Chapter 7

EXACT AND INEXACT GRAPH MATCHING: METHODOLOGY AND APPLICATIONS

Kaspar Riesen
Institute of Computer Science and Applied Mathematics, University of Bern
Neubrückstrasse 10, CH-3012 Bern, Switzerland
riesen@iam.unibe.ch

Xiaoyi Jiang
Department of Mathematics and Computer Science, University of Münster
Einsteinstrasse 62, D-48149 Münster, Germany
xjiang@math.uni-muenster.de

Horst Bunke
Institute of Computer Science and Applied Mathematics, University of Bern
Neubrückstrasse 10, CH-3012 Bern, Switzerland
bunke@iam.unibe.ch

Abstract Graphs provide us with a powerful and flexible representation formalism which can be employed in various fields of intelligent information processing. The process of evaluating the similarity of graphs is referred to as graph matching. Two approaches to this task exist, viz. exact and inexact graph matching. The former approach aims at finding a strict correspondence between two graphs to be matched, while the latter is able to cope with errors and measures the difference of two graphs in a broader sense. The present chapter reviews some fundamental concepts of both paradigms and shows two recent applications of graph matching in the fields of information retrieval and pattern recognition.

Keywords: Exact and Inexact Graph Matching, Graph Edit Distance, Information Retrieval by means of Graph Matching, Graph Embedding via Graph Matching

1. Introduction

After many years of research, the fields of pattern recognition, machine learning and data mining have reached a high level of maturity [4]. Powerful methods for classification, clustering, information retrieval, and other tasks have become available. However, the vast majority of these approaches rely on object representations given in terms of feature vectors. Such object representations have a number of useful properties. For instance, the dissimilarity, or distance, of two objects can be easily computed by means of the Euclidean distance. Moreover, a large number of well-established methods for data mining, information retrieval, and related tasks in intelligent information processing are available. Recently, however, a growing interest in graph-based object representation can be observed [16]. Graphs are powerful and universal data structures able to explicitly model networks of relationships between substructures of a given object. Thereby, the size as well as the complexity of a graph can be adopted to the size and complexity of a particular object (in contrast to vectorial approaches where the number of features has to be fixed beforehand).

Yet, after the initial enthusiasm induced by the "smartness" and flexibility of graph representations in the late seventies, a number of problems became evident. First, working with graphs is unequally more challenging than working with feature vectors, as even basic mathematic operations cannot be defined in a standard way, but must be provided depending on the specific application. Hence, almost none of the common methods for data mining, machine learning, or pattern recognition can be applied to graphs without significant modifications.

Second, graphs suffer from of their own flexibility. For instance, computing the distances of a pair of objects, which is an important task in many areas, is linear in the number of data items in the case where vectors are employed. The same task for graphs, however, is much more complex, since one cannot simply compare the sets of nodes and edges, which are generally unordered and of different size. More formally, when computing graph dissimilarity or similarity one has to identify common parts of the graphs by considering all of their subgraphs. Regarding that there are $O(2^n)$ subgraphs of a graph with n nodes, the inherent difficulty of graph comparisons becomes obvious.

Despite adverse mathematical and computational conditions in the graph domain, various procedures for evaluating proximity, i.e. similarity or dissimilarity, of graphs have been proposed in the literature [15]. The process of evaluating the similarity of two graphs is commonly referred to as *graph matching*. The overall aim of graph matching is to find a correspondence between the nodes and edges of two graphs that satisfies some, more or less, stringent constraints. That is, by means of the graph matching process similar substructures in one graph are mapped to similar substructures in the other graph. Based on this matching, a dissimilarity or similarity score can eventually be computed indicating the proximity of two graphs.

Graph matching has been the topic of numerous studies in computer science over the last decades. Roughly speaking, there are two categories of tasks in graph matching, viz. *exact matching* and *inexact matching*. In the former case, for a matching to be successful, it is required that a strict correspondence is found between the two graphs being matched, or at least among their subparts. In the latter approach this requirement is substantially relaxed, since also matchings between completely non-identical graphs are possible. That is, inexact matching algorithms are endowed with a certain tolerance to errors and noise, enabling them to detect similarities in a more general way than the exact matching approach. Therefore, inexact graph matching is also referred to as *error-tolerant graph matching*.

For an extensive review of graph matching methods and applications, the reader is referred to [15]. In this chapter, basic notations and definitions are introduced (Sect. 2) and an overview of standard techniques for exact as well as error-tolerant graph matching is given (Sect. 3 and 4). In Sect. 3, dissimilarity models derived from graph isomorphism, subgraph isomorphism, and maximum common subgraph are discussed for exact graph matching. In Sect. 4, inexact graph matching and in particular the paradigm of edit distance applied to graphs is discussed. Finally, two recent applications of graph matching are reviewed. First, in Sect. 5 an algorithmic framework for information retrieval based on graph matching is described. This approach is based on both exact and inexact graph matching procedures and aims at querying large database graphs. Secondly, a graph embedding procedure based on graph matching is reviewed in Sect. 6. This framework aims at an explicit embedding of graphs in real vector spaces, which establishes access to the rich repository of algorithmic tools for classification, clustering, regression, and other tasks, originally developed for vectorial representations.

2. Basic Notations

Various definitions for graphs can be found in the literature, depending upon the considered application. It turns out that the definition given below is sufficiently flexible for a large variety of tasks.

Definition 7.1 (Graph). *Let L_V and L_E be a finite or infinite label alphabet for nodes and edges, respectively. A graph g is a four-tuple $g = (V, E, \mu, \nu)$, where*

- *V is the finite set of nodes,*
- *$E \subseteq V \times V$ is the set of edges,*
- *$\mu : V \to L_V$ is the node labeling function, and*

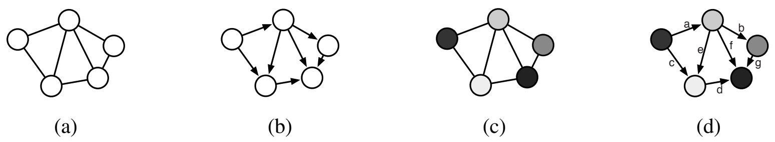


Figure 7.1. Different kinds of graphs: (a) undirected and unlabeled, (b) directed and unlabeled, (c) undirected with labeled nodes (different shades of gray refer to different labels), (d) directed with labeled nodes and edges.

- *$\nu : E \to L_E$ is the edge labeling function.*

The number of nodes of a graph g is denoted by $|g|$, while $\mathcal{G}$ represents the set of all graphs over the label alphabets L_V and L_E.

Definition 7.1 allows us to handle arbitrarily structured graphs with unconstrained labeling functions. For example, the labels for both nodes and edges can be given by the set of integers $L = \{1, 2, 3, \ldots\}$, the vector space $L = \mathbb{R}^n$, or a set of symbolic labels $L = \{\alpha, \beta, \gamma, \ldots\}$. Given that the nodes and/or the edges are labeled, the graphs are referred to as *labeled graphs*. *Unlabeled graphs* are obtained as a special case by assigning the same label ε to all nodes and edges, i.e. $L_V = L_E = \{\varepsilon\}$.

Edges are given by pairs of nodes (u, v), where $u \in V$ denotes the source node and $v \in V$ the target node of a directed edge. Commonly, the two nodes u and v connected by an edge (u, v) are referred to as *adjacent*. A graph is termed *complete* if all pairs of nodes are adjacent. *Directed graphs* directly correspond to the definition above. In addition, the class of *undirected graphs* can be modeled by inserting a reverse edge $(v, u) \in E$ for each edge $(u, v) \in E$ with identical labels, i.e. $\nu(u, v) = \nu(v, u)$. In Fig. 7.1 some graphs (directed/undirected, labeled/unlabeled) are shown.

Definition 7.2 (Subgraph). *Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ be graphs. Graph g_1 is a subgraph of g_2, denoted by $g_1 \subseteq g_2$, if*

(1) $V_1 \subseteq V_2$,

(2) $E_1 \subseteq E_2$,

(3) $\mu_1(u) = \mu_2(u)$ *for all* $u \in V_1$, *and*

(4) $\nu_1(e) = \nu_2(e)$ *for all* $e \in E_1$.

By replacing condition *(2)* in Definition 7.2 by the more stringent condition

(2') $E_1 = E_2 \cap V_1 \times V_1$,

g_1 becomes an *induced subgraph* of g_2. If g_2 is a subgraph of g_1, graph g_1 is called a *supergraph* of g_2.

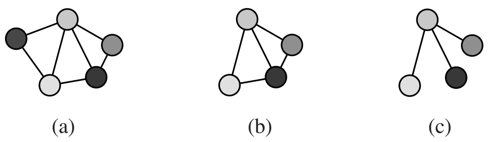


Figure 7.2. Graph (b) is an induced subgraph of (a), and graph (c) is a non-induced subgraph of (a).

Obviously, a subgraph g_1 is obtained from a graph g_2 by removing some nodes and their incident, as well as possibly some additional, edges from g_2. For g_1 to be an induced subgraph of g_2, some nodes and only their incident edges are removed from g_2, i.e. no additional edge removal is allowed. Fig. 7.2(b) and 7.2(c) show an induced and a non-induced subgraph of the graph in Fig. 7.2(a), respectively.

3. Exact Graph Matching

The aim in exact graph matching is to determine whether two graphs, or at least part of them, are identical in terms of structure and labels. A common approach to describe the structure of a graph is to define the *adjacency matrix* $\mathbf{A} = (a_{ij})_{n \times n}$ of graph $g = (V, E, \mu, \nu)$ ($|g| = n$). In this matrix the entry a_{ij} is equal to 1 if there is an edge $(v_i, v_j) \in E$ connecting the i-th node $v_i \in V$ with the $j - th$ node $v_j \in V$, and 0 otherwise.

Generally, for the nodes (and also the edges) of a graph there is no unique canonical order. Thus, for a single graph with n nodes, $n!$ different adjacency matrices exist, since there are $n!$ possibilities to order the nodes of g. Consequently, for checking two graphs for structural identity, we cannot simply compare their adjacency matrices. The identity of two graphs g_1 and g_2 is commonly established by defining a function, termed graph isomorphism, that maps g_1 to g_2.

Definition 7.3 (Graph Isomorphism). *Let us consider two graphs denoted by $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ respectively. A graph isomorphism is a bijective function $f : V_1 \to V_2$ satisfying*

(1) $\mu_1(u) = \mu_2(f(u))$ *for all nodes* $u \in V_1$

(2) for each edge $e_1 = (u, v) \in E_1$, *there exists an edge*

$$e_2 = (f(u), f(v)) \in E_2$$

such that $\nu_1(e_1) = \nu_2(e_2)$

(3) for each edge $e_2 = (u, v) \in E_2$, *there exists an edge*

$$e_1 = (f^{-1}(u), f^{-1}(v)) \in E_1$$

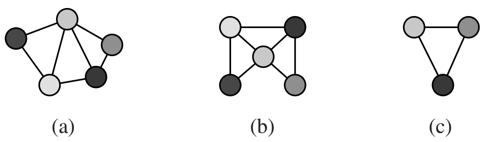


Figure 7.3. Graph (b) is isomorphic to (a), and graph (c) is isomorphic to a subgraph of (a). Node attributes are indicated by different shades of gray.

such that $\nu_1(e_1) = \nu_2(e_2)$

Two graphs are called isomorphic if there exists an isomorphism between them.

Obviously, isomorphic graphs are identical in both structure and labels. That is, a one-to-one correspondence between each node of the first graph and each node of the second graph has to be found such that the edge structure is preserved and node and edge labels are consistent.

Unfortunately, no polynomial runtime algorithm is known for the problem of graph isomorphism [25]. That is, in the worst case, the computational complexity of any of the available algorithms for graph isomorphism is exponential in the number of nodes of the two graphs. However, since most scenarios encountered in practice are often different from the worst case, and furthermore, the labels of both nodes and edges very often help to substantially reduce the complexity of the search, the actual computation time can still be manageable. Polynomial algorithms for graph isomorphism have been developed for special kinds of graphs, such as trees [1], ordered graphs [38], planar graphs [34], bounded-valence graphs [45], and graphs with unique node labels [18].

Standard procedures for testing graphs for isomorphism are based on tree search techniques with backtracking. The basic idea is that a partial node matching, which assigns nodes from the two graphs to each other, is iteratively expanded by adding new node-to-node correspondences. This expansion is repeated until either the edge structure constraint is violated or node or edge labels are inconsistent. In this case a backtracking procedure is initiated, i.e. the last node mappings are iteratively undone until a partial node mapping is found for which an alternative extension is possible. Obviously, if there is no further possibility for expanding the partial node matching without violating the constraints, the algorithm terminates indicating that there is no isomorphism between the considered graphs. Conversely, finding a complete node-to-node correspondence without violating any of the structure or label constraints proves that the investigated graphs are isomorphic. In Fig. 7.3 (a) and (b) two isomorphic graphs are shown.

A well known, and despite its age still very popular, algorithm implementing the idea of a tree search with backtracking for graph isomorphism is described in [89]. A more recent algorithm for graph isomorphism, also based on the idea of tree search, is the VF algorithm and its successor VF2 [17]. Here the basic tree search algorithm is endowed with an efficiently computable heuristic which substantially reduces the search time. In [43] the tree search method for isomorphism is sped up by means of another heuristic derived from *Constraint Satisfaction*. Other algorithms for exact graph matching, which are not based on tree search techniques, are *Nauty* [50], and decision tree based techniques [51], to name just two examples. The reader is referred to [15] for an exhaustive list of exact graph matching algorithms developed since 1973.

Closely related to graph isomorphism is subgraph isomorphism, which can be seen as a concept describing subgraph equality. A subgraph isomorphism is a weaker form of matching in terms of requiring only that an isomorphism holds between a graph g_1 and a subgraph of g_2. Intuitively, subgraph isomorphism is the problem to detect if a smaller graph is identically present in a larger graph. In Fig. 7.3 (a) and (c), an example of subgraph isomorphism is given.

Definition 7.4 (Subgraph Isomorphism). *Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ be graphs. An injective function $f : V_1 \to V_2$ from g_1 to g_2 is a subgraph isomorphism if there exists a subgraph $g \subseteq g_2$ such that f is a graph isomorphism between g_1 and g.*

The tree search based algorithms for graph isomorphism [17, 43, 89], as well as the decision tree based techniques [51], can also be applied to the subgraph isomorphism problem. In contrast with the problem of graph isomorphism, subgraph isomorphism is known to be NP-complete [25]. As a matter of fact, subgraph isomorphism is a harder problem than graph isomorphism as one has not only to check whether a permutation of g_1 is identical to g_2, but we have to decide whether g_1 is isomorphic to any of the subgraphs of g_2 with equal size as g_1.

The process of graph matching primarily aims at identifying corresponding substructures in the two graphs under consideration. Through the graph matching procedure an associated similarity or dissimilarity score can be easily inferred. In view of this, graph isomorphism as well as subgraph isomorphism provide us with a basic similarity measure, which is 1 (maximum similarity) for (sub)graph isomorphic, and 0 (minimum similarity) for non-isomorphic graphs. Hence, two graphs must be completely identical, or the smaller graph must be identically contained in the other graph, to be deemed similar. Consequently, the applicability of this graph similarity measure is rather limited. Consider a case where most, but not all, nodes and edges in two graphs are identical. The rigid concept of (sub)graph isomorphism fails in such a situation in the sense of considering the two graphs to be totally dissimilar. Due to this observation, the formal concept of the largest common part of two graphs is established.

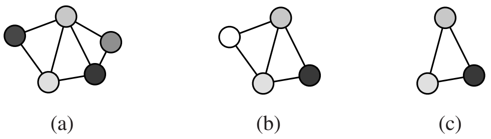


Figure 7.4. Graph (c) is a maximum common subgraph of graph (a) and (b).

Definition 7.5 (Maximum common subgraph). *Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ be graphs. A common subgraph of g_1 and g_2, $cs(g_1, g_2)$, is a graph $g = (V, E, \mu, \nu)$ 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. In Fig. 7.4(c) the maximum common subgraph is shown for the two graphs in Fig. 7.4(a) and (b). 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, 49]. The association graph of two graphs represents the whole set of possible node-to-node mappings that preserve the edge structure and labels 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 [10] the reader can find an experimental 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. For instance, in [12] such a distance measure is introduced, defined by

$$d_{MCS}(g_1, g_2) = 1 - \frac{|mcs(g_1, g_2)|}{\max\{|g_1|, |g_2|\}} \tag{7.1}$$

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; on the other hand, if two graphs have no part in common, their d_{MCS} distance is 1. 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 in [94], based on the idea of graph union, is

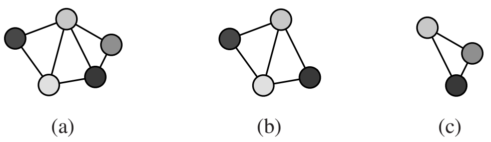


Figure 7.5. Graph (a) is a minimum common supergraph of graph (b) and (c).

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

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 on 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 [7] 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 [21] 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 concept of minimum common subgraph.

Definition 7.6 (Minimum common supergraph). *Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ be graphs. A common supergraph of g_1 and g_2, $CS(g_1, g_2)$, is a graph $g = (V, E, \mu, \nu)$ 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.*

In Fig. 7.5(a) the minimum common supergraph of the graphs in Fig. 7.5(b) and (c) is given. The computation of the minimum common supergraph can be reduced to the problem of computing a maximum common subgraph [11].

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 an empty 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, because of $|MCS(g_1, g_2)| = |g_1| + |g_2| - |mcs(g_1, g_2)|$, d_{UGU} and d_{MMCS} are identical. The same is true for d_{WGU} and d_{MMCSN}.

The main advantage of exact graph matching methods is their stringent definition and solid mathematical foundation. This advantage may turn into a disadvantage, however, because in exact graph matching for finding two graphs g_1 and g_2 to be similar, it is required that a significant part of the topology together with the corresponding node and edge labels in g_1 and g_2 have to be identical. In fact, this constraint is too rigid in some applications. For this reason, a large number of error-tolerant, or inexact, graph matching methods have been proposed, dealing with a more general graph matching problem than the one of (sub)graph isomorphism.

4. Inexact Graph Matching

Due to the intrinsic variability of the patterns under consideration and the noise resulting from the graph extraction process, it cannot be expected that two graphs representing the same class of objects are completely, or at least to a large part, identical in their structure. Moreover, if the node or edge label alphabet L is used to describe non-discrete properties of the underlying patterns, e.g. $L \subseteq \mathbb{R}^n$, it is most probable that the actual graphs differ somewhat from their ideal model. Obviously, such noise crucially hampers the applicability of exact graph matching techniques, and consequently exact graph matching is rarely used in real-world applications.

In order to overcome this drawback, it is advisable to endow the graph matching framework with a certain tolerance to errors. That is, the matching process must be able to accommodate the differences of the graphs by relaxing –to some extent– the underlying constraints. In the first part of this section the concept of graph edit distance is introduced to exemplarily illustrate the paradigm of inexact graph matching. In the second part, several other approaches to inexact graph matching are briefly discussed.

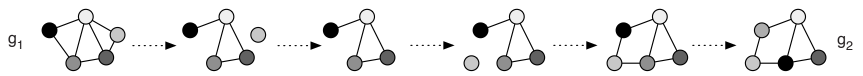


Figure 7.6. A possible edit path between graph g_1 and graph g_2 (node labels are represented by different shades of gray).

4.1 Graph Edit Distance

Graph edit distance [8, 71] 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 [93] 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 and labeling. A standard set of edit operations is given by *insertions*, *deletions*, and *substitutions* of both nodes and edges. Note that other edit operations, such as *merging* and *splitting* of nodes [2], can be useful in certain applications. Given two graphs, the source graph g_1 and the target graph g_2, the idea of graph edit distance is to delete some nodes and edges from g_1, relabel (substitute) some of the remaining nodes and edges, and insert some nodes and edges in g_2, such that g_1 is finally transformed into g_2. A sequence of edit operations $e_1, \ldots, e_k$ that transform g_1 into g_2 is called an *edit path* between g_1 and g_2. In Fig. 7.6 an example of an edit path between two graphs g_1 and g_2 is given. This edit path consists of three edge deletions, one node deletion, one node insertion, two edge insertions, and two node substitutions.

Let $\Upsilon(g_1, g_2)$ denote the set of all possible edit paths between two graphs g_1 and g_2. Clearly, every edit path between two graphs g_1 and g_2 is a model describing the correspondences found between the graphs' substructures. That is, the nodes of g_1 are either deleted or uniquely substituted with a node in g_2, and analogously, the nodes in g_2 are either inserted or matched with a unique node in g_1. The same applies for the edges. In [58] the idea of fuzzy edit paths was reported where both nodes and edges can be simultaneously mapped to several nodes and edges. The optimal fuzzy edit path is then determined by means of quadratic programming.

To find the most suitable edit path out of $\Upsilon(g_1, g_2)$, one introduces a cost for each edit operation, measuring the strength of the corresponding operation. The idea of such a cost is to define whether or not an edit operation represents a strong modification of the graph. Clearly, between two similar graphs, there should exist an inexpensive edit path, representing low cost operations, while for dissimilar graphs an edit path with high costs is needed. Consequently, the *edit distance* of two graphs is defined by the minimum cost edit path between two graphs.

Definition 7.7 (Graph Edit Distance). *Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ be the source and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ the target graph. The graph edit distance between g_1 and g_2 is defined by*

$$d(g_1, g_2) = \min_{(e_1,\ldots,e_k)\in\Upsilon(g_1,g_2)} \sum_{i=1}^{k} c(e_i),$$

where $\Upsilon(g_1, g_2)$ denotes the set of edit paths transforming g_1 into g_2, and c denotes the cost function measuring the strength $c(e)$ of edit operation e.

The definition of adequate and application-specific cost functions is a key task in edit distance based graph matching. Prior knowledge of the graphs' labels is often inevitable for graph edit distance to be a suitable proximity measure. This fact is often considered as one of the major drawbacks of graph edit distance. Yet, contrariwise, the possibility to parametrize graph edit distance by means of the cost function crucially amounts for the versatility of this dissimilarity model. That is, by means of graph edit distance it is possible to integrate domain specific knowledge about object similarity, if available, when defining the costs of the elementary edit operations. Furthermore, if in a particular case prior knowledge about the labels and their meaning is not available, automatic procedures for learning the edit costs from a set of sample graphs are available as well [55, 56].

The overall aim of the cost function is to favor weak distortions over strong modifications of the graph. Hence, the cost is defined with respect to the underlying node or edge labels, i.e. the cost $c(e)$ is a function depending on the edit operation e. Typically, for numerical node and edge labels the Euclidean distance can be used to model the cost of a particular substitution operation on the graphs. For deletions and insertions of both nodes and edges, often a constant cost τ_{node}/τ_{edge} is assigned. We refer to this cost function as *Euclidean Cost Function*.

The Euclidean cost function defines substitution costs proportional to the Euclidean distance of two respective labels. The basic intuition behind this approach is that the further away two labels are, the stronger is the distortion associated with the corresponding substitution. Note that any node substitution having a higher cost than $2 \cdot \tau_{node}$ will be replaced by a composition of a deletion and an insertion of the involved nodes (the same accounts for the edges). This behavior reflects the basic intuition that substitutions should be favored over deletions and insertions to a certain degree.

Optimal algorithms for computing the edit distance of graphs g_1 and g_2 are typically based on combinatorial search procedures that explore the space of all possible mappings of the nodes and edges of g_1 to the nodes and edges of g_2 [8]. A major drawback of those procedures is their computational complexity, which is exponential in the number of nodes of the involved graphs.

Consequently, the application of optimal algorithms for edit distance computations is limited to graphs of rather small size in practice.

To render graph edit distance computation less computationally demanding, a number of suboptimal methods have been proposed. In some approaches, the basic idea is to perform a local search to solve the graph matching problem, that is, to optimize local criteria instead of global, or optimal ones [57, 80]. In [40], a linear programming method for computing the edit distance of graphs with unlabeled edges is proposed. The method can be used to derive lower and upper edit distance bounds in polynomial time. Two fast but suboptimal algorithms for graph edit distance computation are proposed in [59]. The authors propose simple variants of a standard edit distance algorithm that make the computation substantially faster. In [20] another suboptimal method has been proposed. The basic idea is to decompose graphs into sets of subgraphs. These subgraphs consist of a node and its adjacent nodes and edges. The graph matching problem is then reduced to the problem of finding a match between the sets of subgraphs. In [67] a method somewhat similar to the method described in [20] is proposed. However, while the optimal correspondence between local substructures is found by dynamic programming in [20], a bipartite matching procedure [53] is employed in [67].

4.2 Other Inexact Graph Matching Techniques

Several other important classes of error-tolerant graph matching algorithms have been proposed. Among others, algorithms based on Artificial Neural Networks, Relaxation Labeling, Spectral Decompositions, and Graph Kernels have been reported.

Artificial Neural Networks. One class of error-tolerant graph matching methods employs *artificial neural networks*. In two seminal papers [24, 81] it is shown that neural networks can be used to classify directed acyclic graphs. The algorithms are based on an energy minimization framework, and use some kind of Hopfield network [84]. Hopfield networks consist of a set of neurons connected by synapses such that, upon activation of the network, the neuron output is fed back into the network. By means of an iterative learning procedure the given energy criterion is minimized. Similar to the approach of relaxation labeling (see below), compatibility coefficients are used to evaluate whether two nodes or edges constitute a successful match.

In [83] the optimization procedure is stabilized by means of a Potts MFT network. In [85] a self-organizing Hopfield network is introduced that learns most of the network parameters and eliminates the need for specifying them a priori. In [52, 72] the graph neural network is crucially extended such that also undirected and acyclic graphs can be processed. The general idea is to represent the nodes of a graph in an encoding network. In this encoding network local transition functions and local output functions are employed, expressing the dependency of a node on its neighborhood and describing how the output is produced, respectively. As both functions are implemented by feedforward neural networks, the encoding network can be interpreted as a recurrent neural network.

Further examples of graph matching based on artificial neural networks can be found in [37, 73, 101]

Relaxation Labeling. Another class of error-tolerant graph matching methods employs *relaxation labeling techniques*. The basic idea of this particular approach is to formulate the graph matching problem as a labeling problem. Each node of one graph is to be assigned to one label out of a discrete set of possible labels, specifying a matching node of the other graph. During the matching process, Gaussian probability distributions are used to model compatibility coefficients measuring how suitable each candidate label is. The initial labeling, which is based on the node attributes, node connectivity, and other information available, is then refined in an iterative procedure until a sufficiently accurate labeling, i.e. a matching of two graphs, is found. Based on the pioneering work presented in [22], the idea of relaxation labeling has been refined in several contributions. In [30, 41] the probabilistic framework for relaxation labeling is endowed with a theoretical foundation. The main drawback of the initial formulation of this technique, viz. the fact that node and edge labels are used only in the initialization of the matching process, is overcome in [14]. A significant extension of the framework is introduced in [97] where a Bayesian consistency measure is adapted to derive a graph distance. In [35] this method is further improved by taking also edge labels into account in the evaluation of the consistency measure. The concept of Bayesian graph edit distance, which in fact builds up on the idea of probabilistic relaxation, is presented in [54]. The concept has also been successfully applied to special kinds of graphs, such as trees [87].

Spectral Methods. *Spectral methods* build a further class of graph matching procedures [13, 47, 70, 78, 90, 98]. The general idea of this approach is based on the following observation. The eigenvalues and the eigenvectors of the adjacency or Laplacian matrix of a graph are invariant with respect to node permutation. Hence, if two graphs are isomorphic, their structural matrices will have the same eigendecomposition. The converse, i.e. deducing from the equality of eigendecompositions to graph isomorphism, is not true in general. However, by representing the underlying graphs by means of the eigendecomposition of their structural matrix, the matching process of the graphs can be conducted on some features derived from their eigendecomposition. The main problem of spectral methods is that they are rather sensitive towards structural errors, such as missing or spurious nodes. Moreover, most of these methods are purely structural, in the sense that they are only applicable to unlabeled graphs, or they allow only severely constrained label alphabets.

Graph Kernel. Kernel methods were originally developed for vectorial representations, but the kernel framework can be extended to graphs in a very natural way. A number of *graph kernels* have been designed for graph matching [26, 57]. A seminal contribution is the work on convolution kernels, which provides a general framework for dealing with complex objects that consist of simpler parts [32, 95]. Convolution kernels infer the similarity of complex objects from the similarity of their parts.

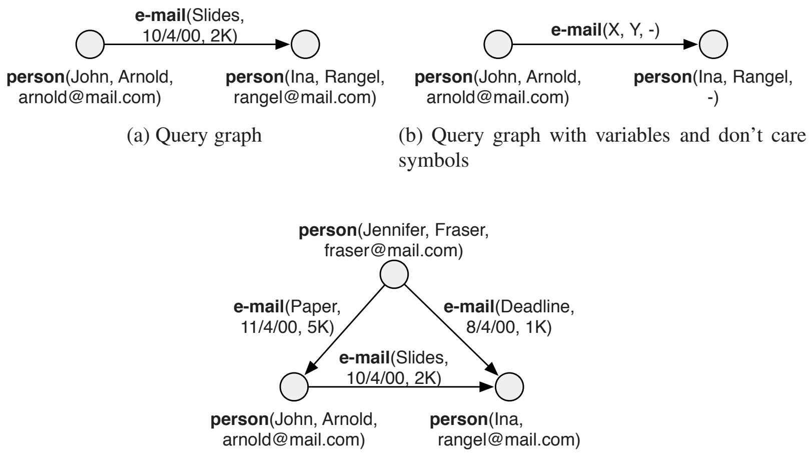
A second class of graph kernels is based on the analysis of random walks in graphs. These kernels measure the similarity of two graphs by the number of random walks in both graphs that have all or some labels in common [5, 27]. In [27] an important result is reported. It is shown that the number of matching walks in two graphs can be computed by means of the product graph of two graphs, without the need to explicitly enumerate the walks. In order to handle continuous labels the random walk kernel has been extended in [5]. This extension allows one to also take non-identically labeled walks into account.

A third class of graph kernels is given by diffusion kernels. The kernels of this class are defined with respect to a base similarity measure which is used to construct a valid kernel matrix [42, 79, 92]. This base similarity measure only needs to satisfy the condition of symmetry and can be defined for any kind of objects.

Miscellaneous Methods. Several other error-tolerant graph matching methods have been proposed in the literature, for instance, graph matching based on the Expectation Maximization algorithm [46], on replicator equations [61], and on graduated assignment [28]. Random walks in graphs [29, 69], approximate least-squares and interpolation theory algorithms [91], and random graphs [99] have also been employed for error-tolerant graph matching.

5. Graph Matching for Data Mining and Information Retrieval

The use of graphs and graph matching has become a promising approach in data mining and related areas [16]. In fact, querying graph databases has a long tradition and dates back to the time when the first algorithms for subgraph isomorphism detection became available. Yet, the use of conventional subgraph isomorphism in graph based data mining implicates severe limitations. First of all, the underlying database graph often includes a rather large number of attributes, some of which might be irrelevant for a particular query. The second


e-mail(Slides, 10/4/00, 2K)
person(John, Arnold, arnold@mail.com)
person(Ina, Rangel, rangel@mail.com)
e-mail(X, Y, -)
person(John, Arnold, arnold@mail.com)
person(Ina, Rangel, -)
person(Jennifer, Fraser, fraser@mail.com)
e-mail(Paper, 11/4/00, 5K)
e-mail(Deadline, 8/4/00, 1K)
e-mail(Slides, 10/4/00, 2K)
person(John, Arnold, arnold@mail.com)
person(Ina, rangel@mail.com)

(a) Query graph

(b) Query graph with variables and don't care symbols

(c) Database graph

Figure 7.7. Query and database graphs.

restriction arises from the limited answer format provided by conventional subgraph isomorphism which is only able to check whether or not a query graph is embedded in a larger database graph. Thirdly, subgraph isomorphism in its original mode does not allow constraints that may be imposed on the attributes of a query to model restrictions or dependencies.

The generalized subgraph isomorphism retrieval procedure described in [6] overcomes these three restrictions. First, the approach offers the possibility to mask out attributes in queries. To this end, *don't care* values are introduced for attributes that are irrelevant. Secondly, to make the retrieval of more specific information from the database graph possible than just a binary decision **yes** or **no**, *variables* are used. By means of these variables, one is able to retrieve values of specific attributes from the database graph. Thirdly, the concept of *constrained variables*, for example, variables that can assume only values from a certain interval, allows one to define more specific queries.

The approach to knowledge mining and information retrieval proposed in [6] is based on the idea of specifying a query by means of a query graph, which can be used to extract information from a large database graph. In contrast with Definition 7.1, the graphs employed are defined in a more general way. Rather than using just a single label, each node in a graph is labeled by a type and some attributes. The same accounts for the edges. In Fig. 7.7 (a) an example of a query graph is shown. In this illustration nodes are of the type *person* and labeled with the person's first and second name, and e-mail address. Edges are of the type *e-mail* and labeled with the e-mail's subject, the date, and the size. Note that in general there may occur nodes as well as edges of different type in the same graph.

Query graphs are more general than common graphs in the sense that don't care symbol and variables may occur as the values of attributes on the nodes and edges. The purpose of the variables is to define those attributes whose values are to be returned as an answer to a query (we will come back to this point later). In Fig. 7.7 (b) an example of a query graph with variables (X, Y) and don't care symbols $(-)$ is given. According to this query, we are particularly interested in the subject (X) and the date (Y) of an e-mail sent from John Arnold to Ina Rangel. As we do not care about the size of the e-mail and we do not know the e-mail address of Ina Rangel, two don't care symbols are used. Variables may also occur in a query because they may be used to express constraints on one or several attribute values. A constraint on a set of variables occurring in a query graph is a condition on one or several variables that evaluates to **true** or **false** if we assign a concrete attribute value to each variable. For instance, the query in Fig. 7.7 (b) can be augmented by the constraint that the e-mail in question was sent between October 1 and October 3 (formally 9/31/00 $< Y <$ 10/4/00).

Once the query graph has been constructed by the user, it is matched against a database graph. The process of matching a query graph to a database graph essentially means that we want to find out whether there exists a subgraph isomorphism from the query to the database graph. Obviously, as the query graph may include don't care symbols and variables, we need a more general notion of subgraph isomorphism than the one provided in Definition 7.4. Such a generalized subgraph isomorphism between a query and a database graph is referred to as a *match*, i.e., if a query graph q matches a database graph G, we call the injective function f a match between q and G. Note that for given q and G and a given set of constraints over the variables in q, there can be zero, one, or more than one matches.

For a match we require each edge of the query graph being included in the database graph. A node, u, can be mapped, via injective function f, only to a node of the same type. If the (type, attribute)-pair of a node u of the query graph includes an attribute value x_i, then it is required that the same value occur at the corresponding position in the (type, attribute)-pair of the node $f(u)$ in the database graph. Don't care symbols occurring in the (type, attribute)-pair of a node u will match any attribute value at the corresponding position in the (type, attribute)-pair of node $f(u)$. Similarly, unconstrained variables match any attribute value at their corresponding position in $f(u)$. In case there exist constraints on a variable in the query graph, the attribute values at the corresponding positions in $f(u)$ must satisfy these constraints.

By means of variables we indicate which attribute values are to be returned by our knowledge mining system as an answer to a query. Therefore, the answer to a query can be **no**, if there is no such structure as the query graph contained as a substructure in the database graph, or **yes** if the query graph exists (at least once) as a substructure in the database graph and the query graph does not contain any answer variables. In the case where answer variables are defined in the query graph and one or several matches are found an individual answer is generated for each match f_j. An answer is of the form $X_1 = x'_1, \ldots, X_n = x'_n$ where $X_1, \ldots, X_n$ are the answer variables occurring in the query and x'_i are the values of the attributes in the database graph that correspond to the variables X_i under match f_j. Obviously, there is a match between the query graph in Fig. 7.7 (b) and the database graph in Fig. 7.7 (c). Hence, the variables are linked by $X =$ Slides and $Y =$ 10/4/00.

The proposed system described so far does not return any information from the database graph whenever no match is found. However, in some cases this behavior may be undesirable. Let us consider, for instance, a query graph that contains spurious attribute values or edges which do not occur in the underlying database graph. The graph matching framework presented so far merely returns the answer **no** as it finds no match in the database graph. However, we can easily endow the graph isomorphism framework with a certain tolerance to errors. To this end one can use graph edit distance. In cases when no perfect match of the query graph to the database graph is possible, the query is minimally modified such that a match becomes possible. The well-founded possibility of augmenting the data mining framework with some tolerance to errors definitely accounts for the power of this particular procedure based on graph matching.

In [6] an algorithmic procedure is described for finding matches between a query q and a database graph G. This procedure checks two given graphs, q and G, whether there exists a match from q to G by constructing all possible mappings f: $V_1 \to V_2$. This matching algorithm is of exponential complexity. However, as the underlying query graphs are typically limited in size and due to the fact that the attributes and constraints limit the potential search space for a match significantly, the computational complexity of this algorithm is usually still manageable, as shown in the experiments reported in [6].

For applications where large query graphs occur a novel approximate approach for querying graph databases has been introduced in [86]. This algorithm proceeds as follows. First, a number of important nodes from the query graph are selected. The importance of the nodes can be measured, for instance, by their degree. Using the label, the degree, and information about a node's local neighborhood, the most important nodes are matched against the database graph nodes. Clearly, by means of this procedure each node from the query graph may be mapped to several database nodes and vice versa. Given a quality criterion for the individual node mappings, a bipartite optimization procedure can be applied resulting in a one-to-one correspondence between query nodes and database nodes. The node pairs returned by the bipartite matching procedure serve us as *anchor points* of the complete matching. Based on these anchor points, the initial graph match is iteratively extended. For each node that has already been mapped to a database node, its nearby nodes (nodes that are at most two hops away) are tried to be mapped to database nodes. This extension is repeated until no more nodes can be added to the match. Clearly, in contrast with the method described in [6] this procedure is suboptimal in the sense of finding subgraphs in the database graph that are similar, but not necessarily equal, to the query graph. In exchange, a graph matching framework applicable to very large query graphs (hundreds to thousands of nodes and edges) is established.

6. Vector Space Embeddings of Graphs via Graph Matching

Classification and clustering of objects are common tasks in intelligent information processing. Classification refers to the process of assigning an unknown input object to one out of a given set of classes, while clustering refers to the process of dividing a set of given objects into homogeneous groups. A vast number of algorithms for classification [19] and clustering [100] have been proposed in the literature. Almost all of these algorithms have been designed for object representations given in terms of feature vectors. This means that there exists a severe lack of algorithmic tools for graph classification and clustering. This lack is mainly due to the fact that some of the basic operations needed in classification as well as clustering are not available for graphs. In other words, while it is possible to define graph dissimilarity measures via specific graph matching procedures, this is often not sufficient for standard algorithms in intelligent information processing. In fact, graph distance based pattern recognition is basically limited to nearest-neighbor classification and k-medians clustering [57].

A promising direction to overcome this severe limitation is graph embedding into vector spaces. Basically, such an embedding of graphs establishes access to the rich repository of algorithmic tools developed for vectorial representations. In [47], for instance, features derived from the eigendecomposition of graphs are studied. Another idea deals with string edit distance applied to the eigensystem of graphs [96]. This procedure results in distances between graphs which are used to embed the graphs into a vector space by means of multidimensional scaling. In [98] the authors turn to the spectral decomposition of the Laplacian matrix of a graph. They show how the elements of the spectral matrix of the Laplacian can be used to construct symmetric polynomials. In order to encode graphs as vectors, the coefficients of these polynomials are used as graph features. Another approach for graph embedding has been proposed in [70]. The authors use the relationship between the Laplace-

Beltrami operator and the graph Laplacian to embed a graph in a Riemannian manifold.

The present section considers a new class of graph embedding procedures which are based on dissimilarity representation and graph matching. Originally the idea was proposed in [60] in order to map feature vectors into dissimilarity spaces. Later it was generalized to string based object representation [82] and to the domain of graphs [62]. Graphs from a given problem domain are mapped to vector spaces by computing the distance to some predefined prototype graphs. The resulting distances can be used as a vectorial representation of the considered graph.

Formally, assume we have a set of sample graphs, $\mathcal{T} = \{g, \ldots, g_N\}$ from some graph domain $\mathcal{G}$ and an arbitrary graph dissimilarity measure $d : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}$. Note that $\mathcal{T}$ can be any kind of graph set. However, for the sake of convenience we define $\mathcal{T}$ as a training set of given graphs. After selecting a set of prototypical graphs $\mathcal{P} \subseteq \mathcal{T}$, we compute the dissimilarity of a given input graph g to each prototype graph $p_i \in \mathcal{P}$. Note that g can be an element of $\mathcal{T}$ or any other graph set $\mathcal{S}$. Given n prototypes, i.e. $\mathcal{P} = \{p_1, \ldots, p_n\}$, this procedure leads to n dissimilarities, $d_1 = d(g, p_1), \ldots, d_n = d(g, p_n)$, which can be arranged in an n-dimensional vector $(d_1, \ldots, d_n)$.

Definition 7.8 (Graph Embedding). *Let us assume a graph domain $\mathcal{G}$ is given. If $\mathcal{T} = \{g, \ldots, g_N\} \subseteq \mathcal{G}$ is a training set with N graphs and $\mathcal{P} = \{p_1, \ldots, p_n\} \subseteq \mathcal{T}$ is a prototype set with n graphs, the mapping*

$$\varphi_n^{\mathcal{P}} : \mathcal{G} \rightarrow \mathbb{R}^n$$

is defined as the function

$$\varphi_n^{\mathcal{P}}(g) = (d(g, p_1), \ldots, d(g, p_n)),$$

where $d(g, p_i)$ is any graph dissimilarity measure between graph g and the i-th prototype graph.

Obviously, by means of this definition we obtain a vector space where each axis corresponds to a prototype graph $p_i \in \mathcal{P}$ and the coordinate values of an embedded graph g are the distances of g to the elements in $\mathcal{P}$. In this way we can transform any graph g from the training set $\mathcal{T}$ as well as any other graph set $\mathcal{S}$ (for instance a validation or a test set of a classification problem), into a vector of real numbers. In [65] this procedure is further generalized towards Lipschitz embeddings [33]. Rather than singleton reference sets (i.e. prototypes $p_1, \ldots, p_n$), sets of prototypes $\mathcal{P}_1, \ldots, \mathcal{P}_n$ are used for embedding the graphs via dissimilarities.

The embedding procedure proposed in [62] makes use of graph edit distance. Note, however, that any other graph dissimilarity measure can be used as well. Yet, using graph edit distance allows us to deal with a large class of graphs (directed, undirected, unlabeled, node and/or edge labels from any finite or infinite domain). Furthermore, a high degree of robustness against various graph distortions can be expected. Hence, in contrast with other graph embedding techniques, where sometimes restrictions on the type of underlying graph are imposed (e.g. [47, 70, 98]), this approach is distinguished by a high degree of flexibility in the graph definition. Since the computation of graph edit distance is exponential in the number of nodes for general graphs, the complexity of this graph embedding is exponential as well. However, as mentioned in Sect. 4, there exist efficient approximation algorithms for graph edit distance computation with cubic time complexity (e.g. the procedure described in [67]). Consequently, given n predefined prototypes the embedding of one particular graph is established by means of n distance computations with polynomial time.

Dissimilarity embeddings are closely related to kernel methods [75, 77]. In the kernel approach objects are described by means of pairwise kernel functions, while in the dissimilarity approach they are described by pairwise dissimilarities. However, there is one fundamental difference between kernels and dissimilarity embeddings. In the former method, the kernel values are interpreted as dot products in some implicitly existing feature space. By means of kernel machines, the underlying algorithm is eventually carried out in this kernel feature space. In the latter approach, the set of dissimilarities is interpreted as a novel vectorial description of the object under consideration. Hence, no implicit feature space, but an explicit dissimilarity space is obtained.

Obviously, the embedding paradigm established by mapping $\varphi_n^{\mathcal{P}} : \mathcal{G} \to \mathbb{R}^n$ constitutes a foundation for a novel class of graph kernels. One can define a valid graph kernel κ based on the graph embedding by computing the standard dot product of two graph maps in the resulting vector space. Formally,

$$\kappa_{\langle\rangle}(g_1, g_2) = \langle \varphi_n^{\mathcal{P}}(g_1), \varphi_n^{\mathcal{P}}(g_2) \rangle \quad .$$

Note that this approach is very similar to the empirical kernel map described in [88] where general similarity measures are turned into kernel functions. Of course, not only the standard dot product can be used but any valid kernel function defined for vectors. For instance an RBF kernel function

$$\kappa_{RBF}(g_1, g_2) = \exp\left(-\gamma ||\varphi_n^{\mathcal{P}}(g_1) - \varphi_n^{\mathcal{P}}(g_2)||^2\right)$$

with $\gamma > 0$ can thus be applied to graph maps.

The selection of the n prototypes $\mathcal{P} = \{p_1, \ldots, p_n\}$ is a critical issue since not only the prototypes $p_i \in \mathcal{P}$ themselves but also their number n affect the resulting graph mapping $\varphi_n^{\mathcal{P}}(\cdot)$ and thus the performance of the corresponding pattern recognition algorithm. A good selection of n prototypes seems to be crucial to succeed with the classification or clustering algorithm in the embedding vector space. A first and very simple idea might be to use all available training graphs from $\mathcal{T}$ as prototypes. Yet, two severe shortcomings arise with such a plain approach. First, the dimensionality of the resulting vector space is equal to the size N of the training set $\mathcal{T}$. Consequently, if the training set is large, the dimensionality of the feature vectors will be high, which possibly leads to overfitting effects and compromises computational efficiency. Secondly, the presence of similar prototypes as well as outlier graphs in the training set $\mathcal{T}$ is most likely. Therefore, redundant, noisy, or irrelevant information will be captured in the graph maps which in turn may harm the performance of the underlying algorithms.

The selection of prototypes for graph embedding has been addressed in various papers [62, 64, 66, 68]. In [62], for instance, a number of *prototype selection methods* are discussed. These selection strategies use some heuristics based on the underlying dissimilarities in the original graph domain. The basic idea of these approaches is to select prototypes from $\mathcal{T}$ that reflect the distribution of the training set $\mathcal{T}$ or cover a predefined region of $\mathcal{T}$ in the best possible way.

A severe shortcoming of such heuristic prototype selection strategies is that the dimensionality of the embedding space has to be determined by the user. In other words, the number of prototypes to be selected by a certain prototype selection algorithm has to be experimentally defined by means of the target algorithm on a validation set. In order to overcome this limitation, in [68], various *prototype reduction schemes* [3] are adopted for the task of graph embedding. In contrast with the heuristic prototype selection strategies, with these procedures the number of prototypes n, i.e. the resulting dimensionality of the vector space, is defined by an algorithmic procedure.

Another solution to the problem of noisy and redundant vectors with too high dimensionality is offered by the following procedure. Rather than selecting the prototypes beforehand, the embedding is carried out first and then the problem of prototype selection is reduced to a feature subset selection problem. That is, for graph embedding all available elements from the training set are used as prototypes, i.e. we define $\mathcal{P} = \mathcal{T}$. Next, a huge number of different feature selection strategies [23, 36, 39] can be applied to the resulting large scale vectors eliminating redundancies and noise, finding good features, and reducing the dimensionality. In [66], for instance, principal component analysis (PCA) [39] and Fisher linear discriminant analysis (LDA) [23] are applied to the vector space embedded graphs. Rather than traditional PCA, in [64], kernel PCA [76] is used for feature transformation.

Regardless of the strategy actually employed for the task of prototype selection, it has been experimentally shown that the general graph embedding procedure proposed in [62] has great potential. Its performance in various graph classification and clustering problems was evaluated and compared to alternative methods, including various graph kernels [62–66]. The data sets used in the experimental evaluation are publicly available[1].

7. Conclusions

Due to the ability of graphs to represent properties of entities and binary relations at the same time, a growing interest in graph-based object representation in intelligent information processing can be observed. In the fields of bioinformatics and chemoinformatics, for instance, graph based representations have been intensively used [5, 48]. Another field of research where graphs have been studied with emerging interest is that of web content mining [74]. Image classification is a further area of research where graph based representation draws the attention [31]. Finally, we like to mention computer network analysis, where graphs have been used to detect network anomalies and predict abnormal events [9].

The concept of similarity or dissimilarity is an important issue in many application domains. In case where graphs are employed as representation formalism, various procedures for evaluating proximity, i.e. similarity or dissimilarity, of graphs have been proposed [15]. The process of evaluating the similarity of two graphs is commonly referred to as graph matching. Graph matching has successfully been applied to various problems in pattern recognition, computer vision, machine learning, data mining, and related fields.

In the case of exact graph matching, the graph extraction process is assumed to be structurally flawless, i.e. the conversion of the underlying data into graphs always proceeds without errors. Otherwise, if distortions are present, graph and subgraph isomorphism detection are rather unsuitable, which seriously restricts the applicability of exact graph matching algorithms.

Inexact methods, sometimes also referred to as error-tolerant methods, are characterized by their ability to cope with errors, or non-corresponding parts, in terms of 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 on the fundamental concepts and on two recent applications. In the first application, it is shown how the concept of subgraph isomorphism can be extended, such that a powerful and flexible information retrieval framework is established. This framework can be used to retrieve information from large database graphs by means of query graphs. In

[1](`www.iam.unibe.ch/fki/databases/iam-graph-database`)

a further application it is shown how graphs can be embedded in vector spaces by means of dissimilarities derived from graph edit distance or some other dissimilarity measure. The crucial benefit of such a graph embedding is that it instantly makes available all algorithmic tools originally developed for vectorial object descriptions.

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison