Error-Tolerant Graph Matching: A Formal Framework and Algorithms

H. Bunke

Department of Computer Science,University of Bern,
Neubrückstr. 10, CH-3012 Bern, Switzerland
bunke@iam.unibe.ch

Abstract. This paper first reviews some theoretical results in error-tolerant graph matching that were obtained recently. The results include a new metric for error-tolerant graph matching based on maximum common subgraph, a relation between maximum common subgraph and graph edit distance, and the existence of classes of cost functions for error-tolerant graph matching. Then some new optimal algorithms for error-tolerant graph matching are discussed. Under specific conditions, the new algorithms may be significantly more efficient than traditional methods.

Keywords: structural pattern recognition, graphs, graph matching, error-tolerant matching, edit distance, maximum common subgraph, cost function

1 Introduction

In structural pattern recognition, graphs are widely used for object representation. Typically, parts of a complex object, or pattern, are represented by nodes, and relations between the various parts by edges. Labels and/or attributes for nodes and edges are used to further distinguish between different classes of nodes and edges, or to incorporate numerical information in the symbolic representation. Using graphs to represent both known models from a database and unknown input patterns, the recognition task turns into a graph matching problem. That is, the database is searched for models that are similar to the unknown input graph.

Standard algorithms for graph matching include graph isomorphism, subgraph isomorphism, and maximum common subgraph search [1, 2]. However, in real world applications we can't always expect a perfect match between the input and one of the graphs in the database. Therefore, what is needed is an algorithm for error-tolerant matching, or equivalently, a method that computes a measure of similarity between two given graphs.

Numerous applications of exact and error-tolerant graph matching have been described in the literature. In the reviews [3, 4] applications such as character

recognition, schematic drawing analysis, 2-D shape analysis, stereo matching, interpretation of 3-D objects, dynamic scene analysis, machine learning and discovery, medical image analysis and chip inspection have been reported. Additional applications are described in [5, 6, 7, 8, 9].

In this paper we review some recent work in the area of graph matching with an emphasis on error-tolerant graph matching. Basic definitions and notations are introduced in Section 2. Then in Section 3, some theoretical results are presented, showing relations between the concepts of graph edit distance and maximum common subgraph. Recent algorithms for error-tolerant graph matching are discussed in Section 4. Finally, in Section 5 some conclusions are drawn.

2 A Formal Framework for Error-Tolerant Graph Matching

There are various notations and definitions of error-tolerant graph matching that can be found in the literature. In this section we introduce a simple formal framework following [10]. Let L be a set of labels for nodes and edges. (This set may consist of objects of any type, for example, symbols from a finite alphabet, numbers, vectors a.s.o.)

Def. 1: A *graph* is a triple $g = (V, \alpha, \beta)$ where

- V is the finite set of *nodes*,
- $\alpha : V \rightarrow L$ is the node labeling function,
- $\beta : V \times V \rightarrow L$ is the edge labeling function. □

The set of *edges* E is implicitly given by assuming that our graphs are fully connected, i.e., $E = V \times V$. In other words, there exists exactly one edge between any pair of nodes. This assumption is for notational convenience only and doesn't restrict generality. If it is necessary to model the situation where edges exist only between distinguished pairs of nodes, we include a special label *null* in the set of labels, L. Edges are directed, i.e., edge (x, y) originates at node $x \in V$ and terminates at node $y \in V$. An undirected graph is obtained as a special case if $\beta(x, y) = \beta(y, x)$ for any $x, y \in V$. Node and edge labels come from the same alphabet, for notational convenience. If node and edge labels need to be explicitly distinguished, the set L can be partitioned into two disjoint subsets. If $V = \emptyset$ then g is called the *empty graph*.

Def. 2: Let $g = (V, \alpha, \beta)$ and $g' = (V', \alpha', \beta')$ be two graphs; g' is a *subgraph* of g, $g' \subseteq g$, if

- $V' \subseteq V$,
- $\alpha'(x) = \alpha(x)$ for all $x \in V'$,
- $\beta'((x, y)) = \beta((x, y))$ for all $(x, y) \in V' \times V'$. □

From Def. 2 it follows that, given a graph $g = (V, \alpha, \beta)$, any subset $V' \subseteq V$ of its vertices uniquely defines a subgraph. This subgraph is called the subgraph that is *induced* by V'.

Def. 3: Let $g_1 = (V_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, \alpha_2, \beta_2)$ be two graphs. A *graph isomorphism* between g_1 and g_2 is a bijective mapping $f : V_1 \to V_2$ such that

- $\alpha_1(x) = \alpha_2(f(x))$ for all $x \in V_1$,
- $\beta_1((x, y)) = \beta_2((f(x), f(y)))$ for all $(x, y) \in V_1 \times V_1$. □

If $V_1 = V_2 = \emptyset$, then f is called the *empty graph isomorphism*. If $f : V_1 \to V_2$ is a graph isomorphism between g_1 and g_2, and g_2 is a subgraph of another graph g_3, i.e. $g_2 \subseteq g_3$, then f is termed a *subgraph isomorphism* from g_1 to g_3.

