5

Graph Matching

X. Jiang[1] and H. Bunke[2]

[1] Department of Mathematics and Computer Science, University of Münster
Einsteinstrasse 62, D-48149 Münster, Germany
xjiang@math.uni-muenster.de
[2] Institute of Informatics and Applied Mathematics, University of Bern
Neubrückstrasse 10, CH-3012 Bern, Switzerland
bunke@iam.unibe.de

5.1 Introduction

Graphs are a powerful and universal tool widely used in information processing. Numerous methods for graph analysis have been developed. Examples include the detection of Hamiltonian cycles, shortest paths, vertex coloring, graph drawing, and so on [5]. In particular, graph representations are extremely useful in image processing and understanding, which is the complex process of mapping the initially numeric nature of an image (or images) into symbolic representations for subsequent semantic interpretation of the sensed world.

Case-based reasoning (CBR) provides us powerful strategies to fulfill the high demands on robustness, accuracy, and flexibility of image interpretation systems [53]. In CBR systems a concept is described by a case base and an associated similarity measure. Cases can be organized into a flat case base or in a hierarchical fashion. In a flat organization, we have to calculate similarity between the problem case and each case in memory. It is clear that this will take a considerable amount of time. To speed up the retrieval process, a more sophisticated, hierarchical organization of the case base is necessary. This organization should allow separating the set of similar cases from those cases not similar to the recent problem at the earliest stage of the retrieval process. In case base creation, maintenance, and retrieval, a central issue is that of case (object) similarity. In this chapter we are concerned with structural case representations [20, 54], which are common in computer vision and image interpretation [6], building design [31], timetabling [19], etc. Given such representations, we consider the problem of determining the equality or similarity of graphs, which is generally referred to as *graph matching*.

Standard concepts in *exact* graph matching include graph isomorphism and subgraph isomorphism. Two graphs are called *isomorphic* if they have identical structure. There exists a *subgraph isomorphism* between two graphs

X. Jiang and H. Bunke: *Graph Matching*, Studies in Computational Intelligence (SCI) **73**, 149–173 (2008)
www.springerlink.com

if one graph contains a subgraph that is isomorphic to the other. Subgraph isomorphism is useful to find out if a given object is part of another object or a collection of several objects. Although exact graph matching offers a rigorous way to describe structure equality in mathematical terms, it is generally only applicable to a restricted set of real-world problems. *Inexact*, or *error-tolerant*, graph matching methods, on the other hand, are able to cope with strong inner-class distortion, which is often present in real-world applications.

In this chapter we provide an overview of graph matching. We mainly concentrate on the fundamental concepts and some recent developments. The reader is referred to the recent survey [23] for a detailed discussion of the numerous graph matching algorithms and also the vast applications of graph matching. Other recent collections of papers on graph matching (and mining) can be found in [18, 24, 26, 29].

5.2 Basic Definitions and Notation

Attributed graphs with an unrestricted label alphabet is one of the most general ways to define graphs. It turns out that the definition given below is sufficiently flexible for a large variety of applications.

Definition 1 (Graph). *A graph is a 4-tuple* $g = (V, E, \alpha, \beta)$*, where*

- V *is the finite set of vertices*
- $E \subseteq V \times V$ *is the set of edges*
- $\alpha : V \rightarrow L_V$ *is a function assigning labels to the vertices*
- $\beta : E \rightarrow L_E$ *is a function assigning labels to the edges*

Edge (u, v) originates at node u and terminates at node v. The labeling function can be used to integrate information about nodes and edges into graphs by assigning attributes from L_V and L_E to nodes and edges, respectively. Usually, there are no constraints imposed on the label alphabets. In practical applications, however, label alphabets are often defined as vector spaces $\Re^k$ of a fixed dimension k or discrete sets of symbols $\{s_1, s_2, \ldots, s_k\}$. In principle, nodes and edges may also have other, arbitrarily complex labels. The notation $|g|$ will be used for the number of nodes of graph g.

The graph definition introduced above includes a number of special cases. To define undirected graphs, for instance, we require that $(v, u) \in E$ for every edge $(u, v) \in E$ such that $\beta(u, v) = \beta(v, u)$. In the case of nonattributed graphs, the label alphabets are defined by $L_V = L_E = \{\phi\}$, so that every node and edge gets assigned the null label ϕ.

For some applications, it is important to detect whether a smaller graph is present in a larger graph – for instance, if the larger graph represents an aggregation of objects and the smaller graph a specific object in the larger context. This intuitively leads to the formal definition of a subgraph.

Definition 2 (Subgraph). *Let* $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ *and* $g_2 = (V_2, E_2, \alpha_2, \beta_2)$ *be graphs;* g_2 *is a* subgraph *of* g_1*, written as* $g_2 \subseteq g_1$*, if*

- $V_2 \subseteq V_1$
- $E_2 = E_1 \cap (V_2 \times V_2)$
- $\alpha_1(v) = \alpha_2(v)$ *for all* $v \in V_2$
- $\beta_1(e) = \beta_2(e)$ *for all* $e \in E_2$

Conversely, graph g_1 is called a supergraph of g_2 if g_2 is a subgraph of g_1. Sometimes, the second condition of this definition is replaced by $E_2 \subseteq E_1 \cap (V_2 \times V_2)$, and a subgraph fulfilling the more stringent condition given above is called an induced subgraph. The notion of subgraph can be used to approach more complex problems such as the largest common part of several graphs, which will be discussed below.

5.3 Exact Graph Matching

In exact graph matching, the objective is to determine whether or not the structure and labels, or part of the structure, of two graphs are identical.

Definition 3 (Graph isomorphism). *Let* g_1 *and* g_2 *be graphs. A* graph isomorphism *between* g_1 *and* g_2 *is a bijective mapping* $f : V_1 \to V_2$ *such that*

- $\alpha_1(v) = \alpha_2(f(v))$ *for all* $v \in V_1$
- *for any edge* $e_1 = (u, v) \in E_1$ *there exists an edge* $e_2 = (f(u), f(v)) \in E_2$ *such that* $\beta_1(e_1) = \beta_2(e_2)$*, and for any edge* $e_2 = (u, v) \in E_2$ *there exists an edge* $e_1 = (f^{-1}(u), f^{-1}(v)) \in E_1$ *such that* $\beta_1(e_1) = \beta_2(e_2)$

Two graphs g_1 *and* g_2 *are called isomorphic if there exists a graph isomorphism between them.*

From this definition we conclude that isomorphic graphs are identical in terms of structure and labels. To establish an isomorphism one has to map each node from the first graph to a node of the second graph such that the edge structure is preserved and the node and edge labels are consistent.

The graph isomorphism problem is of considerable practical importance and also of theoretical interest due to its relationship to the concept of NP-completeness. Despite intensive research for over three decades [30,55,58] still no efficient (polynomial-bound) algorithm for graph isomorphism is known. Neither has the conjecture been proved that no such algorithm can exist. While it is easy to determine equality of patterns in case of feature vectors or strings, the same computation is much more complex for graphs. Because the nodes and edges of a graph cannot be ordered in general, unlike the components of a feature vector or the symbols of a string, the problem of graph equality (graph isomorphism) is computationally very demanding. The most straightforward approach to checking the isomorphism of two graphs is to traverse a search tree considering all possible node-to-node correspondences [61].

The expansion of tree branches is continued until the edge structure implied by the node mapping does not correspond in both graphs. If nodes and edges are additionally endowed with labels, matching nodes and edges must also be consistent in terms of their labels. Reaching a leaf node of the search tree is equivalent to successfully mapping all nodes without violating the structure and label constraints and is therefore equivalent to having found a graph isomorphism. In general, the computational complexity of this procedure is exponential in the number of nodes of either graph.

