \\ \nu_2(e) & \text{if } e \in E_2 \\ \nu(e) & \text{if } e \in E' \end{cases}$$

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 $f : V \to V'$ is a* graph isomorphism *from a graph $G = (V, E, \mu, \nu)$ to a graph $G' = (V', E', \mu', \nu')$ 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 $\nu(e) = \nu(e')$ 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 $\nu(e') = \nu(e)$.*

**Definition 6.** *An injective function $f : V \to V'$ 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 $S$.*

Notice that 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* that is given online. The input and model graphs will be also referred to as input and models, for short. The problem to be solved is to find all subgraph isomorphisms from the models to the input.

## 3 Decomposition-Based Subgraph Isomorphism

### 3.1 Overview of the Method

Given a set of model graphs $G_1, \ldots, G_N$ and an input graph $G_I$, we want to find all subgraph isomorphisms from any of the models to the input graph. Under a naive strategy, we would match the input graph sequentially to each model using, for example, Ullman's algorithm. The main disadvantage of this approach is that it is linearly dependent on the number of model graphs. Moreover, it is inefficient if different model graphs have common substructures, because these substructures will be matched with the input graph for each model repeatedly. In order to overcome this inefficiency, we propose a different approach.

Instead of matching each model graph individually onto the input graph, we recursively decompose the model graphs offline into smaller subgraphs. At run-time, these subgraphs are matched onto the input graph and all detected subgraph isomorphisms are combined to form subgraph isomorphisms for complete model graphs. This idea is similar to the RETE matching algorithm for forward chaining production systems [40], [41]. The main advantage of this scheme is that subgraphs that appear multiple times in the same or in different model graphs must be matched only once onto the input. Consequently, the corresponding subgraph isomorphism detection process will be more efficient than the sequential matching of the input graph with each of the models.

The new approach consists of two parts. First, there is an offline process in which the model graphs are recursively decomposed and the resulting subgraphs are represented by a special data structure. The second part is an online process, in which an input graph is matched with the model graphs, which are represented by the data structure generated in the first step. In the following, we first describe the offline decomposition of the model graphs and the data structure for their representation. Next, the new subgraph isomorphism algorithm that is based on this representation and an example will be given.

### 3.2 Decomposing the Model Graphs

The main idea of the new approach is to recursively decompose the model graphs into smaller subgraphs in an offline processing step. At run-time, the subgraph isomorphism problem is solved in a divide-and-conquer fashion. That is, we first look for subgraph occurrences of parts of the models in the input graph. All such occurrences are then successively combined to form subgraph isomorphisms for the complete models.

**Definition 7.** *Let $B = \{G_1, \ldots, G_N\}$ be a set of model graph. A* decomposition *of $B$, $D(B)$, is a finite set of 4-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 4-tuple $(G, G', G'', E) \in D(B)$ with $G = G_i$; $i = 1, \ldots, N$.*
4. *For each 4-tuple $(G, G', G'', E) \in D(B)$ there exists no other 4-tuple $(G_1, G'_1, G''_1, E_1) \in D(B)$ with $G = G_1$.*
5. *For each 4-tuple $(G, G', G'', E_1) \in D(B)*

   a. *if $G'$ consist of more than one vertex, then there exists a 4-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 4-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 4-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 4-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 4-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 7). Condition 3, in Definition 7, 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 $G_i, G_j, \ldots$ 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 4-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.

The decomposition of a set of models will be used to guide the search for subgraph isomorphisms from the models to the input. If there is a 4-tuple $(G, G', G'', E)$ in $D(B)$, then subgraph isomorphisms from $G'$ and $G''$ to the

DECOMPOSITION($B$)

1. let $B = \{G_1, \ldots, G_N\}$ and $D(B) = \emptyset$
2. for $i = 1$ to $N$
   decompose($G_i$)

Fig. 1. Algorithm *decomposition*.

DECOMPOSE($G$)

1. let $D$ be a decomposition and $S_{max} = \emptyset$.
2. if $G$ consists of only one vertex then exit.
3. for all $(G_i, G_i', G_i'', E_i)$
   if $G_i$ is a subgraph of $G$ and $S_{max}$ is smaller than $G_i$ then let $S_{max} = G$.
4. if $S_{max}$ is isomorphic to $G$ then exit.
5. if no subgraph $S_{max}$ was found and $G$ consists of more than one vertex then
   (a) choose randomly a subgraph $S_{max}$ of $G$.
   (b) decompose($S_{max}$)
6. decompose($G - S_{max}$)
7. add $(G, S_{max}, G - S_{max}, E)$ to $D$ where $E$ is the set of edges between $S_{max}$ and $G - S_{max}$ in $G$.

Fig. 2. Procedure *decompose*.

input will be searched for first. Once such subgraph isomorphisms have been found, they will be combined, whenever possible, into subgraph isomorphisms from $G$ to the input. This procedure is started with subgraphs $G'$ and $G''$ that consist of single vertices only and is recursively continued until the level is reached where $G$ represents a complete model.

Apparently, there exist many different decompositions for a given set of models. This property holds even in the case where the set of models consists of only a single graph. One could now define an optimal decomposition being, for example, one that contains the minimum number of 4-tuples, or one where the largest subgraph $G$ that occurs in all models is represented by a 4-tuple $(G, G', G'', E)$. However, the computation of such an optimal decomposition is a highly exponential problem [7]. In this paper, we propose a decomposition algorithm which usually does not generate an optimal decomposition, but is computationally inexpensive.

In Fig. 1, the algorithm *decomposition* is displayed. The input to the algorithm is a set $B$ of models that are to be decomposed and represented by the decomposition $D(B)$. In the beginning, $D(B)$ is empty. The basic idea is to sequentially consider one model after the other and to decompose each model $G$ such that subgraphs of previously decomposed model graphs are being reused for the decomposition of $G$. For this purpose, the procedure *decompose* given in Fig. 2 is called sequentially for each model graph $G$. Note that the decomposition $D(B)$—or $D$ if $B$ is not explicitly mentioned—is considered a global variable which retains its contents for each call to *decompose*. The task of the procedure *decompose* is to find the largest subgraph $S_{max}$ in the model graph $G$ that is already represented in $D$.[1] If $S_{max}$ is isormorphic to $G$, then $G$ is already represented in $D$ and the algorithm exists. If $G$ consists of a single vertex only, it cannot be decomposed any further and the algorithm exists. Otherwise, $G$ is decomposed into $S_{max}$ and $G - S_{max}$. Clearly, $S_{max}$ has been previously treated by the algorithm and, hence, only $G - S_{max}$ must be further decomposed by calling the algorithm, recursively. If at some point in the recursion, no subgraph of $G$ is already respresented by $D(B)$, we randomly choose a subgraph $S_{max}$ of $G$, for example, one that consists of half the vertices of $G$ for further decomposition. Finally, the tuple $(G, S_{max}, G - S_{max}, E)$ is added to $D$.

Although the algorithm *decomposition* will usually not generate an optimal decomposition, it was shown in practical experiments that this has no significant influence on the performance of the run-time algorithm (see Section 5). Very important, however, is the fact that the algorithm *decomposition* is incremental, i.e., given a set $B$ of model graphs that are represented by the decomposition $D(B)$, a new model graph $G_{N+1}$ can be added to the database by simply calling *decompose*($G_{N+1}$). Thus, $D(B)$ can be updated incrementally without the need for a complete recomputation of $D(B)$ from

1. In order to find the largest subgraph in $G$ that is already represented, it is necessary to apply a subgraph isomorphism algorithm. As the decomposition is an offline process, some conventional algorithm such as Ullman's algorithm may be used. However, for the complexity analysis in Section 4, we will assume that the new algorithm *NA* described in Section 3.3 is applied in the decomposition process.

VERTEX_TEST$(v, l, G_I)$

1. let $G_I = (V_I, E_I, \mu_I, \nu_I)$, $F = \emptyset$, and $l = \mu(v)$.
2. for all $v_I \in V_I$
   (a) if $l = \mu(v_I)$ then set $f(v) = v_I$ and $F = F \cup \{f\}$.
3. return $F$.

Fig. 3. Procedure *vertex_test*.

COMBINE$(S_1, F_1, S_2, F_2, E, G_I)$

1. let $S_1 = (V_1, E_1, \mu_1, \nu_1), S_2 = (V_2, E_2, \mu_2, \nu_2)$ and $F = \emptyset$.
2. for all pairs $f_1, f_2$ where $f_1 \in F_1$ and $f_2 \in F_2$
   (a) test the conditions (1) and (2):
   (1) $f_1(V_1) \cap f_2(V_2) = \emptyset$.
   (2) for each edge $e = (v_1, v_2) \in E$ there exists an edge $e_I = (f_1(v_1), f_2(v_2)) \in E_I$ with $\nu(e) = \nu_I(e_I)$ and for each edge $e_I = (f_1(v_I), f_2(v_I')) \in E_I$ between $f_1(V_1)$ and $f_2(V_2)$ there exists an edge $e = (f_1^{-1}(v_I), f_2^{-1}(v_I')) \in E$ with $\nu_I(e_I) = \nu(e)$.
   (b) if both (1) and (2) are true then let the subgraph isomorphism $f : V_1 \cup V_2 \rightarrow V_I$ from $S_1 \cup_E S_2$ to $G_I$ be defined as follows:

$$f(v) = \begin{cases} f_1(v) & \text{if } v \in V_1 \\ f_2(v) & \text{if } v \in V_2 \end{cases}$$

   Add $f$ to the set $F$, i.e., $F = F \cup \{f\}$.
3. output $F$.

Fig. 4. Procedure *combine*.

scratch. This is particularly of interest in applications where large databases of graphs are involved and new model graphs must be added to the database at run-time.

### 3.3 Subgraph Isomorphism Based on Graph Decomposition

The decomposition of a set of model graphs presented in the Section 3.2 is the basis for an efficient algorithm that detects subgraph isomorphisms from a set of model graphs to an input. Instead of matching each model individually onto a given input, the new algorithm first finds all occurrences of the individual vertices of the model in an input graph. These occurrences are then recursively merged into larger structures until the level of complete model graphs is reached. There are two basic problems that must be solved in this scheme. First, as the smallest component of a graph is a single vertex, there must be a procedure for the detection of subgraph isomorphisms from single vertices to an input graph. Secondly, given a decomposition $D(B)$ and a 4-tuple $(G, G', G'', E) \in D(B)$, if all subgraph isomorphisms from $G'$ and $G''$ to the input graph have been found, they must be combined into subgraph isomorphisms from $G$ to the input graph. For this purpose, a procedure for the combination of subgraph isomorphisms is required.

In Fig. 3, the procedure *vertex_test* is given. It returns all mappings of a single vertex $v$ with label $l$ onto an input graph $G_I$. The procedure simply consists of a loop over all vertices of $G_I$ in which the label of each input graph vertex $v_I$ is compared to the label of the model graph vertex $v$. If the labels are identical then a subgraph isomorphism from $v$ to $G_I$ has been found and can be added to the set of subgraph isomorphisms $F$.

In Fig. 4, the procedure for the combination of subgraph isomorphisms is given. The procedure takes as input two graphs $S_1, S_2$, an input graph $G_I$, a set of edges $E$ with a corresponding edge labeling function, and two sets of functions $F_1, F_2$ which contain all subgraph isomorphisms from $S_1$ and $S_2$ to $G_I$, respectively. Note that each edge $e \in E$ describes an edge between $S_1$ and $S_2$, i.e., $e = (v_1, v_2)$ and $v_1 \in V_1, v_2 \in V_2$, or $v_1 \in V_2, v_2 \in V_1$. In order to combine two functions $f_1 \in F_1$ and $f_2 \in F_2$, there are two conditions that must be satisfied. First, the images of $f_1$ and $f_2$ must be disjoint, i.e., $f_1(V_1) \cap f_2(V_2) = \emptyset$. Otherwise, the combination of $f_1$ and $f_2$ will not be an injective function. Second, it must be ensured that each edge that is specified in the set $E$ is mapped correctly onto edges in $G_I$ and vice versa. Thus, for each edge $e = (v_1, v_2) \in E$ there must be an edge $e_I = (f_1(v_1), f_2(v_2)) \in E_I$ and $\nu(e) = \nu_I(e_I)$. Also, for each edge

NA$(D(B), G_I)$

1. let $D(B) = \{(S_1, S_1', S_1'', E_1), \ldots, (S_n, S_n', S_n'', E_n)\}$, and $P$ be the set of all graphs occurring in $D(B)$, i.e. $P = \cup_{i=1}^{n} \{S_i, S_i', S_i''\}$ .
2. for all $S \in P$ mark $S$ as *unsolved*.
3. for all $S = (V_s, E_s, \mu_s, \nu_s) \in P$ with $|V_s| = 1$
   (a) $F_s = vertex\_test(v, \mu_s(v), G_I)$ where $\{v\} = V_s$.
   (b) if $F_s = \emptyset$ mark $S$ *dead* else mark $S$ *alive* and associate $F_s$ to $S$.
4. while there are $S \in P$ which are marked *unsolved* do
   (a) choose $(S, S_1, S_2, E) \in D(B)$ where $S$ is marked *unsolved* and $S_1, S_2$ are both marked *alive*. If no such entry can be found then goto 5.
   (b) let $F_1$,$F_2$ be the set of subgraph isomorphism associated with $S_1$ and $S_2$, respectively.
   (c) $F_s = combine(S_1, F_1, S_2, F_2, E, G_I)$
   (d) if $F_s = \emptyset$ then mark $S$ *dead* else mark $S$ *alive* and associate $F_s$ to $S$.
5. for each model graph $G_i$ which is marked *alive*, output the subgraph isomorphisms that are associated with $G_i$.

Fig. 5. Algorithm *NA*.

$e_I = (v_I, v_I') \in E_I$ between $f_1(V_1)$ and $f_2(V_2)$ there must be an edge $e = (f_1^{-1}(v_I), f_2^{-1}(v_I')) \in E$ with $\nu_I(e_I) = \nu(e)$. If both conditions are satisfied, the functions $f_1$ and $f_2$ can be combined into a subgraph isomorphism from $S_1 \cup_E S_2$ to the input. When all combinations of functions in $F_1$ and $F_2$ have been tested, the procedure terminates by outputting the set $F_s$ of subgraph isomorphisms from the union graph $S_1 \cup_E S_2$ to $G_I$.

Based on the decomposition of a set of model graphs and the procedures *vertex_test* and *combine*, we can formulate the new subgraph isomorphism algorithm (Fig. 5). The input to the algorithm consists of a decomposition $D(B)$, which represents the model graphs $B = \{G_1, \ldots, G_N\}$ and an input graph $G_I$. Informally speaking, the algorithm must first search for subgraph isomorphisms from the smallest components described in the decomposition $D(B)$ to the input graph $G_I$ and then gradually combine them into larger subgraph isomorphisms. In order to keep track of the components that have been matched already with the input graph, each subgraph $S, S'$ or $S''$ occurring in a 4-tuple $(S, S', S'', E)$ in $D(B)$ can be marked with one of three different tags. In the beginning, all subgraphs in the decomposition are marked *unsolved*. As soon as a subgraph has been tested for subgraph isomorphisms with the input graph, it is either marked *alive* or *dead*. If the search for subgraph isomorphisms was successful, then the subgraph is marked *alive* and all detected subgraph isomorphisms are associated with it. Otherwise, the subgraph is marked *dead* and no subgraph isomorphisms are associated with it. First, in Step 3a and Step 3b, the algorithm loops over all components of the model graphs that consist of only one vertex and calls the procedure *vertex_test* for each of these components. Notice that if a vertex—or, more precisely, a particular vertex label—appears multiple times in the same or different model graphs, the decomposition represents this vertex only once and, thus, *vertex_test* is called exactly once for this type of vertex. Next, in Steps 4a, 4b, 4c, and 4d, each graph $S$ is considered which is marked *unsolved*, but decomposed into two subgraphs $S_1, S_2$ which have been previously tested and successfully matched to the input graph. The subgraph isomorphisms that are associated with $S_1$ and $S_2$ are combined into possible subgraph isomorphisms for $S$ by calling the procedure *combine*. If the set of subgraph isomorphisms returned by *combine* is empty, then $S$ is marked *dead* and, consequently, for each model graph which contains $S$ as a subgraph, there exists no subgraph isomorphism to the input graph. On the other hand, if the returned set $F$ of subgraph isomorphisms is not empty then $S$ is marked *alive*. This process continues until either all 4-tuples in $D(B)$ have been tested or no 4-tuple can be found in Step 4a for which both subgraphs are marked *alive*. In this case, the search for further subgraph isomorphisms can be terminated immediately. Finally, in Step 5, all subgraph isomorphism that have been found for the model graphs represented by $D(B)$ are output.

It is easy to see that the new algorithm finds all subgraph isomorphisms from each of the model graphs $G_1, \ldots, G_N$ to the input graph $G_I$. Furthermore, if a subgraph $S$ is part of several model graphs and represented by the decomposition, i.e., there is a 4-tuple $(S, S_1, S_2, E)$, then the computation of all subgraph isomorphisms from $S$ to the input graph $G_I$ is done only once.

### 3.4 An Example

In order to illustrate the new algorithm, we give a detailed example. In Fig. 6, two model graphs, $g_1$ and $g_2$, and an input graph, $g_3$, are shown. In Fig. 7, the decomposition of $g_1$ and $g_2$ is graphically represented by a network. The

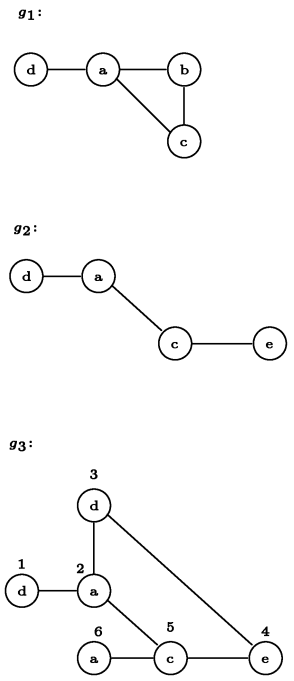



Fig. 6. Two model graphs $g_1, g_2$ and an input graph $g_3$.

network is displayed with vertices of the model graphs in the top layer. Below are subgraphs of the models that consist of more than one vertex. The complete models are represented at the bottom of the network. The numbers to the right of each network node are identifiers. At the top of the network, the five different vertex labels occurring in $g_1$ and $g_2$ are represented. Notice that, for example, the vertex with label $d$ represented in node 2 appears in both $g_1$ and $g_2$. It appears, however, only once in the decomposition. On the second level in Fig. 7, a subgraph that is decomposed into two vertices with label $d$ and $a$, respectively, and an edge connecting them is represented. This subgraph occurs in both $g_1$ and $g_2$. In network node 7, the largest common subgraph of $g_1$ and $g_2$ is represented. It is decomposed into the subgraph represented in node 6, a vertex represented in node 4, and an edge. Finally, network nodes 8 and 9 represent models $g_1$ and $g_2$, respectively.