Def. 4: Let g, g_1 and g_2 be graphs. The graph g is a *common subgraph* of g_1 and g_2 if there exist subgraph isomorphisms from g to g_1 and from g to g_2. □

Def. 5: Let g_1 and g_2 be two graphs. A graph g is called a *maximum common subgraph* of g_1 and g_2 if g is a common subgraph of g_1 and g_2, and there exists no other common subgraph of g_1 and g_2 that has more nodes than g. □

Def. 6: Let $g_1 = (V_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, \alpha_2, \beta_2)$ be two graphs. An *error-tolerant graph matching* (*etgm*) from g_1 to g_2 is a bijective function $f : \hat{V}_1 \to \hat{V}_2$, where $\hat{V}_1 \subseteq V_1$ and $\hat{V}_2 \subseteq V_2$. □

We say that node $x \in \hat{V}_1$ is *substituted* by node $y \in \hat{V}_2$ if $f(x) = y$. If $\alpha_1(x) = \alpha_2(f(x))$ then the substitution is called an *identical* substitution. Otherwise it is termed a *non-identical* substitution. Any node from $V_1 - \hat{V}_1$ is *deleted* from g_1, and any node from $V_2 - \hat{V}_2$ *inserted* in g_2 under f. We will use $\hat{g}_1$ and $\hat{g}_2$ to denote the subgraphs of g_1 and g_2 that are induced by the sets $\hat{V}_1$ and $\hat{V}_2$, respectively.

The mapping f *directly* implies an edit operation on each node in g_1 and g_2. I.e., nodes are substituted, deleted, or inserted, as described above. Additionally, the mapping f *indirectly* implies edit operations on the edges of g_1 and g_2. If $f(x_1) = y_1$ and $f(x_2) = y_2$, then edge (x_1, x_2) will be substituted by edge (y_1, y_2). If a node x is deleted from g_1, then any edge incident to x is deleted, too. Similarly, if a node x' is inserted in g_2, then any edge incident to x' is inserted, too. Obviously, any *etgm* f can be understood as a set of edit operations (substitutions, deletions, and insertions of both nodes and edges) that transform a given graph g_1 into another graph g_2.

Example 1: A graphical representation of two graphs is given in Fig. 1. For these graphs, we have:

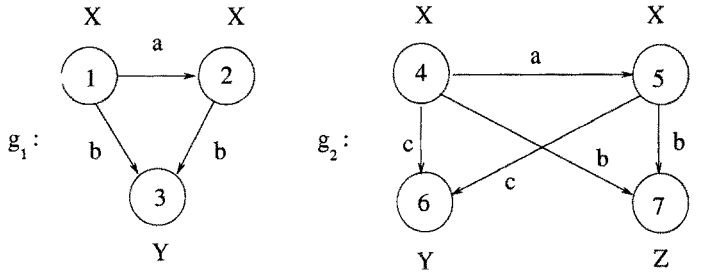

Fig. 1. Two graphs g_1 and g_2

$V_1 = \{1,2,3\}$; $V_2 = \{4,5,6,7\}$; $L = \{X, Y, Z, a, b, c, null\}$.
α_1: $1 \mapsto X$, $2 \mapsto X$, $3 \mapsto Y$.
α_2: $4 \mapsto X$, $5 \mapsto X$, $6 \mapsto Y$, $7 \mapsto Z$.
β_1: $(1,2) \mapsto a$, $(1,3) \mapsto b$, $(2,3) \mapsto b$.
β_2: $(4,5) \mapsto a$, $(4,6) \mapsto c$, $(4,7) \mapsto b$, $(5,6) \mapsto c$, $(5,7) \mapsto b$.

All other edges are labeled with *null* and not shown in Fig. 1.
Three examples of *etgm* are:

- $f_1 : 1 \mapsto 4, 2 \mapsto 5, 3 \mapsto 7$ with $\hat{V_1} = \{1,2,3\}$ and $\hat{V_2} = \{4,5,7\}$
- $f_2 : 1 \mapsto 4, 2 \mapsto 5, 3 \mapsto 6$ with $\hat{V_1} = \{1,2,3\}$ and $\hat{V_2} = \{4,5,6\}$
- $f_3 : 1 \mapsto 4, 2 \mapsto 5$ with $\hat{V_1} = \{1,2\}$ and $\hat{V_2} = \{4,5\}$

Under f_1, nodes 1,2 and 3 are substituted by nodes 4,5, and 7, respectively. Consequently, edges $(1,2), (1,3)$, and $(2,3)$ are substituted by $(4,5), (4,7)$, and $(5,7)$, respectively. The substitution of nodes 1 and 2 by 4 and 5 are identical substitutions that involve no label change; there are no label changes involved in the edge substitutions, either. The label Y of node 3 is substituted by Z of node 7, and node 6 together with its incident edges $(4,6)$ and $(5,6)$ is inserted in g_2. There are, of course, many other *etgm's* from g_1 to g_2. □

Def. 7: The *cost* of an *etgm* f: $\hat{V_1} \rightarrow \hat{V_2}$ from a graph $g_1 = (V_1, \alpha_1, \beta_1)$ to a graph $g_2 = (V_2, \alpha_2, \beta_2)$ is given by

$$\gamma(f) = \sum_{x \in V_1 - \hat{V_1}} c_{nd}(x) + \sum_{x \in V_2 - \hat{V_2}} c_{ni}(x) + \sum_{x \in \hat{V_1}} c_{ns}(x) +$$

$$\sum_{e \in \bar{E}_1} c_{ed}(e) + \sum_{e \in \bar{E}_2} c_{ei}(e) + \sum_{e \in \hat{V_1} \times \hat{V_1}} c_{es}(e)$$

where

- $c_{nd}(x)$ is the cost of deleting a node $x \in V_1 - \hat{V_1}$ from g_1,
- $c_{ni}(x)$ is the cost of inserting a node $x \in V_2 - \hat{V_2}$ in g_2,
- $c_{ns}(x)$ is the cost of substituting a node $x \in \hat{V_1}$ by $f(x) \in \hat{V_2}$,
- $c_{ed}(e)$ is the cost of deleting an edge from $\bar{E_1}$ in g_1,
- $c_{ei}(e)$ is the cost of inserting an edge in $\bar{E_2}$ in g_2,

- $c_{es}(e)$ is the cost of substituting an edge $e = (x, y) \in \hat{V}_1 \times \hat{V}_1$ by $e' = (f(x), f(y)) \in \hat{V}_2 \times \hat{V}_2$,
- $\bar{E}_1 = (V_1 \times V_1) - (\hat{V}_1 \times \hat{V}_1)$, $\bar{E}_2 = (V_2 \times V_2) - (\hat{V}_2 \times \hat{V}_2)$.

All costs are non-negative real numbers. □

The costs introduced in Def. 7 are used to model the likehood of errors and distortions that may corrupt ideal patterns. Typically, the more likely a certain distortion is to occur, the lower is its cost. Concrete values of $c_{nd}, c_{ni}, \ldots, c_{es}$ have to be chosen dependent on the particular application. Note that the deletion of edges from $\hat{V}_1 \times \hat{V}_1$ and the insertion of edges in $\hat{V}_2 \times \hat{V}_2$ is modeled via edge substitutions (replacing label $l \neq null$ by *null*, or replacing *null* by $l \neq null$).

Under the simple model introduced in this paper, the deletion, insertion, and substitution of an edge form $\hat{V}_1 \times \hat{V}_1$ all necessarily have the same cost. However, it is straightforward to extend this model by introducing individual substitution costs for the edges from $\hat{V}_1 \times \hat{V}_1$ depending on the edge label. Thus different costs can be assigned to the deletion, insertion and substitution of an edge from $\hat{V}_1 \times \hat{V}_1$.

Usually it is assumed that the costs $c_{nd}(x), c_{ni}(x)$, and $c_{ns}(x)$ don't depend on node x; neither do $c_{ed}(e), c_{ei}(e)$, and $c_{es}(e)$ depend on edge e. In other words, $c_{nd}(x), c_{ni}(x)$, and $c_{ns}(x)$ will be the same for any node x, and $c_{ed}(e), c_{ei}(e)$, and $c_{es}(e)$ will be the same for any edge e. Thus the notation $c_{nd}(x) = c_{nd}, c_{ni}(x) = c_{ni}, \ldots, c_{es}(e) = c_{es}$ will be used in the following. The tuple $C = (c_{nd}, c_{ni}, c_{ns}, c_{ed}, c_{ei}, c_{es})$ is called a *cost function*. If the cost function C is to be explicitly mentioned, the notation $\gamma_C(f)$ is used instead of $\gamma(f)$. Usually it is assumed that the cost of an identical node or edge substitution is zero, while the cost of any other edit operation is greater the zero.

Def. 8: Let f be an *etgm* from g_1 to g_2 and C a cost function. We will call f an *optimal etgm* under C if there is no other *etgm* f' from g_1 to g_2 with $\gamma_C(f') < \gamma_C(f)$. □

The cost of an optimal *etgm*, $\gamma_C(f)$, is also called the *edit distance* between g_1 and g_2, $d(g_1, g_2)$. It corresponds to the minimum cost sequence of graph edit operations that transform one graph into the other. For a given cost function C there are usually several optimal *etgm's* from a graph g_1 to another graph g_2.

Example 2: Consider cost function $C = (c_{nd}, c_{ni}, c_{ns}, c_{ed}, c_{ei}, c_{es}) = (1, 1, 1, 1, 1, 1)$. Then the *etgm* f_1 given in Example 1 has cost $\gamma_C(f_1) = 4$ (one node label substitution, one node insertion, and two edge insertions). It can be easily verified that there is no other *etgm* from g_1 to g_2 that has a smaller cost under C.□

In a typical application that involves *etgm* we are given a database of model graphs $g_1, \ldots, g_n$ and an input graph g, and we match g with $g_1, \ldots, g_n$ in order to find the model g_i that is most similar to g. Using the terminology introduced

above, we are looking for the g_i that has the smallest edit distance $d(g, g_i)$ to g. One crucial observation in the context of this task is the fact that, given two graphs g and g', their edit distance $d(g, g')$ crucially depends on the underlying cost function. Following is an example.

Example 3: The *etgm* f_1 given in Example 1 is optimal under the cost function $C = (1, 1, 1, 1, 1, 1)$; see Example 2. However, under cost function $C' = (1, 1, 3, 1, 1, 1)$ we observe $\gamma_{C'}(f_1) = 6$ and $\gamma_{C'}(f_2) = 5$. Thus f_1 is no longer optimal. Actually it can be easily verified that f_2 is optimal under C'. If we consider a third cost function $C'' = (1, 1, 7, 1, 1, 7)$ then f_3 becomes optimal. □

3 Error-Tolerant Graph Matching and Maximum Common Subgraph

As it was noticed in the last section, the underlying cost function has an important influence on optimal *etgm*. For practical applications, unfortunately, there is no automatic procedure known today in order to derive a cost function, suitable for a given task, from a set of samples. Typically, cost functions are defined in an ad hoc manner, purely guided by heuristics and intuition. In order to avoid the problem of finding a suitable cost function, a new graph distance measure based on the maximum common subgraph of two graphs was proposed in [11]. Given two non-empty graphs, g_1 and g_2, their distance is defined as

$$\delta(g_1, g_2) = 1 - \frac{|mcs(g_1, g_2)|}{max(|g_1|, |g_2|)} \tag{1}$$

In this definition, $mcs(g_1, g_2)$ denotes the maximum common subgraph of g_1 and g_2, and $|g|$ is the number of nodes of a graph g. First of all, one notices that no edit operations are involved in eq.(3.1) and, consequently, no cost function has to be defined to compute $\delta(g_1, g_2)$ for any two given graphs g_1 and g_2. Moreover, it was shown in [11] that the distance in eq.(3.1) has some interesting properties. In particular it is a metric satisfying

$$0 \leq \delta(g_1, g_2) \leq 1 \tag{2}$$

$$\delta(g_1, g_2) = 0 \Leftrightarrow g_1 \text{ and } g_2 \text{ are isomorphic to each other} \tag{3}$$

$$\delta(g_1, g_2) = \delta(g_2, g_1) \tag{4}$$

$$\delta(g_1, g_3) \leq \delta(g_1, g_2) + \delta(g_2, g_3) \tag{5}$$

for any three graphs g_1, g_2 and g_3. It is known that the edit distance $d(g_1, g_2)$ introduced in Section 2 is a metric if and only if the underlying cost function satisfies certain conditions. These conditions, however, may be too restrictive or counterintuitive for certain problem domains. But there are applications where

metric properties of the underlying distance measure are very much desired. One example is information retrieval from image and video databases [6]. This area relies heavily on browsing to locate required database elements. Thus it is necessary for the distance measure to be well behaved to allow sensible navigation of the database. For example, property (2) makes sure that the range of all possible distances is known in advance, regardless of the particular objects to be compared. By means of property (3) objects have zero distance if and only if they are identical. Eq.(4) implies that the distance from any object A to any object B is the same as from B to A. Finally, because of the triangular inequality (5), we know that no two objects that are dissimilar to each other can be both similar to the same object.

The graph distance measure according to eq.(1) is based on the maximum common subgraph of two graphs. Obviously, it can be regarded an alternative to graph edit distance as introduced in Section 2. However, it was recently shown that there is also a direct relation between graph edit distance and maximum common subgraph in the sense that graph edit distance and maximum common subgraph computation are equivalent to each other under a certain cost function [12]. In [12] the following cost function was considered:

$$c_{ns}(x) = \left\{ \begin{array}{l} 0, \text{ if } \alpha_1(x) = \alpha_2(f(x)) \\ \infty, \text{ otherwise} \end{array} \right\} \text{for any } x \in \hat{V}_1,$$

$c_{nd}(x) = 1$ for any $x \in V_1 - \hat{V}_1$,
$c_{ni}(x) = 1$ for any $x \in V_2 - \hat{V}_2$

$$c_{es}(e) = \left\{ \begin{array}{l} 0, \text{ if } \beta_1((x,y)) = \beta_2((f(x),f(y))) \\ \infty, \text{ otherwise} \end{array} \right\} \text{for any } e = (x,y) \in \hat{V}_1 \times \hat{V}_1, \quad (6)$$

$c_{ed}(e) = 0$ for any $e = (x,y) \in (V_1 \times V_1) - (\hat{V}_1 \times \hat{V}_1)$,
$c_{ei}(e) = 0$ for any $e = (x,y) \in (V_2 \times V_2) - (\hat{V}_2 \times \hat{V}_2)$.

Under this cost function, any node deletion and insertion has a cost equal to one. Identical node and edge substitutions have zero cost, while substitutions involving different labels have infinity cost. The insertion or deletion of an edge incident to a node that is inserted or deleted, respectively, has no cost. As for any two graphs $g_1 = (V_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, \alpha_2, \beta_2)$ there is always an *etgm* f with cost $c(f) = |V_1| + |V_2|$ (corresponding to the case where all nodes together with their incident edges are deleted from g_1, and all nodes with their incident edges are inserted in g_2), any edit operation with infinity cost will never need to be considered when looking for an optimal *etgm*. Thus we may think of edit operations with infinity cost as non-admissible. In other words, under the given cost function we can restrict our attention on *etgm's* involving only insertions, deletions and identical node and edge substitutions, but no non-identical substitutions. For example, for the *etgm* f_3 discussed in Example 1, we have $c(f_3) = 3$ under the considered cost function. Obviously both f_1 and f_2 have infinity cost.

It was shown in [12] that under this cost function the following equation holds true for any two graphs g_1 and g_2, and a maximum common subgraph g of g_1 and g_2 (this maximum common subgraph may be empty):

$$d(g_1, g_2) = |g_1| + |g_2| - 2|g| \tag{7}$$

Obviously, this equation establishes a relation between the size $|g|$ of the maximum common subgraph of two graphs g_1 and g_2, and their edit distance $d(g_1, g_2)$. Thus given one of the two quantities and the size of g_1 and g_2, we can immediately calculate the other. It was furthermore shown in [12] that the mapping $f : \hat{V}_1 \rightarrow \hat{V}_2$, defining an optimal *etgm* according to Def. 8, represents a maximum common subgraph of g_1 and g_2. I.e., f is a graph isomorphism between $\hat{g}_1$, the graph induced by $\hat{V}_1$, and $\hat{g}_2$, the graph induced by $\hat{V}_2$, and there are no larger subgraphs in g_1 and g_2, respectively, that are isomorphic to each other.

This theoretical result has an interesting practical consequence, namely, any algorithm for graph edit distance computation can be applied for maximum common subgraph computation if it is run under the cost function given in (6). Conversely, any algorithm that computes the maximum common subgraph of two graphs can be used for graph edit distance computation under cost function (6), using formula (7). A similar relation between string edit distance and longest common subsequence has been known for long [13].

The results derived in [12] were recently shown to hold not only for the cost function given in (6), but for a whole class of infinitely many cost functions. In [10] cost functions C with $c_{ns} = c_{es} = 0$ for identical substitutions and

$$c_{nd} + c_{ni} < c_{ns} \quad \text{and} \quad c_{nd} + c_{ni} < c_{es} \tag{8}$$

are considered. (Note that (6) is a special case of this class.) It is shown that for this whole class of cost functions the minimum cost mapping $f : \hat{V}_1 \rightarrow \hat{V}_2$ represents a maximum common subgraph of g_1 and g_2 and, conversely, any maximum common subgraph represents a minimum cost mapping in the sense of Def. 8. Intuitively speaking, the conditions in (8) imply that a node deletion together with a node insertion will be always preferred over a node or an edge substitution because of a smaller cost. This means that all nodes and edges in g_1 that can't be mapped to a node or an edge with an identical label in g_2 will be deleted from g_1. Similarly, all nodes and edges in g_2 that are not part of the mapping f (i.e, that don't have a corresponding node or edge with identical label, respectively) will be inserted. What remains for the mapping f is exactly the maximum common subgraph of g_1 and g_2. An example is the *etgm* f_3 in Example 1. It is optimal under the cost function $C'' = (1, 1, 7, 1, 1, 7)$ as explained in Example 3. As a matter of fact, f_3 corresponds to the maximum common subgraph of g_1 and g_2 in Fig. 1, and cost function C'' satisfies conditions (8).

The equivalence of maximum common subgraph and graph edit distance computation shown in [10] is based on the assumption $c_{ei}(e) = c_{ed}(e) = 0$ for any

edge e from $\bar{E}_1$ and $\bar{E}_2$, respectively (see Def. 7). Thus, no individual costs for the deletion of edges from $\bar{E}_1$, and no individual costs for the insertion of edges in $\bar{E}_2$ are taken into regard. The reason is that these operations are automatically implied by the deletion of nodes from $(V_1 - \hat{V}_1)$, and the insertion of nodes in $(V_2 - \hat{V}_2)$, respectively. Thus it is assumed that their costs are included in the costs of the corresponding node deletions and insertions. In other words, the cost of a node deletion (insertion) includes not only the cost of deleting (inserting) a node, but also the deletion (insertion) of the edges that connect it to the other nodes of the graph. This assumption may be justified in many applications.

The equivalence of graph edit distance and maximum common subgraph shown in [10] yields additional insight on the measure $\delta(g_1, g_2)$ of eq. (1). Although no *explicit* costs of graph edit operations are needed to compute $\delta(g_1, g_2)$, there are, nevertheless, costs involved in an *implicit* fashion, because the quantity $|mcs(g_1, g_2)|$ in (1) is equivalent to the graph edit distance $d(g_1, g_2)$ in the sense of eq. (7), assuming a cost function satisfying (8). In other words, whenever we compute the maximum common subgraph of two graphs we may consider this as a graph edit distance computation under an arbitary cost function belonging to the class studied in [10]. From this point of view, the measure defined in (1) may be still regarded an advantage over convential graph edit distance computation because it is robust against changing the costs of the underlying graph edit operations in a fairly wide range.

Another important result shown in [10] is the existence of classes of cost functions that always result in the same optimal mapping $f : \hat{V}_1 \rightarrow \hat{V}_2$ for any two given graphs g_1 and g_2. Intuitively speaking, if we consider two cost functions C and C', where C' is a scaled version of C, i.e., $c'_{nd} = \alpha c_{nd}, c'_{ni} = \alpha c_{ni}, \ldots, c'_{ei} = \alpha c_{ei}$ for some $\alpha > 0$, then we expect that any *etgm* f that is optimal under C is also optimal under C' for any two given graphs g_1 and g_2. Just the absolute cost of the two optimal *etgm's* would differ by a factor α, i.e., $\gamma_{C'}(f) = \alpha\gamma_C(f)$. In [10] it was shown that any optimal *etgm* under a cost function C is optimal under another cost function C' not only if C' is a scaled version of C, but for a much larger class of cost functions C'. If the conditions

$$(c_{ni} + c_{nd})/c_{ns} = (c'_{ni} + c'_{nd})/c'_{ns} \qquad \text{and} \tag{9}$$

$$c_{es}/c_{ns} = c'_{es}/c'_{ns} \tag{10}$$

for cost functions C and C' are satisfied then any *etgm* f is optimal under C if and only if it is optimal under C' for any two given graphs g_1 and g_2. Furthermore, there is a relation between $\gamma_C(f)$ and $\gamma_{C'}(f)$ that is similar to eq. (7). Given the edit distance under cost function C we can analytically compute the edit distance under C' using just the parameters of C and C' and the size of the two graphs under consideration. Hence, given an algorithm that was designed for a particular cost function C, we can use the same algorithm for any other cost function C' for which (9) and (10) are satisfied. The existence of similar classes of cost functions for string edit distance has been discovered recently [14].

4 Algorithms for Error-Tolerant Graph Matching

All results presented in Section 3 are independent of the algorithm that is actually employed for graph edit distance or maximum common subgraph computation. In the past, various approaches to *etgm* have been proposed. The most common approach is based on tree search with A^*-like algorithms [15]. The search space of the A^* algorithm can be greatly reduced by applying heuristic error estimation functions. Numerous heuristics have been proposed [16, 17, 18, 19, 20]. All of this methods are guaranteed to find the optimal solution but require exponential time in the worst case. Suboptimal, or approximative 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 [21, 22] probabilistic relaxation schemas are described. Other approaches are based on neural networks such as the Hopfield network [23] or the Kohonen map [24]. Also genetic algorithms have been proposed recently [25, 26]. In [27] an approximate method based on maximum flow is introduced. However, all of these approximate methods may get tracked in local minima and miss the optimal solution. Optimal algorithms to find a maximum common subgraph of two graphs are based on maximum clique detection [1] or backtracking [28]. A suboptimal method using a neural network has been reported in [29].

In the remainder of this section we briefly review three optimal methods for *etgm* that were proposed recently. In [30, 31] a new method is described for matching a graph g against a database of model graphs $g_1, \ldots, g_n$ in order to find the model g_i with the smallest edit distance $d(g, g_i)$ to g. The basic assumption is that the models in the database are not completely dissimilar. Instead, it is supposed that there are graphs $s'_j s$ that occur simultaneously as subgraphs in several of the $g'_i s$, or multiple times in the same g_i. Under a naive procedure, we will match g sequentially with each of the $g'_i s$. However, because of common subgraphs s_j shared by several models g_i the $s'_j s$ will be matched with g multiple times. This clearly implies some redundancy.

In the approach described in [30, 31] the model graphs $g_1, \ldots, g_n$ are preprocessed generating a symbolic data structure, called *network* of models. This network is a compact representation of the models in the sense that multiple occurences of the same subgraph s_j are represented only once. Consequently, such subgraphs will be matched only once with the input. Hence the computational effort will be reduced. A further enhancement of the computational efficiency of the method is achieved by a lookahead procedure. This lookahead procedure returns an estimation of the future matching cost. It is precise and can be efficiently computed based on the network.

Figs. 2 and 3 show the results of an experiment that was done to compare the new method with a traditional A^*-based algorithm for *etgm*. In this experiment random graphs were used as input. Fig. 2 shows the computation time needed by the new and the traditional algorithm depending on a growing number of

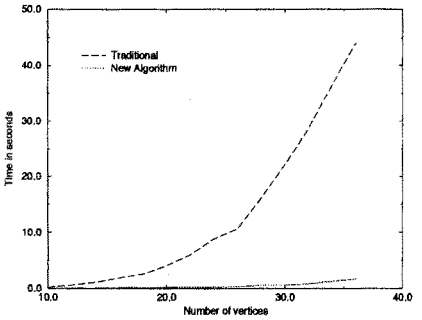

Fig. 2. Computation time depending on the size of the underlying graphs for constant edit distance.

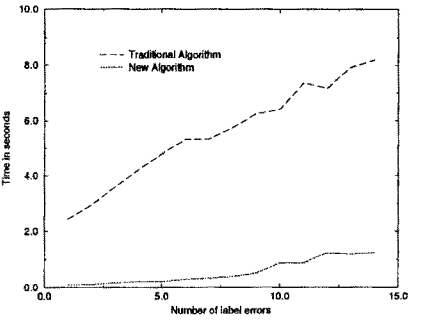

Fig. 3. Computation time depending on the edit distance for constant size ot the underlying graphs.

nodes in the graphs to be matched, keeping the number of errors, i.e., the edit distance between the two graphs, constant. Fig. 3 shows a similar experiment where the size of the underlying graphs is kept constant but their edit distance is increased. The figures clearly show the superior performance of the new method. For further experimental results and a more detailed discussion see [30, 31].

In [4, 30] a fast algorithm for graph and subgraph isomorphism detection was described. It is based on an intensive preprocessing step in which a database of model graphs is converted into a decision tree. At run time, the input graph is classified by the decision tree and all model graphs for which there exists a subgraph isomorphism from the input are detected. If we neglect the time needed for preprocessing, the computational complexity of the new subgraph isomorphism algorithm is only polynomial in the number of input graph vertices. Furthermore, it is independent of the number of model graphs and the number of edges in any of the graphs. However, the decision tree that is constructed in the preprocessing step is of exponential size in terms of the number of vertices of the model graphs. The actual implementation described by the authors is able to cope with a single graph in the database of up to 22 nodes, or up to 30 models in the database consisting of up to 11 nodes each.

Recently the decision tree method was extended from exact graph and subgraph isomorphism detection to *etgm* [32]. Actually, there are different possible approaches. In one approach, error correction is considered at the time of the creation of the decision tree. That is, for each model graph a set of distorted copies are created and compiled into the decision tree. The number of distorted copies depends on the maximal admissible error. At run time, the decision tree is used to classify the unknown input graph in the same way as in case of exact subgraph isomorphism detection. The time complexity of this procedure at run time is only quadratic in the number of input graph nodes. However, the size of the decision tree is exponential in the number of vertices of the model graphs and in the degree of distortion that is to be considered. Therefore, this approach is limited to (very) small graphs.

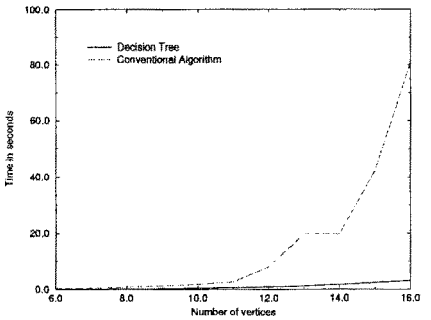

Fig. 4. Computation time in seconds for $\vartheta = 1$ and a growing number of vertices

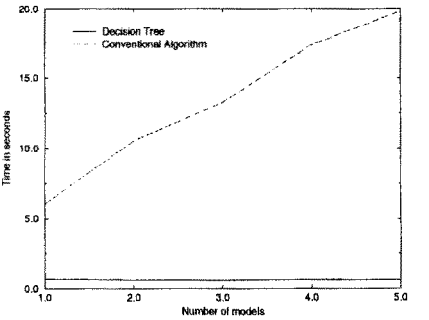

Fig. 5. Computation time in seconds for $\vartheta = 1$ and a growing number of models.

In the second approach, the error corrections are considered at run time only. That is, the decision tree for a set of model graphs does not incorporate any information about possible errors. Hence, the decision tree compilation step is identical to the original preprocessing step and, consequently, the size of the decision tree is exponential only in the size of the model graphs. At run time, a set of distorted copies of the input graph are constructed such that all possible error corrections up to a certain error threshold are considered. Each graph in this set is then classified by the decision tree. The run time complexity of this method is $O(\vartheta n^{2(\vartheta+1)})$ where n is the number of nodes in the input graph and ϑ is a threshold that defines the maximum number of admissible edit operations.

Figures 4 and 5 show the results of an experiment where the second approach was compared to a conventional A^*-based *etgm* algorithm. In this experiment the threshold was set to $\vartheta = 1$. The input graphs were generated by copying one of the model graphs and then inserting or deleting an edge. Fig. 4 shows the time needed by both algorithms when matching an input graph with one model graph depending on the number of nodes of the input and model. In Fig. 5 the input and the model graph consists of 11 vertices and the number of models is varied from 1 to 5. Fig. 5 confirms the result of the theoretical complexity analysis, i.e., the time complexity of the decision tree algorithm is independent of the number of models in the database. The present implementation is limited to graphs consisting of up to a maximum of 16 nodes in case of just one error. For further details and additional experimental results see [32].

The decision tree approach was furthermore extended to maximum common subgraph detection [7, 6]. For this problem it is necessary, unfortunately, to consider all permutations of the adjacency matrix of the input graph, which leads to an exponential time complexity at run time despite the fact that all permutations of the models have been encoded in the decision tree. Using a pruning strategy however, the run time of the resulting algorithm is still significantly better than that of traditional algorithms. For further details and experimental results see [6, 7].

5 Conclusions

Graph matching is an area where steady progress has been taking place for many years. Recently, the focus of attention has shifted from 'simple' combinatorial procedures, comparing two graphs at a time, to suboptimal stochastic algorithms and optimal algorithms that employ some kind of preprocessing to reduce the computational effort at run time. Future applications of graph matching with new challenges are emerging, for example, image and video database retrieval, or image sequence analysis. There are many interesting open problems in *etgm*, for example, the combination of stochastic and preprocessing based optimal methods, or a deeper study of the influence of the cost function on the complexity of matching algorithms.

References

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

[2] G. Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo 9*, pages 341–354, 1972.

[3] H. Bunke. Structural and syntactic pattern recognition. in C.H. Chen, L.F. Pau, P. Wang, *Handbook of Pattern Recognition and Computer Vision*, World Scientific Publ. Co., Singapore, 1993, 163–209.

[4] H. Bunke and B. Messmer. Recent advances in graph matching. *Int. Journal of Pattern Recognition and Art. Intell.*, Vol. 11, No. 1, 1997 169–203.

[5] H. Walischewski. Automatic knowledge acquisition for spatial document interpretation. *Proc. 4th ICDAR, Ulm*, 1997, 243–247.

[6] K.R. Shearer. Indexing and retrieval of video using spatial reasoning techniques. *PhD thesis, Curtin University of Technology, Perth, Australia*, 1998.

[7] K. Shearer, H. Bunke, S. Ventakesh and D. Kieronska. Efficient graph matching for video indexing. Accepted for publication in *Computing*, Springer Verlag, 1998.

[8] L. P. Cordella, P. Foggia, C. Sansone and M. Vento. Subgraph transformations for the inexact matching of attributed relational graphs. Accepted for publication in *Computing*, Springer Verlag, 1998.

[9] T. Lourens. A biologically plausible model for corner-based object recognition from color images. *PhD thesis*, University of Groningen, The Netherlands, 1998.

[10] H. Bunke. Error correcting graph matching: On the influence of the underlying cost function. Submitted for publication.

[11] H. Bunke and K. Shearer. A graph distance metric based on maximal common subgraph. Accepted for publication in *Pattern Recognition Letters*.

[12] H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters* 18, 1997, 689–694.

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

[14] S. Rice, H. Bunke and T. Nartker. Classes of cost functions for string matching. *Algorithmica*, Vol. 18 No. 2, 271–280, 1997.

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

[16] W.H. Tsai and K.S Fu. Error-correcting isomorphisms of attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 9:757–768, 1979.

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

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

[19] M.A. Eshera and K.S. Fu. A graph distance measure for image analysis. *IEEE Transactions on Systems, Man, and Cybernetics*, 14(3):398–408, May 1984.

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

[21] R. Wilson, E. Hancock. Graph matching by discrete relaxation. In E.S. Gelsema and L.N. Kanal, editors, *Pattern Recognition in Practice IV: Multiple Paradigms, Comparative Studies and Hybrid Systems*, pages 165–176. North-Holland, 1994.

[22] W.J. Christmas, J. Kittler, and M. Petrou. Structural matching in computer vision using probabilistic relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI*, 17(8):749–764, 1995.

[23] J. Feng, M. Laumy, and M. Dhome. Inexact matching using neural networks. In E.S. Gelsema and L.N. Kanal, editors, *Pattern Recognition in Practice IV: Multiple Paradigms, Comparative Studies and Hybrid Systems*, pages 177–184. North-Holland, 1994.

[24] L. Xu and E. Oja. Improved simulated annealing, Boltzmann machine, and attributed graph matching. In L. Almeida, editor, *Lecture Notes in Computer Science 412*, pages 151–161. Springer Verlag, 1990.

[25] A. Cross, R. Wilson, E. Hancock. Genetic search for structural matching. In B. Buxton, R. Cipolla (eds.): *Computer Vision - FCCV '96, Lecture Notes in Comp. Science 1064*, Springer Verlag, 1996, 514–525.

[26] Y. -K. Wang, K. -C Fan, J. -T Horng. Genetic-based search for error-correcting graph isomorphism. *IEEE Trans. on Systems, Man and Cybernetics*, Vol. 27, May, 1997, 588–597.

[27] I. Wang, K. Zhang, G. Chirn. The approximate graph matching problem. *Proc. 12th ICPR, Jerusalem 1994*, 284–288.

[28] J. Mc Gregor. Backtrack search algorithms and the maximal common subgraph problem. *Software-Practice and Experience*, Vol. 12, 1982, 23–34.

[29] A. Shonkry, M. Aboutabl. Neural network approach for solving the maximal common subgraph problem. *IEEE Trans. on Systems, Man and Cybernetics*, Vol. 26, 1996, 785–790.

[30] B. T. Messmer. Efficient graph matching algorithms for preprocessed model graphs. *PhD thesis*, University of Bern, Switzerland, 1995.

[31] B. Messmer and H. Bunke. A new algorithm for error tolerant aubgraph isomorphism. Accepted for publication in *IEEE Trans. PAMI.*

[32] B. T. Messmer and H. Bunke. Error-correcting graph isomorphism using decision trees. To appear in *Int. Journal of Pattern Recognition and Art. Intelligence.*