By imposing certain restrictions on the underlying graphs, however, it is possible to derive algorithms of polynomial-time complexity. For instance, Luks [45] described a polynomially bounded method for the isomorphism detection of graphs with bounded valence. For the special case of trivalent graph isomorphism, it was shown in [45] that algorithms with a computational complexity of $O(n^4)$ exist. Low-order polynomial-time methods [35, 36, 64] are also known for planar graphs. Quadratic-time algorithms [37, 38] have been reported for ordered graphs, in which the edges incident to a vertex are uniquely ordered. Further special graph classes, for which the isomorphism problem is solvable in polynomial time, are trees [1], interval graphs [9], permutation graphs [22], chordal $(6, 3)$ graphs [3], graphs with bounded genus [48], graphs with bounded treewidth [7], graphs with bounded eigenvalue multiplicity [2], and rooted directed path graphs [4].

Closely related to graph isomorphism is the problem to detect if a smaller graph is present in a larger graph. If graph isomorphism is regarded as a formal notion of graph equality, subgraph isomorphism can be seen as subgraph equality.

Definition 4 (Subgraph isomorphism). *Let* $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ *and* $g_2 = (V_2, E_2, \alpha_2, \beta_2)$ *be graphs. An injective function* $f : V_1 \to V_2$ *is called a subgraph isomorphism from* g_1 *to* g_2 *if there exists a subgraph* $g \subseteq g_2$ *such that* f *is a graph isomorphism between* g_1 *and* g*.*

A subgraph isomorphism exists from g_1 to g_2 if the larger graph g_2 can be turned into a graph that is isomorphic to the smaller graph g_1 by removing some nodes and edges. Subgraph isomorphism can also be determined with the procedure outlined above for graph isomorphism [61]. It is known that subgraph isomorphism belongs to the class of NP-complete problems.

5.4 Inexact Graph Matching

In graph representations of real-world patterns, it is often the case that graphs from the same class differ in terms of structure and labels. Hence, graph matching systems need to take structural errors into account. In this section several variants of realizing this goal are discussed.

5.4.1 Graph Edit Distance

Graph edit distance offers an intuitive way to integrate error-tolerance into the graph matching process and is applicable to virtually all types of graphs. Originally, edit distance has been developed for string matching [62] and a considerable amount of variants and extensions to the edit distance have been proposed for strings and graphs. The key idea is to model structural variation by edit operations reflecting modifications in structure, such as the removal of a single node or the modification of an attribute attached to an edge. A standard set of edit operations consists of a node insertion, node deletion, node substitution, edge insertion, edge deletion, and edge substitution operation.

Definition 5 (Edit path). *Let $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, E_2, \alpha_2, \beta_2)$ be graphs. Any bijective function $f : \hat{V}_1 \to \hat{V}_2$, where $\hat{V}_1 \subseteq V_1$ and $\hat{V}_2 \subseteq V_2$, is called an edit path from g_1 to g_2.*

We say that node $u \in \hat{V}_1$ is *substituted* by node $v \in \hat{V}_2$ if $f(u) = v$. If $\alpha_1(u) = \alpha_2(f(u))$ then the substitution is called an *identical* substitution. Otherwise, it is termed a *nonidentical* substitution. Furthermore, 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 $f(u_1) = v_1$ and $f(u_2) = v_2$ and there exist edges $(u_1, u_2) \in E_1$ and $(v_1, v_2) \in E_2$ then edge (u_1, u_2) is substituted by (v_1, v_2) under f. If $\beta_1((u_1, u_2)) = \beta_2((v_1, v_2))$ then the edge substitution is called an *identical* substitution. Otherwise, it is termed a *nonidentical* substitution. If there exists no edge $(u_1, u_2) \in E_1$, but an edge $(v_1, v_2) \in E_2$, then edge (v_1, v_2) is inserted. Similarly, if $(u_1, u_2) \in E_1$ exists but no edge (v_1, v_2), then (u_1, u_2) is deleted under f. If a node u is deleted from g_1, then any edge incident to u is deleted, too. Similarly, if a node u' is inserted in g_2, then any edge incident to u' is inserted, too. Obviously, any edit path 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. 5.1. For those graphs, we have the following:
$L_V = \{X, Y, Z\}$; $L_E = \{a, b, c\}$
$V_1 = \{1, 2, 3\}$; $V_2 = \{4, 5, 6, 7\}$
$E_1 = \{(1, 2), (1, 3), (2, 3)\}$; $E_2 = \{(4, 5), (4, 6), (4, 7), (5, 6), (5, 7)\}$
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: $(4, 5) \mapsto a$, $(4, 6) \mapsto c$, $(4, 7) \mapsto b$, $(5, 6) \mapsto c$, $(5, 7) \mapsto b$

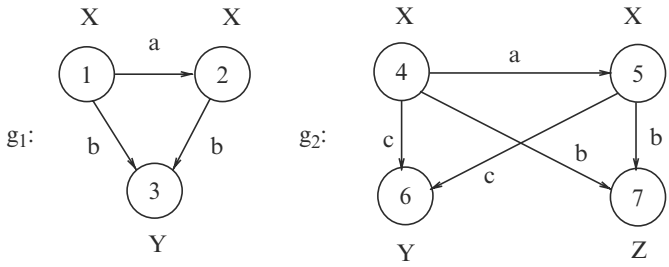

Fig. 5.1. An example of edit paths (see text)

Three examples of edit paths are the following:

- $f_1 : 1 \mapsto 4, 2 \mapsto 5, 3 \mapsto 7$ 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 $(5, 6)$ are inserted in g_2. There are, of course, many other paths from g_1 to g_2.

With substitutions, deletions, and insertions for both nodes and edges at our disposal, any graph can be transformed into any other graph by iteratively applying edit operations. Consequently, the concept of graph editing can be used to define a dissimilarity measure on graphs. To quantify how strongly an edit operation modifies the structure of a graph, it is common to use an edit cost function that assigns a cost value to each edit operation. An edit operation associated with a low cost is assumed to only slightly alter the graph under consideration, while an edit operation with a high cost is assumed to strongly modify the graph. To obtain a cost function on edit paths, we simply accumulate individual edit operation costs of the edit path.

Definition 6. *The* cost *of an edit path* $f : \hat{V}_1 \to \hat{V}_2$ *from a graph* $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ *to a graph* $g_2 = (V_2, E_2, \alpha_2, \beta_2)$ *is given by*

$$c(f) = \sum_{u \in \hat{V}_1} c_{ns}(u) + \sum_{u \in V_1 - \hat{V}_1} c_{nd}(u) + \sum_{u \in V_2 - \hat{V}_2} c_{ni}(u) + \sum_{e \in E_s} c_{es}(e) + \sum_{e \in E_d} c_{ed}(e) + \sum_{e \in E_i} c_{ei}(e),$$

where

- $c_{ns}(u)$ *is the cost of substituting node* $u \in \hat{V}_1$ *by* $f(u) \in \hat{V}_2$
- $c_{nd}(u)$ *is the cost of deleting node* $u \in V_1 - \hat{V}_1$ *from* g_1

- $c_{ni}(u)$ *is the cost of inserting node* $u \in V_2 - \hat{V}_2$ *in* g_2
- $c_{es}(e)$ *is the cost of substituting edge* e
- $c_{ed}(e)$ *is the cost of deleting edge* e
- $c_{ei}(e)$ *is the cost of inserting edge* e

and E_s, E_d, *and* E_i *are the sets of edges that are substituted, deleted, and inserted, respectively. All costs are nonnegative real numbers.*

Notice that the sets E_s, E_d, and E_i are implied by the mapping f. That is, if edge $e = (u_1, u_2) \in E_1$, $f(u_1) = v_1$, $f(u_2) = v_2$, and $(v_1, v_2) \notin E_2$, then $e \in E_d$. Similarly, if $(u_1, u_2) \notin E_1$, $f(u_1) = v_1$, $f(u_2) = v_2$, and $e = (v_1, v_2) \in E_2$, then $e \in E_i$. Likewise, if $e_1 = (u_1, u_2) \in E_1$, $f(u_1) = v_1$, $f(u_2) = v_2$, and $e_2 = (v_1, v_2) \in E_2$, then $e_1 \in E_s$. Because an edge is deleted (inserted) whenever one or both of its incident nodes are deleted (inserted), we furthermore observe that (1) $e \in E_d$ if $e \in (V_1 \times V_1) - (\hat{V}_1 \times \hat{V}_1)$ and (2) $e \in E_i$ if $e \in (V_2 \times V_2) - (\hat{V}_2 \times \hat{V}_2)$.