At run-time, given the input graph $g_3$, the algorithm *NA* described in Section 3.3 first tries to find all subgraph isomorphisms for the vertices represented in the top nodes of the network. For example, the vertex labeled $d$ can be matched onto the vertices 1 and 3 of $g_3$. Consequently, there are two subgraph isomorphisms associated with the node 2 (denoted by $\{1\}$ and $\{3\}$ below the node 2 in Fig. 7). Next, for each subgraph in the decomposition whose components have already been successfully tested, the algorithm tries to combine the subgraph isomorphisms of its components. In node 6, the subgraph isomorphisms $\{1\}$ and $\{3\}$ associated to node 2 and $\{2\}$ and $\{6\}$ associated to node 3 are combined. However, only the combinations $\{12\}$ and $\{32\}$ are valid subgraph isomorphisms from the graph in network node 6 to the input graph $g_3$. Because of the existence of these two subgraph isomorphisms, network node 6 is marked alive and, consequently, the subgraph in node 7 will be tested next. This process continues until a single subgraph isomorphism, $\{1, 254\}$, has been found from the graph $g_2$ to the input graph $g_3$. It can be easily verified that no other subgraph isomorphism from either $g_1$ or $g_2$ to $g_3$ exists.

## 4 Computational Complexity Analysis

In this section, we give a detailed analysis of the run-time complexity of the new algorithm for subgraph isomorphism detection and of the algorithm for the decomposition of the model graphs. The analysis will be based on the following quantities:

$N$ = the number of model graphs in the database,
$M_1$ = the number of vertices of a subgraph that is common to all models,
$M_2$ = the number of vertices that are unique to each model,
$M$ = the total number of vertices of a model graph, i.e., $M = M_1 + M_2$,
$I$ = the number of vertices in the input graph.

Due to the fact that the decomposition algorithm makes extensive use of some subgraph isomorphism algorithm (Step 3 in Fig. 2), its performance is strongly dependent on the performance of the applied algorithm. For this reason, we first study the computational complexity of the subgraph isomorphism algorithms and then examine complexity of the proposed decomposition algorithm.

In our analysis, we only consider the idealized situation where the model graphs have a single subgraph of size $M_1$ in common. In practice, the situation will usually be more complicated due to the existence of subgraphs that are common to some, but not all models. Furthermore, for reasons of convenience, we assume that the model graphs are decomposed as follows: Given a model graph $G$ and $S$, the subgraph of $G$ which is common to all the graphs, we assume that the decomposition contains an entry $(G, S, G - S, E)$. That is, the graph $G$ is decomposed into the common subgraph $S$ and the difference graph $G - S$. The graph $S$ is then decomposed into a subgraph $S_i$ that consists of a single vertex and the difference graph $S - S_i$, which is again decomposed into a single vertex and the remaining difference graph. This process is continued until the difference graph itself consists of a single vertex only. The same decomposition scheme is applied to the graph $G - S$. Consequently, for a common subgraph of $M_1$ vertices, there are $M_1 - 1$ 4-tuples $(S_1, S_1', S_1'', E)$ where $S_1, S_1'$ and $S_1''$ are subgraphs of $S$. $S_1$ consists of $k$, $S_1'$ of $k - 1$ and $S_1''$ of one vertex, $k = 2, \ldots, M_1$. Analogously, there are $M_2 - 1$ 4-tuples in the decomposition of the difference graph $G - S$, which consists of $M_2$ vertices.[2] Based on this, decomposition of the model graphs it is possible to analyze the computational complexity of the new algorithm *NA* given in Fig. 5. The computation steps that are performed can be estimated in terms of the number of calls to the procedure *vertex_test* times the complexity of *vertex_test*,

2. It must be mentioned that the algorithm *decomposition* given in Section 3.2 usually does not decompose the model graphs the way it is assumed here. However, the results of the complexity analysis were confirmed in a number of practical experiments documented in Section 5.

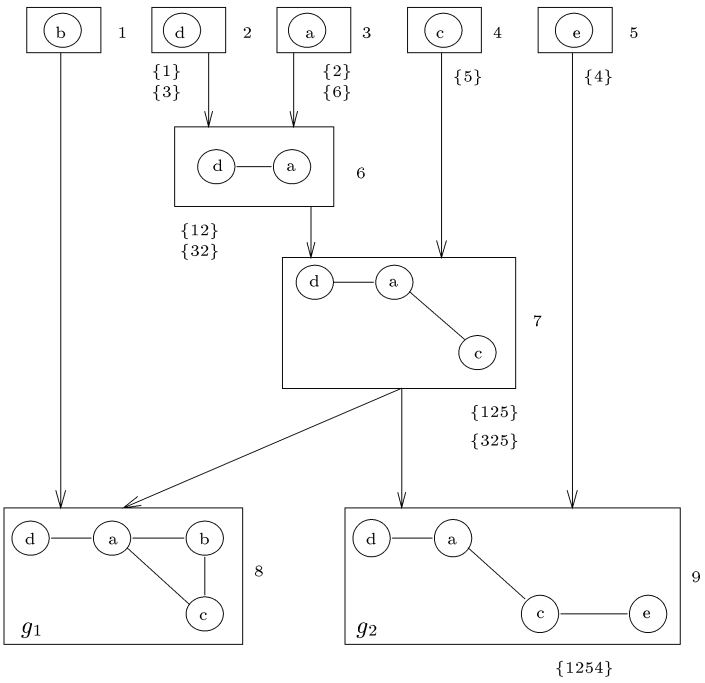



Fig. 7. Decomposition of graphs $g_1, g_2$ from Fig. 6 displayed as a network structure.

and the number of calls to the procedure *combine* times the complexity of *combine* itself. In Table 1, an overview of the theoretical boundaries for both the new algorithm and, for comparison reasons, Ullman's algorithm [21] is given. (A complexity analysis of Ullman's algorithm is given in Appendix A.)

The best case for both algorithms arises when each of the vertices of the model graphs is uniquely labeled. For a single model graph, there will be $M$ different vertices and consequently, the procedure *vertex_test* will be called $M$ times. Each call will test $I$ vertices of the input graph and, thus, cause a total number of $O(IM)$ steps. The number of matchings found by *vertex_test* of a single model graph vertex onto the input graph will be bounded by $O(I/M)$. There will $O(M)$ calls to the procedure *combine*, i.e., for each 4-tuple $(S_1, S_1', S_1'', E)$ there will be a call $combine(S_1', F_1', S_1'', F_1'', E, G_I)$, where $S_1'$ consists of $k-1$ and $S_1''$ of one vertex. Because all vertices are uniquely labeled, $F_1'$ contains at most one subgraph isomorphism and there will be $O(I/M)$ subgraph isomorphisms in $F_1''$. In the procedure *combine*, each mapping in $F_1'$ is combined with each mapping in $F_1''$. Comparing the images of the mappings takes $O(k)$ steps and testing each edge between $S_1'$ and $S_1''$ takes $O(k)$ steps with $k \leq M$. Thus, the total computational complexity for matching a single model graph against an input graph is in the best case bounded by:

$$O(IM + M^2 I/M) = O(IM). \tag{1}$$

When $N$ model graphs must be matched onto the input graph, there will be $O(IM_1)$ steps performed for the common subgraph $S$ and $O(NIM_2)$ steps for each difference graph $G_i - S$ of the $N$ model graphs. Additionally, for each model graph the subgraph isomorphisms found for the common subgraph and the difference graph must be combined, requiring $O(NM_1M_2)$ steps at most. Thus, the computational complexity of the new algorithm for finding all subgraph isomorphisms from $N$ model graphs to an input graph is in the best case bounded by:

$$O(IM_1 + NIM_2 + NM_1M_2). \tag{2}$$

In the extreme case where there is no common subgraph among the model graphs, i.e., $M_1 = 0$ and $M_2 = M$, the complexity is bounded by:

$$O(NIM). \tag{3}$$

On the other hand, in the limit when all models are identical, i.e., $M_1 = M$ and $M_2 = 0$, the complexity of the new algorithm becomes independent of the number of models and is only bounded by:

$$O(IM). \tag{4}$$

The worst case for both the new and the traditional algorithm arises when all vertices in the model graphs have the same label and each vertex is connected to each other vertex. Given a single model graph, the new algorithm makes only one call to the procedure *vertex_test* as there is only one vertex label. Each of the model vertices is then successfully matched onto $O(I)$ input graph vertices and, consequently, there are $O(I)$ subgraph isomorphism returned by *vertex_test*. These subgraph isomorphisms constitute the input to the first call of the procedure *combine* in which $O(I^2)$ subgraph isomorphisms are found at the expense of $O(M)$ steps. Next, according to the decomposition described at the beginning of this section, there will be $M$ calls to *combine*. For each call, the number of subgraph isomorphisms that must be combined will be increased by a factor of $I$. Consequently, the total number of steps for a single model graph amounts in the worst case to

$$O(I + MI^M M) = O(I^M M^2). \tag{5}$$

TABLE 1
Best and Worst Case Complexities of the New and the Traditional Algorithm

| | NEW ALGORITHM | ULLMAN'S ALGORITHM |
|---|---|---|
| All models are completely different: $M_1 = 0$, $M_2 = M$ | | |
| best case | $O(NIM)$ | $O(NIM)$ |
| worst case | $O(NI^M M^2)$ | $O(NI^M M^2)$ |
| All models are identical: $M_1 = M$, $M_2 = 0$ | | |
| best case | $O(IM)$ | $O(NIM)$ |
| worst case | $O(I^M M^2)$ | $O(NI^M M^2)$ |

For a database of $N$ model graphs, the total effort is the sum of the effort for testing the common subgraph and the effort for testing the subgraphs that are unique to each model graph. The common subgraph is tested exactly once at the expense of $O(I^{M_1} M_1^2)$, while the unique subgraph must be tested for each model graph individually at the expense of $O(I^{M_2} M_2^2)$. By adding $O(NM_1M_2I^M)$ steps necessary for the combination of the subgraph isomorphisms, we arrive at a total complexity for $N$ model graphs of

$$O(I_1^M M_1^2 + NI_2^M M_2^2 + NM_1M_2I^M). \tag{6}$$

In the extreme case, when the database consists of completely different model graphs only, i.e., $M_1 = 0, M_2 = M$, the above formula becomes

$$O(NI^M M^2). \tag{7}$$

On the other hand, if all models are identical, i.e., $M_1 = M, M_2 = 0$, the size of the database, $N$, disappears in (6) and the worst case complexity of the new algorithm becomes

$$O(I^M M^2). \tag{8}$$

When compared to the computational complexity of Ullman's algorithm given in the second column of Table 1, we note that both in the best and worst case the new algorithm is faster than the traditional algorithm by a factor of $N$ for similar model graphs in the limit. On the other hand, given a set of completely disjoint model graphs both algorithms have the same computational complexity. In practice, we can expect the computational complexity somewhere between $O(IM)$ and $O(NI^M M^2)$.

While the run-time complexity of the new subgraph isomorphism algorithm is of general importance, the complexity of the decomposition algorithm is only relevant to applications which cannot neglect the time spent for preprocessing. For example, in applications where some models are known beforehand, but others are only acquired at run-time, the time spent for decomposing and adding the new models to the existing decomposition may be restricted. Thus, the performance of the decomposition algorithm will become important.

Given a set of models $B = \{G_1, \ldots, G_N\}$ and a decomposition $D(B)$ of $B$, a new model $G$ is decomposed and added to $D(B)$ by calling the procedure *decompose(G)*. The basic idea of this procedure is to find a tuple $(G_i, G_i', G_i'', E)$ in $D(B)$ such that $G_i$ is a subgraph of $G$ and maximal in $D(B)$. In case of success, the model $G$ is decomposed into $G_i$ and the difference graph $G - G_i$ which will be decomposed itself by recursively calling the procedure *decompose*. Clearly, the computational complexity of the decomposition process depends on the number of calls to *decompose* and the complexity of the search for the largest subgraph in *decompose*. It is easy to see that for a graph $G$ with $M$ vertices, any decomposition of $G$ consists of at most $O(2M) = O(M)$ subgraphs. As each of these subgraphs must be decomposed itself (unless it is already represented in $D(B)$) the number of calls to *decompose* is at most $O(M)$. In each call to *decompose*, a search for the largest subgraph of $G$ that is already represented in $D(B)$ is performed. This search process requires that for each tuple $(G_i, G_i', G_i'', E)$ in $D(B)$ a subgraph isomorphism from $G_i$ to $G$ is computed. Apparently, if these subgraph isomorphisms are computed with a conventional algorithm such as Ullman's algorithm, then the complexity of the search process will be linearly dependent on the number of tuples in $D(B)$. However, due to the fact that the algorithm *NA* was especially designed to solve the subgraph isomorphism problem based on a decomposition of the model graphs, it can also be applied in the decomposition algorithm itself. That is, given a decomposition $D(B)$ and a model graph $G$, the decomposition algorithm calls the algorithm *NA* in order to find all subgraph isomorphisms from graphs in $D(B)$ to $G$. Consequently, the computational complexity of the search process in *decompose* depends on the computational complexity of the new algorithm *NA* (see Table 1). Note that the number of vertices in the input graph is denoted by $I$ in Table 1. However, when *NA* is applied in the decomposition algorithm, the input graph is in fact a model graph and, therefore, $I = M$. This means that in the best case, when all the models are identical and their vertices are uniquely labeled, the number of steps performed in *decompose* is bounded by $O(M^2)$ (see (4)). As there are $O(M)$ recursive calls to *decompose* the total number of steps for the

TABLE 2
Parameters of the Various Experiments

| Experiments | Parameters for the graph generation | | | | |
|---|---|---|---|---|---|
| Nr. | Fig. | Vertices | Edges | Labels | Common | Database size |
| 1 | 8,9,10,11 | 10 - 50 | 12 - 60 | 10 | - | 1 - 20 |
| 2 | 12,13,14 | 50 | 60 | 10 | 5 - 45 | 1 - 20 |
| 3 | 15,16 | 50 | 60 | 4 - 40 | - | 1 - 20 |
| 4 | 17,18,19 | 50 | 50 - 120 | 10 | - | 1 - 10 |
| 5 | 20,21,22 | 50-300 | 60 - 360 | 10 | - | 1 - 20 |
| 6 | 23 | 50-500 | 60-600 | 30 | - | 1 |
| 7 | 24 | 50 | 60 | 10 | - | 1-100 |
| 8 | 25 | 50 | 60 | 10 | 30 | 20 |

decomposition of a model graph with $M$ vertices is in the best case bounded by $O(M^3)$. Notice that in order to decompose a complete set of model graphs from scratch, the procedure *decompose* is sequentially called for each of these models. Thus, in the best case, the computational complexity for decomposing a set of $N$ models is bounded by:

$$O(NM^3). \tag{9}$$

From the previous complexity analysis of the algorithm *NA*, we know that the worst case arises when all the graphs are completely different and the vertices of each graph are identically labeled. Thus, according to the worst case complexity of algorithm *NA* given in (7), the number of steps in *decompose* is bounded by $O(NI^M M^2) = O(NM^{M+2})$ for $I = M$. As there are $O(M)$ subgraphs in a decomposition of a model with $M$ vertices, the computational complexity of the decomposition algorithm is in the worst case bounded by $O(NM^{M+3})$. Again, if there is a set of models that must be decomposed from scratch, the procedure *decompose* will be called for each of these models individually and the worst case complexity will be bounded by:

$$O(N^2 M^{M+3}). \tag{10}$$

Notice that in spite of the exponential worst case complexity of the proposed decomposition algorithm, it is our experience that it performs reasonably well for most applications (see Section 5).

## 5 Experimental Results

In order to examine the performance of the new algorithm in practice, we have performed a number of experiments with randomly generated graphs. The new algorithm (NA) and Ullman's refinement procedure (UA) were both implemented in C++ and run on a SUN Sparc10 Workstation. The code of NA consists of roughly 7,000 lines compared to 2,500 lines of UA.

All model and input graphs were randomly generated. The parameters in the random graph generation process were:

- the number of vertices,
- the number of edges,
- the number of different vertex labels (all edges were unlabeled and undirected),
- the number of graphs in the database,
- the size of the common subgraph that was contained in all models.

In our experiments, we automatically generated model graphs based on these parameters. From each model graph, a corresponding isomorphic input graph was derived by copying and permuting its vertices and edges. Both NA and UA were then used to detect all graph isomorphisms from the models to the inputs. In order to account for the random nature of the underlying graphs, each experimental run was repeated ten times and the average computation time was recorded as a result. By varying different parameters in the graph generation process, the behavior of the new algorithm was studied. In Table 2, an overview of the experiments is given, indicating the different parameter values used in each specific experiment.

In the first experiment, we were interested in measuring the influence of the size of the graphs and the database on the performance of both algorithms. We started with a database consisting of a single graph with 10 vertices and 12 edges. The number of vertex labels was set to 10. We gradually increased the number of graphs in the database to 20 and we also increased the number of vertices from 10 to 50 together with the number of edges from 12 to 60. In Fig. 8, the results for NA (lower plane) and UA (upper plane) are shown. Apparently, the performance of UA became worse than that of NA with an increasing graph as well as an

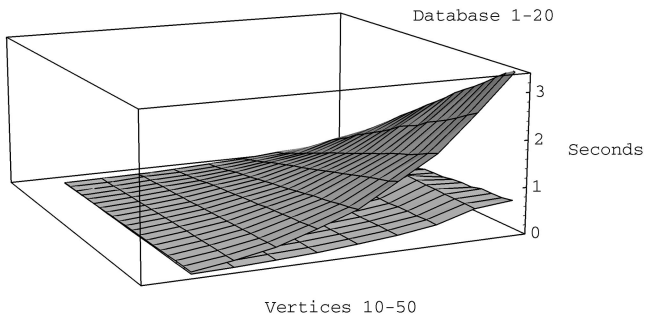



Fig. 8. Time in seconds for increasing the number of vertices and the size of the database. (The lower plane denotes the time of NA, while the upper plane represents the time of UA.)

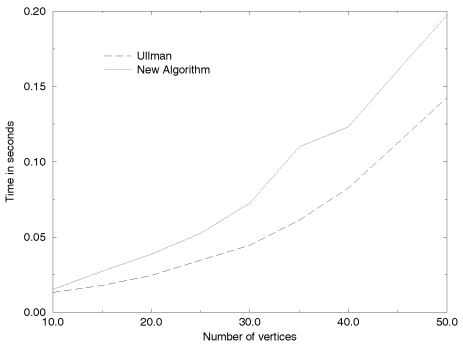



Fig. 9. Cut through Fig. 8 along the vertex axis for one model graph in the database.

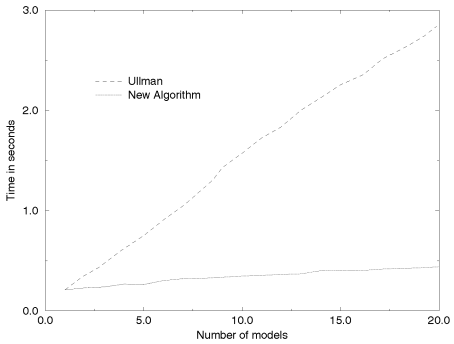



Fig. 11. Cut through Fig. 8 along the database axis for 50 vertices in the model graphs.

increasing database size. In Fig. 9, a cut through the 3D-plot along the axis for one model in the database is shown. We can observe that for a single model graph, UA is slightly faster than NA. However, if the database contains more than one model graph, the ability to compactly represent common substructures will make NA eventually faster than UA. This is confirmed in Fig.10, where the number of graphs in the database was fixed at 20. The observation that UA's performance decreases faster than that of NA for a growing number of vertices is explained by the fact that by increasing the size of the model graphs while leaving the number of labels constant, there will be a growing number of substructures that occur multiple times within the same or different models. These substructures need to be considered only once by NA. UA, however, must search for these substructures each time they appear in a model graph.

In Fig. 11, a cut through Fig. 8 with the number of vertices fixed at 50, shows the influence of the database size more closely. As was expected from the complexity analysis, UA was linearly dependent on the size of the database, while NA's behavior was almost independent of the number of models.

In this first experiment, there were between one and five vertices with identical labels in each graph on the average. Therefore, many of the generated model graphs had some random substructures in common. In the second experiment, the effect of sharing common substructures among models was examined more closely. For this purpose, we generated model graphs consisting of 50 vertices and approximately 60 edges and increased the number of models from one to 20 and the size of the common subgraph from five to 45 vertices. Except for the common subgraph, all models were disjoint, i.e., except for the vertices of the common subgraph for which 15 different labels were allocated, the vertices of the model graphs were all uniquely labeled. The results of the second experiment are given in Fig. 12, where the lower plane at the front corner denotes the times of NA while the upper plane represents UA. Clearly, the intersection of the two planes indicates that for a small or no common subgraph in the different models the performance of NA was slightly worse than that of UA. But for an increasing size of the common subgraph, the time needed by NA decreased and became much less than that required by UA.

In Fig. 13, a cut through Fig. 12 along the common subgraph axis is displayed. The size of the database was set to 20. We observe that if the common subgraph contained less than 15 vertices, NA performed worse than UA. However, for larger common subgraphs the time required by NA decreased rapidly while UA's time requirement increased. Note that in both Fig. 12 and Fig. 13, the database contained at most 20 models. It may be expected that even for small subgraphs there is a size of the database for which NA outperforms UA. Fig. 14 demonstrates this behavior. It

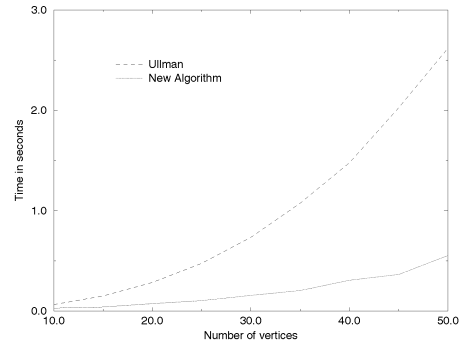



Fig. 10. Cut through Fig. 8 along the vertex axis for 20 model graphs in the database.

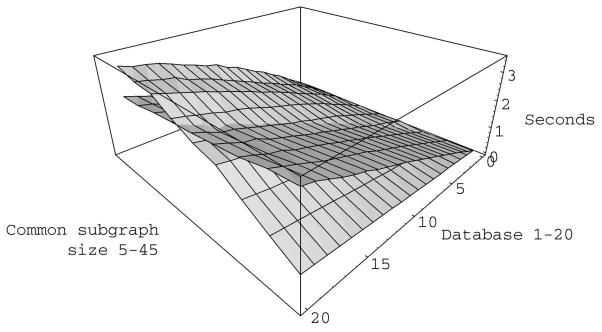



Fig. 12. Time in seconds for an increasing number of vertices in the common subgraph and an increasing size of the database. (The lower plane in the front corner denotes the time of NA, while the upper plane represents the time of UA.)

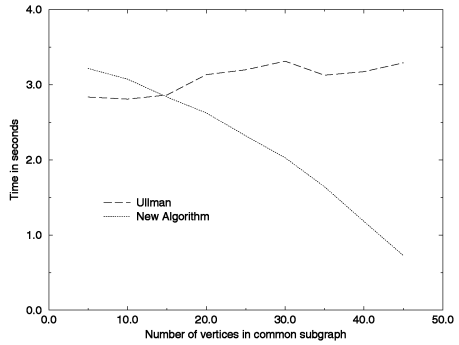



Fig. 13. Cut through Fig. 12 along the common subgraph axis with the size of the database set to 20.

shows a cut through Fig. 12 along the database axis with the common subgraph size set to five vertices. As an extension of the results shown in Fig. 12, the size of the database was varied between 1 and 40. For a database size of 30 models, the run-time of the two algorithms was identical and for larger databases NA performed better than UA. This indicates that NA is more efficient than UA even for small common subgraphs provided that the number of models in the database is sufficiently large.

In the first and second experiment, we always kept the number of vertex labels constant. In the third experiment, this number was varied. All generated graphs contained 50 vertices and 60 edges. The number of vertex labels grew from 1 to 40 and again, we increased the size of the database from 1 to 20. The results are displayed in Fig. 15. The lower (upper) plane in the right corner corresponds to NA (UA). It can be observed that the two planes intersect each other. That is, the performance of NA became worse than that of UA when less than five labels were present. The line of intersection of the two planes grows with the number of graphs in the database as expected. In Fig. 16, a cut through Fig. 15 along the axes for the labels is shown, with the database size set to 10. Between 10 and 40 labels, the performance of NA and UA changed only minimally and NA was always faster. However, for less than five labels, NA took much more time to finish than UA. The worst case emerged for NA when there was only one vertex label, i.e., all vertices were identically labeled. In this case, NA required an average of 12 minutes in order to detect all graph matches. This decrease in the performance of NA is

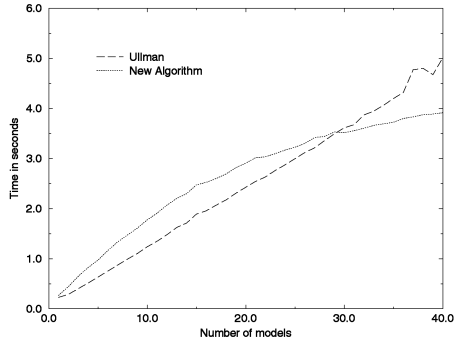



Fig. 14. Cut through Fig. 12 along the database axis with the size of the common subgraph set to 5.

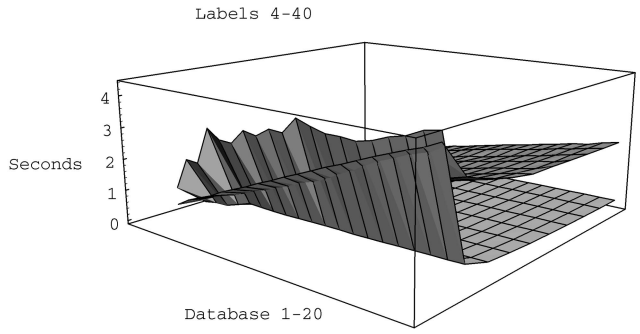



Fig. 15. Time in seconds for increasing number of labels and size of the database. (The lower plane in the right corner denotes time of NA while the upper plane represents UA.)

due to the fact that for unlabeled graphs, the number of matches that are found for small subgraphs of the model graphs is usually very large. Thus, the concept of the new algorithm to first test the smaller components of the model graphs and then to combine them into subgraph isomorphisms for larger components is a major disadvantage when the underlying graphs are unlabeled. Consequently, the worst case behavior, as described in the complexity analysis section can be observed in Fig. 15 and Fig. 16. Obviously, NA tends towards the worst case behavior faster than UA for a decreasing number of labels.

In the fourth experiment, the influence of the connectivity, i.e., the number of edges in a graph, was examined. In all of the previous experiments, the average degree of each vertex was kept at 2.5. For the experiment documented in Fig. 17, the graphs consisted of 50 vertices, 10 vertex labels, and a gradually increasing number of edges, starting at 50 and ending at 120. Similar to the previous experiments, we also varied the number of models in the database from 1 to 10. In Fig. 17, the lower plane (upper plane) in the left corner denotes the times of NA (UA). Apparently, the performance of NA decreased with a growing number of edges in the graphs. In Fig. 18, a cut through Fig. 17 is taken along the axis denoting the number of edges with the number of graphs in the database fixed at 10. There was practically no influence of the number of edges on the performance of UA in the considered range. However, the computation time of NA increased fast with the number of edges. This behavior is due to the fact that with a high

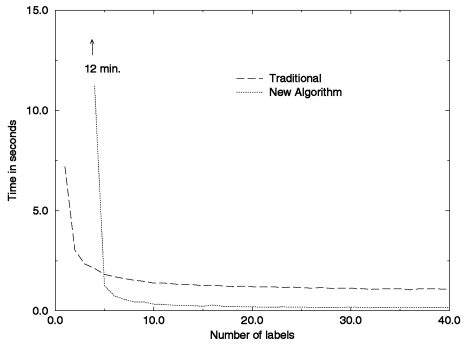



Fig. 16. Cut through Fig. 15 along the labels axis with 10 model graphs in the database.

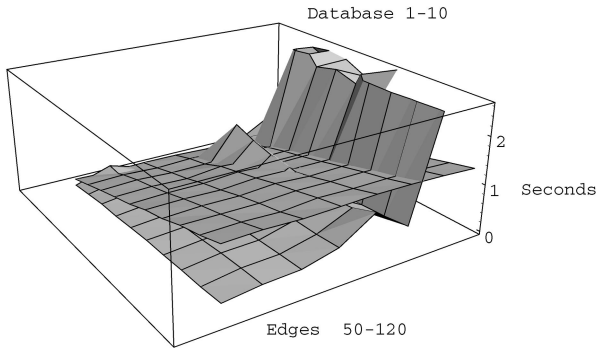



Fig. 17. Time in seconds for increasing number of edges and size of the database. (The lower plane in the left corner denotes the time of NA while the upper plane represents UA.)

connectivity, the number of small subgraphs of a graph that can be matched onto themselves is large. On the other hand, the tendency of UA to spend more time than NA for larger databases was again confirmed in Fig. 19, where a cut of Fig. 17 along the axis denoting the size of the database is shown. The number of edges was constantly set to 100. We observe that for less than seven models in the database, NA was slower than UA. For more than seven models, the sharing of common subgraphs in NA and the linear dependency of UA on the size of the database resulted in UA taking more time than NA.

While in the first four experiments, we examined the behavior of NA for the case of graph isomorphism, the fifth experiment was especially devoted to subgraph isomorphism detection. The models generated for this experiment consisted of 50 vertices, 10 different vertex labels, and 60 edges. For each model graph, a corresponding input graph was created by copying the model and subsequently adding a growing number of vertices and edges. Thus, it was ensured that there existed at least one subgraph isomorphism from the model to the input. The number of additional vertices in the input graph was varied between 0 and 250, i.e., in the beginning, the input graphs consisted of 50 and in the end of 300 vertices. At the same time, the number of models in the database was also increased from 1 to 20. The results of the

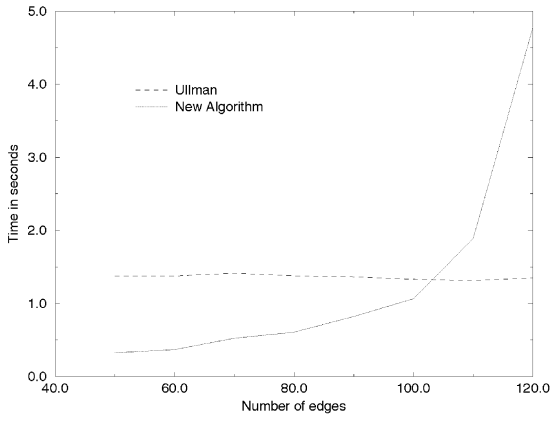



Fig. 18. Cut through Fig. 17 along the edges axis with 10 model graphs in the database.

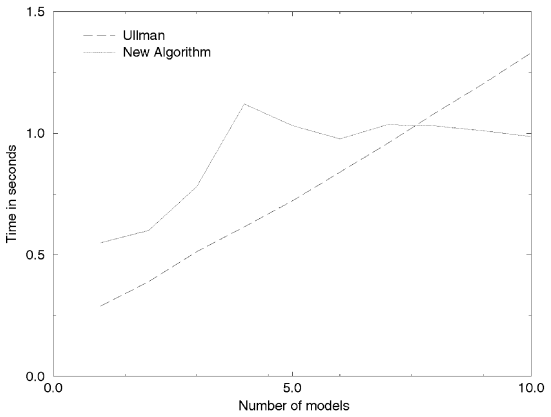



Fig. 19. Cut through Fig. 17 protect along the database axis with 100 edges in the model graphs.

fifth experiment are given in Fig. 20. Note that the lower plane in the left corner denotes the time of NA while the upper plane represents UA. As expected, both NA and UA used more time for a growing input graph and a growing database. However, while UA performed better than NA when there were only few models in the database, the linear dependency on the database size caused UA to take more time than NA when there were more than ten models. In Fig. 21, a cut through Fig. 20 along the axis of the input graph for a database containing one model is given. It confirms that NA did not perform as well as UA for finding all subgraph isomorphisms from a model to a growing input graph. On the other hand, in Fig. 22, a cut through Fig. 20 along the database axis with the input graph constantly set to 300 vertices reveals that due to the compact representation of the models, NA used less time than UA when the database size was increased.

In all the experiments documented so far, the performance of UA and NA were studied for model graphs and databases of moderate sizes. The next two experiments were devoted to testing the behavior of both algorithms for very large graphs on the one hand and very large databases on the other hand. In Fig. 23, the size of the model and input graph was increased from 50 to 500 vertices along with the number of edges that was increased from 60 to 600 edges. There were 30 different vertex labels used, i.e., for a graph with 300 vertices there were on the average 10 vertices with the same label. The database consisted of a single model graph. We can observe that the performance of UA decreased much faster than NA's performance. While UA matched graphs with 250 vertices within 30 seconds, NA was capable of matching graphs twice as large in the same time. This behavior can again be explained by the fact that with an increasing number of vertices (and a constant number of labels) the number of substructures that appear multiple times inside the same model graph grows. Although the algorithm *decomposition* that was proposed for the offline decomposition of the models usually does not detect all of these substructures and represents them only once, the resulting decomposition is still fairly compact and allows NA to perform better than UA. Most important, however, is that *decomposition* is computationally inexpensive. For example, the decomposition of a graph with 50

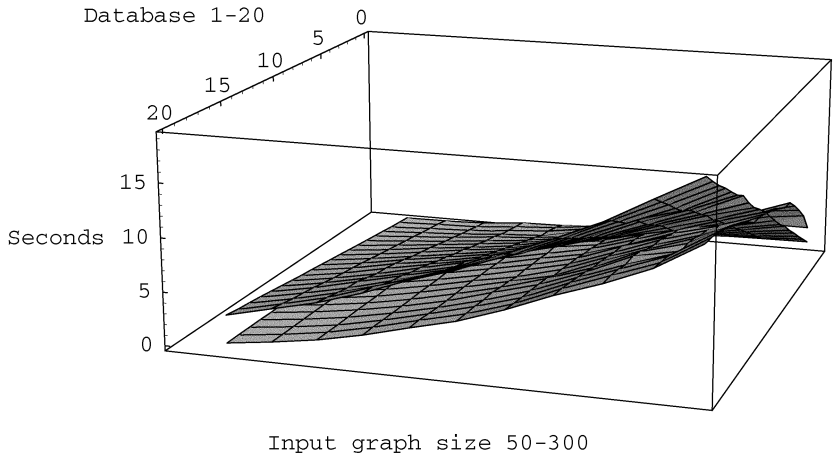



Fig. 20. Time in seconds for increasing number of vertices in the input and size of the database. (The lower plane in the left corner denotes the time of NA while the upper plane represents UA.)

vertices and 60 edges took only 0.21 seconds, and for a graph with 500 vertices and 600 edges only 70 seconds were required (not documented here).

Next, in Fig. 24, the number of model graphs in the database was increased from 1 to 100. As in the very first experiment (Fig. 8) the graphs consisted of 50 vertices and 60 edges with 10 vertex labels. Note that with NA even large databases remain tractable (about 1 second) while UA requires time that is linearly dependent on the database (15 seconds for 100 models).

Finally, in the eighth experiment, we studied the influence of the decomposition algorithm on the performance of NA. In Section 3.2, it was observed that there exists a large number of different decompositions of a set of model graphs. These decompositions differ in terms of compactness and also in terms of the run-time performance of NA. For example, it can be hypothesized that the optimal decomposition, i.e., the decomposition which contains a 4-tuple $(G_c, G_c', G_c'', E_c)$ for the largest subgraph $G_c$ that appears in all the model graphs, will guarantee the optimal run-time performance of NA. However, as mentioned before, the computation of the optimal decomposition is a highly exponential task. Thus, in Section 3.2, we proposed the algorithm *decomposition* which is computationally inexpensive. The drawback of this algorithm, however, is that it does not necessarily generate an optimal decomposition for a given set of models. Furthermore, as the models are treated sequentially, the resulting decomposition is dependent on the order of the models in the database. In order to study the difference in the performance of NA for an optimally decomposed database of models, on the one hand, and for a nonoptimal decomposition of the same database on the other hand, the following experiment was performed. We generated a database containing 20 model graphs, each consisting of 50 vertices, 60 edges, and 10 different vertex labels. Furthermore, a subgraph $G_c$ with 30 vertices and 40 edges that appeared in all the model graphs was explicitly defined. Thus, it was possible to simulate the result of an optimal decomposition algorithm by taking the largest common subgraph $G_c$ directly from the graph generation process. The input graphs were isomorphic copies of the model graphs. The performance of NA for detecting all graph isomorphisms from the models to the inputs based on this optimal decomposition was compared to its performance when the proposed, nonoptimal algorithm *decomposition* was applied. In Fig. 25, the

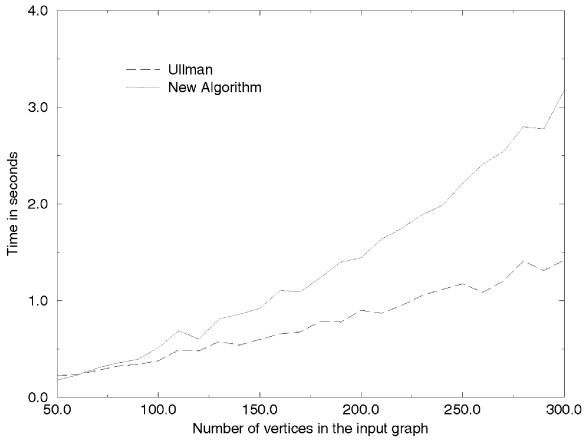



Fig. 21. Cut through Fig. 20 along the input graph axis with 1 model in the database.

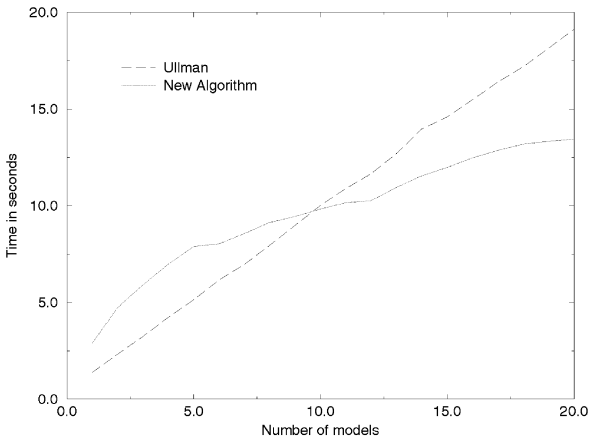



Fig. 22. Cut through Fig. 20 along the database axis with 300 vertices in the input graph.

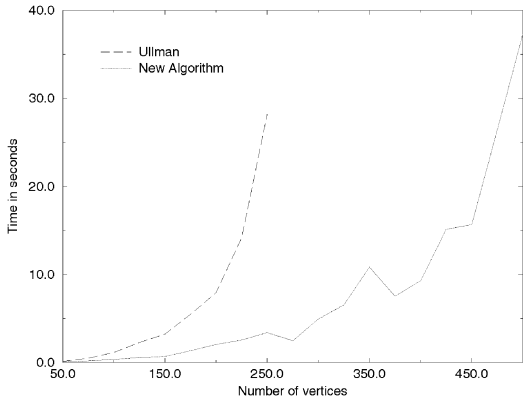



Fig. 23. Computation time for an increasing number of vertices in the model graphs.

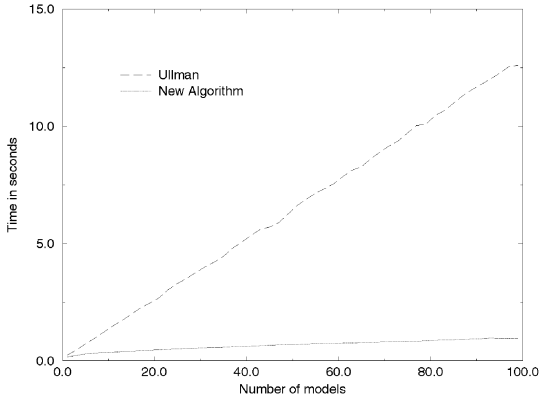



Fig. 24. Computation time for an increasing number of models in the database.

computation time of NA for both types of decompositions is given for various permutations of the models in the database. While the optimal decomposition was independent of the order of models, the proposed decomposition algorithm generated different decompositions for different orderings of the models and the performance of NA varied accordingly. With the optimal decomposition, NA performed faster than with the decompositions produced by algorithm *decomposition*. However, the average (maximum) difference in the performance is 14 percent (31 percent) compared to the optimal decomposition. Considering the fact that UA required more than 2 seconds in this experiment (not documented here), i.e., 870 percent more than NA based on the optimal decomposition, we can conclude that NA outperforms UA independent of the decomposition strategy that is used. Hence, it can be argued that the advantages of the proposed decomposition algorithm (incremental update, computationally inexpensive) outweigh the disadvantages (nonoptimal decomposition).

## 6 Summary and Conclusion

In this paper, a new algorithm for efficient subgraph isomorphism detection was proposed. The new algorithm is based on the idea of finding common subgraphs among different model graphs or within a single model graph. In an offline preprocessing step, these common subgraphs are recorded and at runtime they are used to efficiently detect subgraph isomorphisms from all model graphs to some input graph. In a theoretical complexity analysis it was shown that in the limit, when all model graphs are highly similar, the new method becomes independent of the size of the database. In order to verify the results of the theoretical complexity analysis, we performed a number of practical experiments with randomly generated graphs. The experiments confirmed that for model graphs with some common substructures the new algorithm is only sublinearly dependent on the size of the database. Furthermore, it was observed that for large model graphs the new algorithm is faster than traditional algorithms due to reappearing substructures that naturally evolve in large graphs. On the other hand, the experiments also documented that for unlabeled, highly connected graphs, the new algorithm performs poorly.

We conclude that the new algorithm presented in this paper is highly recommended for applications where we are faced with large databases of large labeled graphs with restricted connectivity. By contrast, the new algorithm

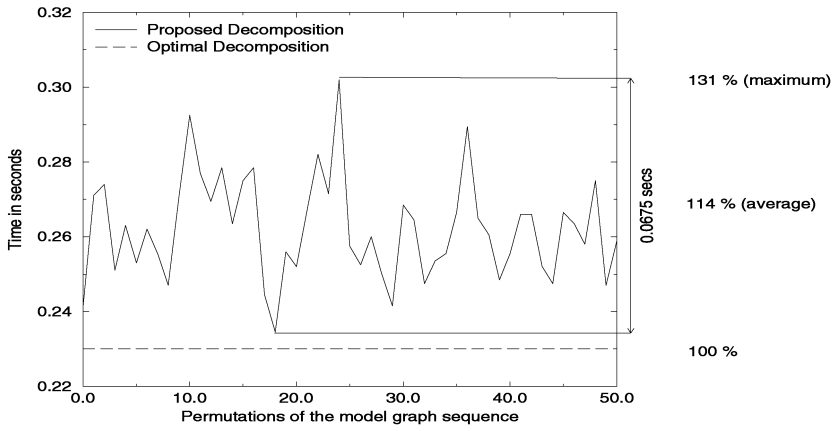



Fig. 25. Computation time for various decompositions of the set of model graphs.

should not be applied to problems dealing with unlabeled, highly connected graphs, to databases of completely disjoint graphs, or to problems where the input graph is considerably larger than the model graphs. We feel that the new algorithm is a substantial contribution to the problem of efficient subgraph matching. The algorithm has been presented in a general form without the explicit use of domain features or heuristics. Therefore, it can be regarded a generic tool that is applicable to various tasks. The extension of the new algorithm to error-tolerant subgraph isomorphism detection is currently under investigation. At last, it is to be mentioned that there is a potential for parallelization inherent in the new approach, which may be used for a further improvement of the performance [42].

## APPENDIX A
## COMPLEXITY OF ULLMAN'S ALGORITHM

In order to understand the complexity analysis of Ullman's algorithm [21] given in this appendix, we first briefly describe the algorithm itself. The basic idea of Ullman's algorithm is to take one vertex of a model after the other and map it onto the input vertices such that the resulting mapping represents a subgraph isomorphism according to Definition 6. If, at some point in the algorithm, the mapping does not represent a subgraph isomorphism then the algorithm backtracks and tries another mapping. Formally, given a model $G = (V, E, \mu, \nu)$ and an input $G_I = (V_I, E_I, \mu_I, \nu_I)$, the first vertex $v_1$ in $V$ is mapped to some vertex $w_i$ in $V_I$. If $\mu(v_1) = \mu_I(w_i)$ then the partial mapping $\{(v_1, w_i)\}$ represents a subgraph isomorphism and the algorithm continues with the second vertex $v_2$ in $V$. Clearly, $v_2$ cannot be mapped to $w_i$ because $w_i$ is already mapped to by $v_1$. Thus, $v_2$ is mapped to some input vertex $w_j$ in $V_I - \{w_i\}$. If $\mu(v_2) = \mu_I(w_j)$ and, furthermore, if for each edge $e = (v_1, v_2) \in E$ there exists a corresponding edge $e_I = (w_i, w_j) \in E_I$ with $\nu(e) = \nu(e_I)$ then the partial mapping $\{(v_1, w_i), (v_2, w_j)\}$ represents a subgraph isomorphism from the subgraph of $G$ induced by the vertices $v_1, v_2$ to the input $G_I$. This process is continued until either all vertices $v_1, \ldots, v_M$ in $V$ are successfully mapped onto $V_I$ and a subgraph isomorphism from $G$ to $G_I$ is found or until for some vertex $v_n$, no corresponding vertex in $V_I - \{w_i, w_j, \ldots, w_k\}$ exists, with $\{w_i, w_j, \ldots, w_k\}$ being the vertices in $V_I$ that are mapped to by $v_1, v_2, \ldots, v_{n-1}$. In both cases, the algorithm backtracks to a previous vertex of $G$ and tries to remap it. Although this algorithm finds all subgraph isomorphisms from a small model to a small input graph in reasonable time, it performs poorly when the underlying graphs become larger. One of the reasons for this behavior is that the algorithm only tests for the subgraph isomorphism conditions in the partial mapping, but does not consider the vertices of the model that have not yet been mapped. In order to overcome this problem, Ullman proposed a forward-checking procedure. In this procedure, it is checked for each vertex of the model whether it can be mapped onto at least one vertex of the input graph such that the subgraph isomorphism conditions are locally true. If, for some partial mapping $\{(v_1, w_i), \ldots, (v_n, w_l)\}$, there is a vertex $v_{n'} \in V$ with $n' > n$ and the forward-checking procedure reveals that $v_{n'}$ cannot be mapped onto any vertex in $V_I - \{w_i, \ldots, w_l\}$ then the algorithm backtracks immediately and a possibly large number of computation steps is avoided. In general, it can be said that the forward-checking procedure introduced by Ullman greatly reduces the number of partial mappings that are generated during the search for subgraph isomorphisms.

We now analyze the computational complexity of Ullman's algorithm based on the description given above and the quantities introduced in Section 4. The definitions of the best and the worst case are identical to the definitions used in Section 4. In the best case, the model graph consists of $M$ uniquely labeled vertices while the input graph consists of $I$ vertices and $M$ different labels, i.e., each label is given to $O(I/M)$ vertices of the input graph. Thus, in Ullman's algorithm, each vertex of the model can be successfully mapped to $(I/M)$ vertices of the input graph but only one of these mappings also satisfies the edge constraints and can be extended into a larger partial mapping. As there are $O(M)$ vertices in the model, and for each vertex there are $O(M)$ edge constraints that must be tested in the forward-checking procedure, the total number of computational steps is bounded by $O(M^2 I/M) = O(IM)$. Furthermore, due to the fact that Ullman's algorithm can only by applied to two graphs at a time, it is linearly dependent on the number of models in the database, $N$. Hence, its computational complexity is in the best case bounded by:

$$O(NIM). \tag{11}$$

In the worst case, the model and the input graph vertices are unlabeled and each vertex is connected to every other vertex in the graph via an edge. Thus, for the first vertex there will be $O(I)$ mappings that can be extended with $O(I-1)$ mappings for the second vertex, resulting in a total of $O(I(I-1))$ successful mappings, and so on. As there are $O(M)$ vertices of the model graph and $O(I^{M-1})$ partial mappings that are generated for each vertex, the algorithm will investigate a total of $O(MI^{M-1})$ partial mappings. For each of these mappings, the forward-checking procedure tentatively maps $O(M)$ model vertices onto $O(I)$ input vertices in $O(IM)$ steps such that the total number of steps performed is $O(I^M M^2)$. Given a database with $N$ model graphs, the worst case complexity of Ullman's algorithm is therefore bounded by:

$$O(NI^M M^2). \tag{12}$$

## ACKNOWLEDGMENTS

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

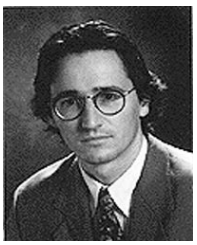
## REFERENCES

[1] D.H. Rouvray and A.T. Balaban, "Chemical Applications of Graph Theory," *Applications of Graph Theory*, R.J. Wilson and L.W. Beineke, eds., Academic Press, pp. 177-221, 1979.

[2] S. Bradtke and W.G. Lehnert, "Some Experiments with Case-Based Search," *Proc. Seventh Nat'l Conf. Artificial Intelligence*, vol. 1, pp. 133-138, 1988.

[3] J. Poole, "Similarity in Legal Case-Based Reasoning as Degree of Matching in Conceptual Graphs," *Preproc. First European Workshop*

*Case-Based Reasoning,* MM. Richter, S. Wess, K.-D. Althoff, and F. Maurer, eds., pp. 54-58, 1993.
[4] H. Ehrig, "Introduction to Graph Grammers with Applications to Semantic Networks," *Computers and Math. with Applications,* vol. 23, pp. 557-572, Sept. 1992.
[5] B. Bhanu and J.C. Ming, "TRIPLE: A Multistrategy Machine Learning Approach to Target Recognition," *Proc. Image Understanding Workshop,* pp. 537-547, 1988.
[6] D.H. Fisher, "Knowledge Acquisition via Incremental Conceptual Clustering," *Readings in Machine Learning,* J.W. Shavlik and T.G. Dietterich, eds., Morgan Kaufmann, pp. 267-283, 1990.
[7] D.J. Cook and L.B. Holder, "Substructure Discovery Using Minimum Description Length and Background Knowledge," *J. Artificial Intelligence Research,* pp. 231-255, 1994.
[8] B.T. Messmer and H. Bunke, "Automatic Learning and Recognition of Graphical Symbols in Engineering Drawings," *Proc. Int'l Workshop Graphics Recognition,* pp. 33-43, 1995.
[9] 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.
[10] H. Bunke, "Attributed Programmed Graph Grammars and Their Application to Schematic Diagram Interpretation," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 4, no. 6, pp. 574-582, 1982.
[11] 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.
[12] S.W. Lee and J.H. Kim, "Attributed Stroke Graph Matching for Seal Imprint Verification," *Pattern Recognition Letters,* vol. 9, pp. 137-145, Feb. 1989.
[13] A. Pearce, T. Caelli, and W.F. Bischof, "Rulegraphs for Graph Matching in Pattern Recognition," *Pattern Recognition,* vol. 27, no. 9, pp. 1,231-1,246, 1994.
[14] E. Gmuer and H. Bunke, "3-D Object Recognition Based on Subgraph Matching in Polynomial Time," *Structural Pattern Analysis,* R. Mohr, T. Pavlidis, and A. Sanfeliu, eds., World Scientific, pp. 131-147, 1990.
[15] R. Horaud and T. Skordas, "Structural Matching for Stereo Vision," *Proc. Ninth Int'l Conf. Pattern Recognition,* pp. 439-445, 1988.
[16] J.K. Cheng and T.S. Huang, "Image Recognition by Matching Relational Structures," *IEEE PRIP,* pp. 542-547, 1981.
[17] A.W. Wong, S.W. Lu, and M. Rioux, "Recognition and Shape Synthesis of 3-D Objects Based on Attributed Graphs," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 11, no. 3, pp. 279-290, Mar. 1989.
[18] C.J. Cho and J.J. Kim, "Recognizing 3-D Objects by Forward Checking Constrained Tree Search," *Pattern Recognition Letters,* vol. 13, no. 8, pp. 587-597, 1992.
[19] E.K. Wong, "Model Matching in Robot Vision by Subgraph Isomorphism," *Pattern Recognition,* vol. 25, no. 3, pp. 287-304, 1992.
[20] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, 1979.
[21] J.R. Ullman, "An Algorithm for Subgraph Isomorphism," *J. of the Assoc. for Computing Machinery,* vol. 23, no. 1, pp. 31-42, 1976.
[22] R.M. Haralick and G.L. Elliot, "Increasing Tree Search Efficiency for Constraint Satisfaction Problems," *Artificial Intelligence,* vol. 14, pp. 263-313, 1980.
[23] W.Y. Kim and A.C. Kak, "3-D Object Recognition Using Bipartite Matching Embedded in Discrete Relaxation, *IEEE Trans. Pattern Analysis and Machine Intelligence,"* vol. 13, pp. 224-251, 1991.
[24] B. Falkenhainer, K.D. Forbus, and D. Gentner, "The Structure-Mapping Engine: Algorithms and Examples," *Artificial Intelligence,* vol. 41, pp. 1-63, 1990.
[25] S.H. Myaeng and A. Lopez-Lopez, "Conceptual Graph Matching: A Flexible Algorithm and Experiments," *J. Experimental and Theoretical Artificial Intelligence,* vol. 4, pp. 107-126, Apr. 1992.
[26] R.E. Blake, "Partitioning Graph Matching with Constraints, *Pattern Recognition,"* vol. 27, no. 3, pp. 439-446, 1994.
[27] P. Kuner and B. Ueberreiter, "Pattern Recognition by Graph Matching–Combinatorial versus Continous Optimization," *Int'l J. Pattern Recognition and Artificial Intelligence,* vol. 2, no. 3, pp. 527-542, 1988.
[28] L. Herault, R. Horaud, F. Veillon, and J.J. Niez, "Symbolic Image Matching by Simulated Annealing," *Proc. British Machine Vision Conf.,* pp. 319-324, 1990.
[29] K.A. De Jong and W.M. Spears, "Using Genetic Algorithms to Solve NP-Complete Problems," *Genetic Algorithms,* J.D. Schaffer, ed., Morgan Kaufmann, pp. 124-132, 1989.
[30] D.E. Brown, C.L. Huntley, and A.R. Spillane, "A Parallel Genetic Heuristic for the Quadratic Assignment Problem," *Genetic Algorithms,* J. D. Schaffer, ed., Morgan Kaufmann, pp. 406-415, 1989.
[31] G.P. Ford and J. Zhang, "A Structural Graph Matching Approach to Image Understanding," *SPIE Intelligent Robots and Computer Vision X: Algorithms and Techniques,* vol. 1,607, pp. 559-569, 1991.
[32] 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.
[33] L.G. Shapiro and R.M. Haralick, "Organization of Relational Models for Scene Analysis," *IEEE Trans. Pattern Analysis and Machine Intelligence,* pp. 595-602, 1982.
[34] D.S. Seong, H.S. Kim, and K.H. Park, "Incremental Clustering of Attributed Graphs," *IEEE Trans. System, Man, and Cybernetics,* vol. 23, no. 5, pp. 1,399-1,411, 1993.
[35] H. Sossa and R. Horaud, "Model Indexing: The Graph-Hashing Approach," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* pp. 811-814, 1992.
[36] J.B. Burns and E.M. Riseman, "Matching Complex Images to Multiple 3D Objects Using View Description Networks," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* pp. 328-334, 1992.
[37] K. Sengupta and K.L. Boyer, "Organizing Large Structural Modelbases," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 17, no. 4, Apr. 1995.
[38] S. Paris, "Structural Recognition Using an Index," *Proc. Seventh Int'l Conf. Image Analysis and Processing: Progress in Image Analysis and Processing III,* S. Impedovo, ed., pp. 258-265, 1993.
[39] R. Levinson, "Pattern Associativity and the Retrieval of Semantic Networks," *Computers and Math. with Applications,* vol. 23, pp. 573-600, Sept. 1992.
[40] C.L. Forgy, "Rete, a Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *Artificial Intelligence,* vol. 19, pp. 17-37, 1982.
[41] H.S. Lee and M.I. Schor, "Match Algorithms for Generalized Rete Networks," *Artificial Intelligence,* pp. 255-270, 1992.
[42] M.A. Kelly and R.E. Seviora, "An Evaluation of DRETE on CUPID for OPS5 Matching," *Proc. 11th Int'l Joint Conf. Artificial Intelligence,* vol. 1, pp. 84-90, 1989.



**Bruno T. Messmer** received a doctoral degree from the University of Berne in 1996 for his work in the area of pattern recognition and graph matching. He is an artificial intelligence and software engineering expert working for the Corporate Technology Unit of Swisscom. He has published more than 20 articles on the subject of efficient graph matching algorithms and software frameworks. Currently, he is working in the area of voice-controlled telephone services, intelligent agents, and electronic commerce applications. His interests include object-oriented technologies, Java and C++, Internet applications and, in general, the application of AI techniques to the telecommunication domain.

**Horst Bunke** received the MS and PhD degrees in computer science from the University of Erlangen, Germany in 1974 and 1979, respectively. In 1984, he joined the University of Bern, Switzerland, where he is a full professor in the Computer Science Department. He was department chair from 1992 to 1996. From 1997 to 1998, he was dean of the Faculty of Science. Dr. Bunke is a fellow and a current vice president of the International Association for Pattern Recognition (IAPR). He is associate editor of the *International Journal on Document Analysis and Recognition*, 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*. 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.