The problem of measuring the dissimilarity of two graphs is then equivalent to the problem of finding the edit path that models the structural difference of two graphs in the least costly way. Consequently, the graph edit distance of two graphs is defined by the minimum cost edit path from the first to the second graph.

Definition 7 (Graph edit distance). *Let* $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ *and* $g_2 = (V_2, E_2, \alpha_2, \beta_2)$ *be graphs and let* $c(f)$ *denote the cost of edit path* f. *The edit distance of* g_1 *and* g_2 *can be defined by*

$$d(g_1, g_2) = \min_{\text{all edit paths } f \text{ from } g_1 \text{ to } g_2} c(f)$$

If the two graphs under consideration are very similar in terms of structure and labels, it can be assumed that only minor edit operations are required to transform the first into the second graph, which results in a low-cost optimal edit path. In this case, the resulting edit distance will be small. Conversely, if the two graphs differ significantly, every edit path will necessarily include strong modifications and hence result in high costs.

In the following it is assumed, for the purpose of simplicity, that the costs $c_{nd}(x)$, $c_{ni}(x)$, and $c_{ns}(x)$ do not 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. Hence, the notation $c_{nd}(x) = c_{nd}$, $c_{ni}(x) = c_{ni}, \ldots, c_{es}(e) = c_{es}$ will be used and a cost function is given by the 6-tuple $C = (c_{nd}, c_{ni}, c_{ns}, c_{ed}, c_{ei}, c_{es})$. Unless otherwise stated, 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 than zero.

Example 2. Consider the uniform cost function $C = (c_{nd}, c_{ni}, c_{ns}, c_{ed}, c_{ei}, c_{es}) = (1, 1, 1, 1, 1, 1)$. Then the edit path f_1 given in Example 1 has cost $c(f_1) = 4$

(one node label substitution, one node insertion, and two edge insertions). It can be easily verified that there is no other edit path from g_1 to g_2 that has a smaller cost under C. For example, $c(f_2) = 5$ (two edge label substitutions, one node insertion, and two edge insertions) and $c(f_3) = 9$ (one node and two edge deletions, two node and four edge insertions). However, if we change the cost function and consider $C' = (1, 1, 3, 1, 1, 1)$ then $c(f_1) = 6$, $c(f_2)$=5, and $c(f_3) = 9$. Thus, f_1 is no longer optimal under C' and it can be easily verified that f_2 has in fact the smallest cost among all possible paths from g_1 to g_2. If we consider a third cost function $C'' = (1, 1, 7, 1, 1, 7)$ then $c(f_1) = 10$, $c(f_2) = 17$, and $c(f_3) = 9$. Under this cost function, f_3 is optimal.

If a cost function satisfies the conditions of positive definiteness and symmetry as well as the triangle inequality at the level of single edit operations, the resulting edit distance is known to be a metric [10]. This fact legitimates the use of the term distance in graph edit distance.

In practical applications, graph edit distance is usually accomplished by means of heuristic A*-based tree search procedures [10]. Because of the exponential nature of the problem, such procedures are normally limited to rather small graphs. Recently, however, several methods to speed up edit distance computation have been proposed. In [8, 59] the authors proposed to optimize local rather than global criteria, which results in a suboptimal procedure. Other suboptimal techniques were introduced in [51, 56]. In Justice and Hero [42] a linear programming method for computing the edit distance of graphs with unlabeled edges is proposed. This method can be used to derive lower and upper edit distance bounds in polynomial time.

5.4.2 Graph Distance Functions Based on mcs

The definition of subgraph isomorphism naturally leads us to the formal definition of the largest common part of two graphs.

Definition 8 (Maximum common subgraph). *Let $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, E_2, \alpha_2, \beta_2)$ be graphs. A common subgraph of g_1 and g_2, $cs(g_1, g_2)$, is a graph $g = (V, E, \alpha, \beta)$ such that there exist subgraph isomorphisms from g to g_1 and from g to g_2. We call g a maximum common subgraph of g_1 and g_2, $mcs(g_1, g_2)$, if there exists no other common subgraph of g_1 and g_2 that has more nodes than g.*

A maximum common subgraph of two graphs represents the maximal part of both graphs that is identical in terms of structure and labels. Note that, in general, the maximum common subgraph is not uniquely defined, that is, there may be more than one common subgraph with a maximal number of nodes. A standard approach to computing maximum common subgraphs is based on solving the maximum clique problem in an association graph [44,46]. The association graph of two graphs represents the whole set of possible node-to-node mappings that preserve the edge structure of both graphs. Finding a

maximum clique in the association graph, that is, a fully connected maximal subgraph, is equivalent to finding a maximum common subgraph. In Bunke et al. [16] the reader can find a comparison of algorithms for maximum common subgraph computation on randomly connected graphs.

Graph dissimilarity measures can be derived from the maximum common subgraph of two graphs. Intuitively speaking, the larger a maximum common subgraph of two graphs is, the more similar are the two graphs. This observation leads to graph dissimilarity measures that are able to cope with structural errors. Bunke and Shearer [12] have introduced such a distance measure:

$$d_{MCS}(g_1, g_2) \;=\; 1 - \frac{|mcs(g_1, g_2)|}{\max\{|g_1|, |g_2|\}} \tag{5.1}$$

where $|\ldots|$ indicates the size of a graph (usually taken to be the number of nodes). Note that, whereas the maximum common subgraph of two graphs is not uniquely defined, the d_{MCS} distance is. If two graphs are isomorphic, their d_{MCS} distance is 0; if two graphs have no part in common, their d_{MCS} distance is 1. The d_{MCS} distance accounts for a certain amount of tolerance towards errors, as two graphs need not be completely identical for a successful match. However, a small d_{MCS} distance, and hence a high graph similarity, can only be obtained if large portions of both graphs are isomorphic. It has been shown that d_{MCS} is a metric and produces a value in $[0, 1]$.

A second distance measure which has been proposed by Wallis et al. [63], based on the idea of graph union, is

$$d_{WGU}(g_1, g_2) \;=\; 1 - \frac{|mcs(g_1, g_2)|}{|g_1| + |g_2| - |mcs(g_1, g_2)|}$$

By "graph union" it is meant that the denominator represents the size of the union of the two graphs in the set-theoretic sense. This distance measure behaves similarly to d_{MCS}. The motivation of using graph union in the denominator is to allow for changes in the smaller graph to exert some influence over the distance measure, which does not happen with d_{MCS}. This measure was also demonstrated to be a metric and creates distance values in $[0, 1]$.

A similar distance measure [11] which is not normalized to the interval $[0, 1]$ is

$$d_{UGU}(g_1, g_2) \;=\; |g_1| + |g_2| - 2 \cdot |mcs(g_1, g_2)|$$

Fernandez and Valiente [27] have proposed a distance measure based on both the maximum common subgraph and the minimum common supergraph

$$d_{MMCS}(g_1, g_2) \;=\; |MCS(g_1, g_2)| - |mcs(g_1, g_2)|$$

where $MCS(g_1, g_2)$ is the minimum common supergraph of graphs g_1 and g_2, which is the complimentary idea of minimum common subgraph.

Definition 9 (Minimum common supergraph). *Let $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, E_2, \alpha_2, \beta_2)$ be graphs. A common supergraph of g_1 and g_2,*

$CS(g_1, g_2)$, is a graph $g = (V, E, \alpha, \beta)$ such that there exist subgraph isomorphisms from g_1 to g and from g_2 to g. We call g a minimum common supergraph of g_1 and g_2, $MCS(g_1, g_2)$, if there exists no other common supergraph of g_1 and g_2 that has less nodes than g.

The concept that drives the distance measure above is that the maximum common subgraph provides a "lower bound" on the similarity of two graphs, while the minimum supergraph is an "upper bound." If two graphs are identical, then both their maximum common subgraph and minimum common supergraph are the same as the original graphs and $|g_1| = |g_2| = |MCS(g_1, g_2)| = |mcs(g_1, g_2)|$, which leads to $d_{MMCS}(g_1, g_2) = 0$. As the graphs become more dissimilar, the size of the maximum common subgraph decreases, while the size of the minimum supergraph increases. This in turn leads to increasing values of $d_{MMCS}(g_1, g_2)$. For two graphs with no maximum common subgraph, the distance will become $|MCS(g_1, g_2)| = |g_1| + |g_2|$. The distance $d_{MMCS}(g_1, g_2)$ has also been shown to be a metric, but it does not produce values normalized to the interval $[0, 1]$, unlike d_{MCS} or d_{WGU}. We can also create a version of this distance measure which is normalized to $[0, 1]$ as follows:

$$d_{MMCSN}(g_1, g_2) = 1 - \frac{|mcs(g_1, g_2)|}{|MCS(g_1, g_2)|}$$

Note that if the conditions holds that $|MCS(g_1, g_2)| = |g_1| + |g_2| - |mcs(g_1, g_2)|$, then d_{UGU} and d_{MMCS} are identical. The same is true for d_{WGU} and d_{MMCSN}.

5.4.3 Relaxation Approaches

We use a matching matrix M to indicate the compatibility of nodes in the two graphs being matched. If the ith row and jth column element M_{ij} is 1, then node i in graph g_1 is matched with node j in graph g_2; otherwise there is no match and $M_{ij} = 0$. Constraints can be imposed on M so that each row has exactly one 1 and no column has more than one 1. Such a representation and the algorithms applied to it for determining graph matching are straightforward; however, they can require generating all the permutations of possible node matchings over the matrix.

To improve time complexity, we can instead attempt to approximate the optimal solution by finding good suboptimal solutions. A method that is sometimes used to achieve this for graph matching problems is called *relaxation* (or more specifically, *discrete relaxation*). Put simply, discrete relaxation is a method of transforming a discrete representation (such as the matrix M used for graph matching) into a continuous representation. Thus, we can transform a discrete optimization problem into a continuous one. Compared to the typical state-space search approaches to graph matching, relaxation is a nonlinear optimization approach. Gold and Rangdarajan [32] applied relaxation to the

graph matching problem. They have posed the problem of attributed graph matching in terms of an optimization problem:

$$E = -\frac{1}{2}\sum_{a=1}^{|V_1|}\sum_{i=1}^{|V_2|}\sum_{b=1}^{|V_1|}\sum_{j=1}^{|V_2|} M_{ai}M_{bj}\sum_{r=1}^{R} C_{aibj}^{(2,r)} + \alpha\sum_{a=1}^{|V_1|}\sum_{i=1}^{|V_2|} M_{ai}\sum_{r=1}^{S} C_{ai}^{(1,s)}$$

Here M is the matching matrix as before, R is the number of edge types, S is the number of node types, α is a weighting factor, and the C's are compatibility measures between the edges of the two graphs. The goal is then to minimize the objective function given above. In [32] the authors use the graduated assignment algorithm to find an M which minimizes E.

Medasani et al. [47] gave a procedure based on fuzzy assignments and relaxation similar to the method just described. The objective function for this approach is

$$J(M,C) = \sum_{i=1}^{|V_1|+1}\sum_{j=1}^{|V_2|+1} M_{ij}^2 f(C_{ij}) + \eta\sum_{i=1}^{|V_1|+1}\sum_{j=1}^{|V_2|+1} M_{ij}(1-M_{ij})$$

where M is now a fuzzy membership matrix ($0 \leq M_{ij} \leq 1$) that relates the degree of match between nodes, C is a compatibility matrix between nodes (rather than edges as above), η is a control parameter, and

$$f(C_{ij}) = e^{-\beta C_{ij}}$$

The summations in the objective function are under the constraint that $(i,j) \neq (|V_1|+1, |V_2|+1)$; the extra nodes in the graphs are dummy nodes. The authors then go on to derive the necessary update equations for M and C in order to minimize $J(M,C)$ and propose an algorithm which updates these matrices in an alternating fashion.

5.4.4 Probabilistic Approaches

In this section we summarize the probabilistic approach proposed by Wilson and Hancock [66]. We attempt to match a data graph g_D and a stored model graph g_M, both being attributed graphs. In Wilson and Hancock [66] an attributed graph is defined to be one $g = (V, E, A)$, where A is a set of attributes associated with each node, $A = x_v^y,\ \forall v \in V$.

The attributes in the data graph are to be matched to those in the model graph, such that the matched nodes have the same or similar attributes. Edges may also have associated attributes, but they are not considered in this approach. Next, we have the concept of super-clique of a node. A *super-clique* [66] of a node i in graph $g = (V, E, A)$ is defined as $C_i = i \cup \{j|(j,i) \in E\}$. In other words, the super-clique of a node i is the set of nodes which contain i and all nodes connected to it by edges. The goal is then to match all super-cliques in the data graph with super-cliques in the model graph.

The set of all possible matches between super-clique C_i in the data graph g_D and super-cliques in the model graph g_M is called a dictionary and denoted Θ_i. To cope with size differences between the data and model super-cliques dummy (or null) nodes ϕ are allowed to be inserted into S_j so that both graphs have the same number of nodes. The function matching a node in C_i to a node in S is $f : V_D \rightarrow V_M \cup \phi$. The probability of matching errors (a node in g_D is matched to the wrong node in g_M) is denoted P_e and the probability of structural errors (a node in g_D is matched to a dummy node in g_M) is denoted P_ϕ. Given these definitions, some assumptions, and through application of Bayes' rule and other probability theoretic constructions, Wilson and Hancock arrive at a mathematical description for the probability of a super-clique matching between two graphs (denoted Γ_j for super-clique C_j):

$$P(\Gamma_j) \;=\; \frac{K_{C_j}}{|\Theta_j|} \sum_{S_j \in \Theta_j} \exp\{-(k_e H(\Gamma_j, S_i) + k_\phi[\psi(\Gamma_j, S_i) + \Psi(\Gamma_j)])\}$$

where

$$\begin{aligned} K_{C_j} &= [(1 - P_e)(1 - P_\phi)]^{|C_j|} \\ k_e &= \ln \frac{1 - P_e}{P_e} \\ k_\phi &= \frac{(1 - P_e)(1 - P_\phi)}{P_\phi} \end{aligned}$$

$H(\Gamma_j, S_i)$ is the Hamming distance between the super-clique of g_D under the mapping f and the super-clique of g_M, $\psi(\Gamma_j, S_i) = |C_j| - |S_i|$ (i.e., the number of null nodes inserted into S_i), and $\Psi(\Gamma_j)$ is the number of nodes in C_j which are mapped onto null nodes in S_i. The deviation of $P(\Gamma_j)$ is beyond the scope of this chapter, but the equation contains three parts which are fairly straightforward. The part associated with K_{C_j} is the probability of no error occurring. The part associated with k_e is concerned with the probability of matching error occurring. Finally, the part associated with k_ϕ deals with the probability of structural errors occurring. For an in-depth derivation of these equations, the reader is referred to [66].

The authors then go on to derive rules which can be applied to update the matching function f under three different methods (null-labeling, constraint filtering, and graph edit operations). The methods use update rules of the form

$$f(u) \;=\; \arg\max_{v \in V_M} \frac{P(u, v | x_u^D, x_v^M)}{P(u, v)} \sum_{j \in C_u} P(\Gamma_j)$$

Here $P(u, v)$ indicates the prior probability that node u in g_D corresponds to node v in g_M, while the other probability in the numerator is the conditional *a posteriori* probability, given the corresponding attributes related with the nodes.

An advantage of this framework is that it can be applied in many situations. For example, an extension of the work [65] deals with multiple graph matching through computations of fuzzy consistency matrices. Finch et al. [28] developed an energy function for graph matching based on the probabilistic framework of this section. A method using this approach for the fitness function in a genetic search for graph matching is described in [25]. A similar probabilistic framework for hierarchical graphs is given in [67]. Myers et al. [49] modified the approach described here to include graph edit distance; the new method achieves better complexity by removing the need to inset null nodes in the model graph.

5.4.5 Distance Preservation Approach

In [21] Chartrand et al. describe an approach for graph distance calculation based on preserving the distance between nodes. The idea comes from the fact that when two graphs are isomorphic, the distance (meaning in this context the number of edges traversed) between every pair of nodes are identical in both graphs. Given a graph $g = (V, E)$, the distance between two nodes $x, y \in V$, denoted $d_g(x, y)$, is defined as the minimum number of edges that need to be traversed when traveling from x to y [21]. Further, the ϕ-distance [21] between two graphs g_1 an g_2, denoted $d_\phi(g_1, g_2)$, is defined as

$$d_\phi(g_1, g_2) \;=\; \sum_{\forall x \forall y \in V_1} |d_{g_1}(x, y) - d_{g_2}(x, y)|$$

where ϕ is a one-to-one mapping (but not necessarily an isomorphism) between g_1 and g_2.

If ϕ is an isomorphism, then $d_\phi(g_1, g_2) = 0$; if g_1 and g_2 are not isomorphic, then $d_\phi(g_1, g_2) > 0$. This leads to a definition of distance between two graphs as

$$d(g_1, g_2) \;=\; \min_{\forall \phi} d_\phi(g_1, g_2)$$

In [21] the authors also go on to show that we can make some other, less expensive calculations if the graphs meet certain requirements. For example, if g_1 and g_2 are connected graphs with an equal number of nodes, then we can determine the lower bound on their distance by

$$d(g_1, g_2) \;\geq\; |td(g_1) - td(g_2)|$$

where

$$td(g) \;=\; \sum_{\forall u, v \in V} d(u, v)$$

or, in other words, the sum of distances between all pairs of nodes in graph. Further theoretical contributions related to this approach can be found in [21].

5.5 Theoretical Foundations of Graph Matching

The graph distance measure according to (5.1) is based on the maximum common subgraph of two graphs. Obviously, it can be regarded an alternative to graph edit distance. Indeed, it was recently shown that there is 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 [11]. In [11] the following cost function was considered:

$$\begin{aligned}
c_{ns}(x) &= \left\{ \begin{array}{l} 0, \text{ if } \alpha_1(x) = \alpha_2(f(x)) \\ \infty, \text{ otherwise} \end{array} \right\} \text{for any } x \in \hat{V}_1, \\
c_{nd}(x) &= 1 \text{ for any } x \in V_1 - \hat{V}_1, \\
c_{ni}(x) &= 1 \text{ for any } x \in V_2 - \hat{V}_2, \\
c_{es}(e) &= \left\{ \begin{array}{l} 0, \text{ if } \beta_1((x,y)) = \beta_2((f(x), f(y))) \\ \infty, \text{ otherwise} \end{array} \right\} \\
&\quad \text{for any } e = (x,y) \in \hat{V}_1 \times \hat{V}_1, \\
c_{ed}(e) &= 0 \text{ for any } e = (x,y) \in (V_1 \times V_1) - (\hat{V}_1 \times \hat{V}_1), \\
c_{ei}(e) &= 0 \text{ for any } e = (x,y) \in (V_2 \times V_2) - (\hat{V}_2 \times \hat{V}_2).
\end{aligned} \tag{5.2}$$

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. Intuitively speaking, it is assumed that the cost of a node deletion (insertion) includes the cost of deleting (inserting) the incident edges. As for any two graphs $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, E_2, \alpha_2, \beta_2)$ there is always an edit path 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 edit path. Thus, we may think of edit operations with infinity cost as nonadmissible. In other words, under the given cost function we can restrict our attention on edit paths involving only insertions, deletions, and identical node and edge substitutions, but no nonidentical substitutions. For example, for the edit path 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 [11] 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):

$$d(g_1, g_2) = |g_1| + |g_2| - 2|g| \tag{5.3}$$

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 [11] that the mapping $f : \hat{V}_1 \rightarrow \hat{V}_2$, defining an optimal edit path according to Def. 7, 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 (5.2). Conversely, any algorithm that computes the maximum common subgraph of two graphs can be used for graph edit distance computation under cost function (5.2), using (5.3). A similar relation between string edit distance and longest common subsequence has been known for long [60].

The results derived in [11] were recently shown to hold not only for the cost function given in (5.2), but for a whole class consisting of infinitely many cost functions. In [13] cost functions C with $c_{ns} = c_{es} = 0$ for identical substitutions and

$$c_{nd} + c_{ni} < c_{ns} \text{ and } c_{nd} + c_{ni} < c_{es} \tag{5.4}$$

are considered. (Note that (5.2) 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. 7. Intuitively speaking, the conditions in (5.4) 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 cannot 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 do not 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 edit path f_3 in Example 1. It is optimal under the cost function $C'' = (1, 1, 7, 1, 1, 7)$ as explained in Example 2. 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 (5.4).

The equivalence of maximum common subgraph and graph edit distance computation shown in [13] is based on the assumption $c_{ei}(e) = c_{ed}(e) = 0$ for any edge e from $(V_1 \times V_1) - (\hat{V}_1 \times \hat{V}_1)$ and $(V_2 \times V_2) - (\hat{V}_2 \times \hat{V}_2)$, respectively, see (5.2). Thus, no individual costs for the deletion of edges from $(V_1 \times V_1) - (\hat{V}_1 \times \hat{V}_1)$ and no individual costs for the insertion of edges in $(V_2 \times V_2) - (\hat{V}_2 \times \hat{V}_2)$ 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 [13] yields additional insight on the measure $d_{MCS}(g_1, g_2)$ of (5.1). Although no *explicit* costs of graph edit operations are needed to compute $d_{MCS}(g_1, g_2)$, there are, nevertheless, costs involved in an *implicit* manner, because the quantity $|mcs(g_1, g_2)|$ in (5.1) is equivalent to the graph edit distance $d(g_1, g_2)$ in the sense of (5.3), assuming a cost function satisfying (5.4). 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 arbitrary cost function belonging to the class studied in [13]. From this point of view, the measure defined in (5.1) may be regarded an advantage over conventional graph edit distance computation because it is robust against changing the costs of the underlying graph edit operations over a fairly wide range.

Another important result shown in [13] is the existence of classes of cost functions that always result in the same optimal mapping $f : \hat{V}_1 \to \hat{V}_2$ 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 edit path 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 edit paths would differ by a factor α. In [13] it was shown that any optimal edit path 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

$$(c_{ni} + c_{nd})/c_{ns} = (c'_{ni} + c'_{nd})/c'_{ns} \qquad \text{and} \tag{5.5}$$

$$c_{es}/c_{ns} = c'_{es}/c'_{ns} \tag{5.6}$$

for cost functions C and C' are satisfied then any edit path 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 the values $c(f)$ obtained under two different cost functions that is similar to (5.3). 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 (5.5) and (5.6) are satisfied. The existence of similar classes of cost functions for string edit distance has been discovered recently Rice et al. [57].

As discussed above, maximum common subgraph computation is a special case of graph edit distance under a particular class of cost functions. It was furthermore shown in [13] that also graph isomorphism and subgraph isomorphism are special cases of edit paths. If we define $c_{nd} = c_{ni} = c_{ns} = c_{ed} = c_{ei} = c_{es} = \infty$ then an edit path f between g_1 and g_2 with $c(f) < \infty$ exists if

and only if there exists a graph isomorphism between g_1 and g_2. Clearly, any such graph isomorphism f is optimal and $c(f) = 0$. Similarly, if

$$
\begin{aligned}
&c_{nd} = c_{ns} = \infty,\\
&0 \leq c_{ni} < \infty,\\
&c_{es}(e) = c_{ei}(e) = c_{ed}(e) = \infty \text{ if } e \in \hat{V}_1 \times \hat{V}_1,\\
&c_{ed}(e) = 0 \text{ if } e \in (V_1 \times V_1) - (\hat{V}_1 \times \hat{V}_1),\\
&c_{ei}(e) = 0 \text{ if } e \in (V_2 \times V_2) - (\hat{V}_2 \times \hat{V}_2),
\end{aligned}
$$

then an optimal edit path f with $c(f) < \infty$ between g_1 and g_2 exists if and only if there exists a subgraph isomorphism from g_1 to g_2. Any optimal edit path f is in fact a subgraph isomorphism and $c(f) = (|g_2| - |g_1|)c_{ni}$.

5.6 Some Recent Developments

In this section we discuss some of the recent developments, in particular, automatic learning of graph edit distance, median graph, and weighted mean of two graphs.

5.6.1 Learning Edit Costs

One of the major difficulties in the application of edit distance in graph matching is the definition of adequate edit costs. The edit costs essentially govern how the structural matching is performed. For some graph representations, it may be crucial whether a node is missing or not, while for other representations, the connecting edges are more important than the nodes. The question of how to define edit costs can therefore only be addressed in the context of an application-specific graph representation.

In the case of labels from n-dimensional space of real numbers, a simple edit cost model that has been used often is based on the distance of labels. The idea is to assign edit costs to substitutions that are proportional to the Euclidean distance of the two labels. Substituting an edge by another edge with the same label therefore does not involve any costs. For nonidentical labels, the further the two labels differ from each other, the higher will be the corresponding substitution cost. Insertions and deletions are often assigned constant costs in this model. The advantage of this simple model is that only a few parameters are involved and the edit costs are defined in a very intuitive way. However, it turns out that for some applications this model is not sufficiently flexible. For instance, it does not take into account that some label components may be more relevant than others. Also, the absolute values of the labels are not evaluated, but only the distance of labels, which means that all regions of the label space are equally weighted in terms of edit costs.

Recently, automatic approaches [50, 52] have been proposed to learn the edit costs. In [50] an approach based on self-organizing maps (SOM) is proposed. SOMs [43] are two-layer neural networks consisting of an input layer

and a competitive layer. The role of the input layer is to forward input patterns to the competitive layer. The competitive neurons are arranged in a homogeneous grid structure such that every neuron is connected to its neighbors. The idea is that the competitive layer reflects the space of input elements, that is, every competitive neuron corresponds to an element of the input space. For pattern representations in terms of feature vectors, this is usually accomplished by assigning weight vectors of the same dimension as the input space to neurons. Upon feeding an input pattern into the network, neurons of the competitive layer compete for the position of the input element. To this end, the weight of the closest competitive neurons and their neighbors are adapted so as to shift them towards the position of the input element. The longer this procedure is carried out, the more the competitive neurons tend to migrate to areas where many input elements are present. That is, the competitive layer can be regarded as a model of the pattern space, where the neuronal density reflects the pattern density. This unsupervised learning procedure is called self-organization as it does not rely on predefined classes of input elements.

SOMs can be used to model the distribution of edit operations. The idea is to use a SOM to represent the label space. A sample set of edit operations is derived from pairs of graphs from the same class in a manner that is equivalent to the probabilistic model. The self-organization process turns the initially regular grid of the competitive layer into a deformed grid. From the deformed grid, we obtain a distance measure for substitution costs and a density estimation for insertion and deletion costs. In the case of substitutions, the SOM is trained with pairs of labels, where one label belongs to the source node (or edge) and one to the target node (or edge). The competitive layer of the SOM is then adapted so as to draw the two regions corresponding to the source and the target label closer to each other. The edit cost of node and edge substitutions is then defined proportional to the distance in the trained SOM. That is, instead of measuring label distance by the Euclidean distance, we measure the deformed distance in the corresponding SOM. In the case of insertions and deletions, the SOM is adapted so as to draw neurons closer to the inserted or deleted label. The edit cost of insertions and deletions is then defined according to the competitive neural density at the respective position. That is, the more neurons at a certain position in the competitive layer are, the lower is the respective insertion or deletion cost. The self-organizing training procedure for substitutions, insertions, and deletions hence results in lower costs for those pairs of graphs that are in the training set and belong to the same class.

5.6.2 Median Graph

The concept of median graph does not give us an indication of graph similarity, but is useful in summarizing a group of graphs. This is among others needed in applications such as clustering, where we have to represent a group of graphs by some representative exemplar graph.

Given a set of graphs $S = \{g_1, g_2, \ldots, g_n\}$ defined over labels from L_V and L_E and a distance function $d()$ that measures the dissimilarity of two graphs, the concept of median graph is given as follows [39]:

Definition 10. *Let U be the set of all graphs that can be constructed using labels from L_V and L_E. The generalized median graph $\overline{g}$ and the set median graph $\hat{g}$ of $S \subset U$ are defined by*

$$\overline{g} \;=\; \arg\min_{g \in U} \sum_{i=1}^{n} d(g, g_i)$$

and

$$\hat{g} \;=\; \arg\min_{g \in S} \sum_{i=1}^{n} d(g, g_i)$$

respectively.

Both the generalized median and the set median graph minimize the sum of distances to all input graphs and the only difference lies in the graph space where the median is searched for. The generalized median is the more general concept and therefore usually a better representation of the given patterns than the set median. Notice that $\overline{g}$ is usually not a member of S. In general several generalized median graphs and several set median graphs may exist. However, this is usually not a drawback in practice since any such graph may serve equally well as a representative of the given set.

Conceptually, searching for the set median graph of n input graphs is an easy task since it suffices to compute $\frac{1}{2}n(n-1)$ pairwise graph distances. Due to the high expense of graph distance computation, however, it is often desired to determine the set median graph more efficiently than under the naive approach. Some suggested methods for this purpose can be found in [41].

Determining the generalized median graph is computationally more complex. On the one hand, the computation time is clearly exponential in the size of the input graphs. On the other hand, it is also exponential in terms of the number of input graphs. The reason for this behavior is that already for the special case of strings, the required time is exponential in the number of input strings [34]. As a consequence, we are generally forced to resort to approximate solutions that can be found in reasonable time. In [39] a genetic solution is proposed for this purpose. A comparison of the genetic algorithm against combinatorial search is described in [14].

When approximative approaches are involved, the question of accuracy of the approximative generalized median graph $\tilde{g}$ arises. In [40] a lower bound is proposed to answer this question. An approximate computation method gives us a solution $\tilde{g}$ such that

$$\mathrm{SOD}(\tilde{g}) \;=\; \sum_{p \in S} d(\tilde{g}, p) \;\geq\; \sum_{p \in S} d(\overline{g}, p) \;=\; \mathrm{SOD}(\overline{g})$$

where SOD stands for sum of distances and $\overline{g}$ represents the (unknown) true generalized median graph. The quality of $\tilde{g}$ can be measured by the difference $\mathrm{SOD}(\tilde{g}) - \mathrm{SOD}(\overline{g})$. Since $\overline{g}$ and $\mathrm{SOD}(\overline{g})$ are unknown in general, we resort to a lower bound $\Gamma \leq \mathrm{SOD}(\overline{g})$ and measure the quality of $\tilde{g}$ by $\mathrm{SOD}(\tilde{g}) - \Gamma$. Note that the relationship

$$0 \;\leq\; \Gamma \;\leq\; SOD(\overline{g}) \;\leq\; SOD(\tilde{g})$$

holds. Obviously, $\Gamma = 0$ is a trivial, and also useless, lower bound. We thus require Γ to be as close to $\mathrm{SOD}(\overline{g})$ as possible.

The distance function $d(p, q)$ is assumed to be a metric. The generalized median graph $\overline{g}$ is characterized by

$$\begin{aligned}
&\text{minimize } \mathrm{SOD}(\overline{g}) \;=\; d(\overline{g}, g_1) + d(\overline{g}, g_2) + \cdots + d(\overline{g}, g_n) \text{ subject to} \\
&\forall i, j \in \{1, 2, \ldots, n\}, i \neq j, \; \begin{cases} d(\overline{g}, g_i) + d(\overline{g}, g_j) \;\geq\; d(g_i, g_j) \\ d(\overline{g}, g_i) + d(g_i, g_j) \;\geq\; d(\overline{g}, g_j) \\ d(\overline{g}, g_j) + d(g_i, g_j) \;\geq\; d(\overline{g}, g_i) \end{cases} \\
&\forall i \in \{1, 2, \ldots, n\}, \; d(\overline{g}, g_i) \geq 0
\end{aligned}$$

Note that the constraints except the last set of inequalities are derived from the triangular inequality of the metric $d(p, q)$. By defining n variables x_i, $i = 1, 2, \ldots, n$, we replace $d(\overline{g}, g_i)$ by x_i and obtain the linear program LP:

$$\begin{aligned}
&\text{minimize } x_1 + x_2 + \cdots + x_n \text{ subject to} \\
&\forall i, j \in \{1, 2, \ldots, n\}, i \neq j, \; \begin{cases} x_i + x_j \;\geq\; d(g_i, g_j) \\ x_i + d(g_i, g_j) \;\geq\; x_j \\ x_j + d(g_i, g_j) \;\geq\; x_i \end{cases} \\
&\forall i \in \{1, 2, \ldots, n\}, \; x_i \geq 0
\end{aligned}$$

If we denote the solution of LP by Γ, then the true generalized median $\overline{g}$ satisfies $\Gamma \leq \mathrm{SOD}(\overline{g})$, i.e., Γ is a lower bound for $\mathrm{SOD}(\overline{g})$.

5.6.3 Weighted Mean of Two Graphs

If we consider two points x and y in the k-dimensional real space $\Re^k$, their weighted mean can be defined as a point z such that

$$z \;=\; (1 - \gamma) \cdot x + \gamma \cdot y, \quad 0 \leq \gamma \leq 1$$

Clearly, if $\gamma = \frac{1}{2}$ then z is the (normal) mean of x and y. If z is as defined above, then $z - x = \gamma \cdot (y - x)$ and $y - z = (1 - \gamma) \cdot (y - x)$. In other words, z is a point on the line segment in n dimensions that connects x and y, and the distance between z and both x and y is controlled via the parameter γ.

Conceptually, the same idea can be easily transformed into other domains, such as strings [17] and graphs [15]. According to Bunke and Günter [15] the weighted mean of two graphs g_1 and g_2 is a graph g such that

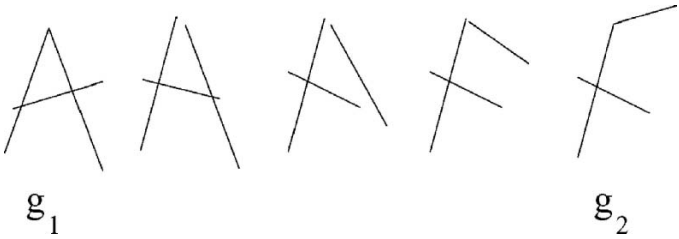

Fig. 5.2. Example of a series of weighted means of two graphs

$$d(g, g_1) \;=\; \gamma \cdot d(g_1, g_2)$$

and

$$d(g, g_2) \;=\; (1 - \gamma) \cdot d(g_1, g_2)$$

where $0 < \gamma < 1$.

An algorithm for finding the weighted mean of two graphs is given in [15]. It is an extension of the method for weighted mean of strings [17] and involves finding a subset of edit operations (given the lowest edit cost between two graphs) for the given γ in order to determine the weighted mean graph. Figure 5.2 shows an example of a series of weighted means of two graphs. In some sense the concept of weighted mean graph gives us a tool of "morphing" one graph into another graph.

If $\gamma = 0.5$, then we obtain the mean of two graphs g_1 and g_2 [33] and

$$d(g_1, g) \;=\; d(g, g_2), \quad d(g_1, g_2) \;=\; d(g, g_1) + d(g, g_2)$$

holds. In other words, the mean graph g is equidistant from both g_1 and g_2. Clearly, the mean will depend on the distance function chosen. There may be more than one graph satisfying these conditions; it is also possible that no (exact) mean graph exists for a given pair of graphs.

5.7 Conclusions

Graph matching has successfully been applied to various problems in image processing and understanding. In the case of exact graph matching, the graph extraction process is assumed to be structurally flawless, i.e., the conversion of image data of a single class into graphs always results in identical structures or substructures. Otherwise, graph isomorphism or subgraph isomorphism detection are rather unsuitable, which seriously restricts the applicability of graph isomorphism algorithms. The main advantages of isomorphism algorithms are their mathematically stringent formulation and the existence of well-known procedures to derive optimal solutions.

Error-tolerant methods, sometimes also referred to as inexact or error-correcting methods, are characterized by their ability to cope with errors, or

noncorresponding parts, in structure and labels of graphs. Hence, in order for two graphs to be positively matched, they need not be identical at all, but only similar. The notion of graph similarity depends on the error-tolerant matching method that is to be applied.

In this chapter we have given an overview of both exact and inexact graph matching. The emphasis has been the fundamental concepts and the recent developments concerned with the automatic learning of edit costs, median graph, and weighted mean of two graphs. Particularly, the concept of inexact graph matching provides us a means of measuring the similarity of graphs and thus lays the foundation of using the versatile and flexible tool of graphs in case-based reasoning in dealing with images.

References

1. Aho AV *et al.* (1974) The design and analysis of computer algorithms. Reading: Addison-Wesley
2. Babai L *et al.* (1982) Isomorphism of graphs with bounded eigenvalue multiplicity. Proc. of 14th ACM Symposium on Theory of Computing, pp. 310–324
3. Babel L (1995) Isomorphism of chordal $(6, 3)$ graphs. Computing 54:303–316
4. Babel L *et al.* (1996) The isomorphism problem for directed path graphs and for rooted directed path graphs. Journal of Algorithms 21:542–564
5. Balakrishnau VK (1997) Theory and Problems of Graph Theory. McGraw-Hill
6. Bischof WF, Caelli T (2001) Learning spatio-temporal relational structures. Applied Artificial Intelligence 15:707–722
7. Bodlaender HL (1990) Polynomial algorithms for graph isomorphism and chromatic index on partial k-tress. Journal of Algorithms 11:631–643
8. Boeres MC, Ribeiro CC, Bloch I (2004) A randomized heuristic for scene recognition by graph matching. In Ribeiro CC, Martins SL (Eds.), Proc. of 3rd Workshop on Efficient and Experimental Algorithms, LNCS 3059, pp. 100–113, Springer
9. Booth KS, Lueker GS (1979) A linear-time algorithm for deciding interval graph isomorphism. JACM 26:183–195
10. Bunke H, Allermann G (1983) Inexact graph matching for structural pattern recognition. Pattern Recognition Letters 1:245–253
11. Bunke H (1997) On a relation between graph edit distance and maximum common subgraph. Pattern Recognition Letters 18:689–694
12. Bunke H, Shearer K (1998) A graph distance metric based on the maximal common subgraph. Pattern Recognition Letters 19:255–259
13. Bunke H (1999) Error correcting graph matching: On the influence of the underlying cost function. IEEE Trans. on Pattern Analysis and Machine Intelligence 21:917–922
14. Bunke H, Münger A, Jiang X (1999) Combinatorial search versus genetic algorithms: a case study based on the generalized mean graph problem. Pattern Recognition Letters 20:1271–1277
15. Bunke H, Günter S (2001) Weighted mean of a pair of graphs. Computing 67:209–224

16. Bunke H, Foggia P, Guidobaldi C, Sansone C, Vento M (2002) A comparison of algorithms for maximum common subgraph on randomly conntected graphs. In Caelli T, Amin A, Duin R, Kamel M, de Ridder D (Eds.), Structural, Syntactic, and Statistical Pattern Recognition, pp. 123–132, Springer
17. Bunke H, Jiang X, Kandel A (2002) On the weighted mean of a pair of strings. Pattern Analysis and Applications 5:23–30
18. Bunke H, Caelli T, Eds. (2004) Special Issue on Graph Matching in Pattern Recognition and Machine Vision, in International Journal of Pattern Recognition and Artificial Intelligence 18(3)
19. Burke EK, MacCarthy B, Petrovic S, Qu R (2001) Case-based reasoning in course timetabling: An attributed graph approach. In Aha DW, Watson I (Eds.), Case-Based Reasoning Research and Development, LNAI 2080, pp. 90–103, Springer
20. Champin PA, Solnon C (2003) Measuring the similarity of labeled graphs. In Ashley KD, Bridge DG (Eds.), Case-Based Reasoning Research and Development, Proc. of 5th Int. Conf. on Case-Based Reasoning, LNCS 2689, pp. 80–95, Springer
21. Chartrand G, Kubicki G, Schultz M (1998) Graph similarity and distance in graphs. Aequationes Mathematicae 55:129–145
22. Colbourn GJ (1981) On testing isomorphism of permutation graphs. Networks 11:13–21
23. Conte D, Foggia P, Sansone C, Vento M (2004) Thirty years of graph matching in pattern recognition. International Journal of Pattern Recognition and Artificial Intelligence 18:265–298
24. Cook DJ, Holder LB, Eds. (2007) Mining Graph Data. Wiley Interscience
25. Cross ADJ, Wilson RC, Hancock ER (1997) Inexact graph matching using genetic search. Pattern Recognition 30:953–970
26. Dickinson SJ, Pelillo M, Zabih R, Eds. (2001) Special Section on Graph Algorithms and Computer Vision, in IEEE Trans. on Pattern Analysis and Machine Intelligence 23(10)
27. Fernandez ML, Valiente G (2001) A graph distance metric combining maximum common subgraph and minimum common supergraph. Pattern Recognition Letters 22:753–758
28. Finch AM, Wilson RC, Hancock ER (1998) An energy function and continuous edit process for graph matching. Neural Computation 10:1873–1894
29. Foggia P, Sansone C, Vento M, Eds. (2003) Special Issue on Graph-based Representations in Pattern Recognition, in Pattern Recognition Letters 24(8)
30. Gati G (1979) Further annotated bibliography on the isomorphism disease. Journal of Graph Theory 3:95–109
31. Gebhardt F, Voss A, Gräther W, Schmidt-Belz B (1997) Reasoning with Complex Cases. Kluwer
32. Gold S, Rangarajan A (1996) A graduated assignment algorithm for graph matching. IEEE Trans. on Pattern Analysis and Machine Intelligence 18: 377–388
33. Günter S, Bunke H (2002) Self-organizing map for clustering in the graph domain. Pattern Recognition Letters 23:401–417
34. de la Higuera C, Casacuberta F (2000) Topology of strings: Median string is NP-complete. Theoretical Computer Science 230:39–48
35. Hopcroft JE, Tarjan RE (1973) A $VlogV$ algorithm for isomorphism of triconnected planar graphs. Journal of Computer and System Sciences 7:323–331

36. Hopcroft JE, Wong JK (1974) Linear time algorithm for isomorphism of planar graphs. Proc. of 6th Annual ACM Symposium on Theory of Computing, pp. 172–184
37. Jiang X, Bunke H (1998) On the coding of ordered graphs. Computing 61:23-38
38. Jiang X, Bunke H (1999) Optimal quadratic-time isomorphism of ordered graphs. Pattern Recognition, 32:1273-1283
39. Jiang X, Münger A, Bunke H (2001) On median graphs: properties, algorithms, and applications. IEEE Trans. on Pattern Analysis and Machine Intelligence 23:1144–1151
40. Jiang X, Bunke H (2002) Optimal lower bound for generalized median problems in metric space. In Caelli T, Amin A, Duin E, Kamel M, de Ridder D (Eds.), Structural, Syntactic, and Statistical Pattern Recognition, pp. 143–151, Springer
41. Juan A, Vidal E (1998) Fast median search in metric spaces. In Amin A, Dori D (Eds.), Advances in Pattern Recognition, pp. 905–912, Springer
42. Justice D, Hero A (2006) A binary linear programming formulation of the graph edit distance. IEEE Trans. on Pattern Analysis and Machine Intelligence 28:1200–1214
43. Kohonen T (1995) Self-Organizing Maps. Springer
44. Levi, G (1972) A note on the derivation of maximal common subgraphs of two directed or undirected graphs. Calcolo 9:341–354
45. Luks EM (1982) Isomorphism of graphs of bounded valence can be tested in polynomial time. Journal of Computer and System Science 25:42–65
46. McGregor JJ (1982) Backtrack search algorithms and the maximal common subgraph problem. Software Practice and Experience 12:23–34
47. Medasani S, Krishnapuram R, Choi YS (2001) Graph matching by relaxation of fuzzy assignments. IEEE Trans. on Fuzzy Systems 9:173–182
48. Miller GL (1980) Isomorphism testing for graphs with bounded genus. Proc. of 12th ACM Symposium on Theory of Computing, pp. 225–235
49. Myers R, Wilson RC, Hancock ER (2000) Bayesian graph edit distance. IEEE Trans. on Pattern Analysis and Machine Intelligence 22:628–635
50. Neuhaus M, Bunke H (2005) Self-organizing maps for learning the edit costs in graph matching. IEEE Trans. on Systems, Man, and Cybernetics - Part B: Cybernetics 35:503–514
51. Neuhaus M, Riesen K, Bunke H (2006) Fast suboptimal algorithms for the computation of graph edit distance. Proc. Joint IAPR Workshops on Structural and Syntactic Pattern Recognition and Statistical Techniques in Pattern Recognition, Hong Kong, 163–172
52. Neuhaus M, Bunke H (2007) Automatic learning of cost functions for graph edit distance. Information Sciences 177:239–247
53. Perner P, Holt A, Richter M (2005) Image processing in case-based reasoning. The Knowledge Engineering Review 20:311–314
54. Perner P (2006) Case-base maintenance by conceptual clustering of graphs. Engineering Applications of Artificial Intelligence 19:381–393
55. Read RC, Corneil DG (1977) The graph isomorphism disease. Journal of Graph Theory 1:339–363
56. Riesen K, Neuhaus M, Bunke H (2007) Bipartite graph matching for computing the edit distance of graphs. Proc. of 6th Int. Workshop on Graph-based Representations, Alicante, Spain
57. Rice S, Bunke H, Nartker T (1997) Classes of cost functions for string matching. Algorithmica 18:271–280

58. Schöning U (1988) Graph isomorphism is in the low hierarchy. Journal of Computer and System Sciences 37:312–323
59. Sorlin S, Solnon S (2005) Reactive tabu search for measuring graph similarity. In Brun L, Vento M (Eds,), Proc. of 5th Int. Workshop on Graph-based Representations in Pattern Recognition, LNCS 3434, pp. 172–182. Springer
60. Stephen GA (1994) String Searching Algorithms. World Scientific, Publ. Co.
61. Ullman, JR (1976) An algorithm for subgraph isomorphism. Journal of the Association for Computing Machinery 23:31–42
62. Wagner RA, Fischer MJ (1974) The string-to-string correction problem. JACM 21:168–173
63. Wallis WD, Shoubridge P, Kraetzl M, Ray D (2001) Graph distances using graph union. Pattern Recognition Letters 22:701–704
64. Weinberg L (1966) A simple and efficient algorithm for determining isomorphism of planar triply connected graphs. IEEE Trans. on Circuit Theory 13:142–148.
65. Williams ML, Wilson RC, Hancock ER (1997) Multiple graph matching with Bayesian inference. Pattern Recognition Letters 18:1275–1281
66. Wilson RC, Hancock ER (1997) Structural matching by discrete relaxation. IEEE Trans. on Pattern Analysis and Machine Intelligence 19:634–647
67. Wilson RC, Hancock ER (1999) Graph matching with hierarchical discrete relaxation. Pattern Recognition Letters 20:1041–1052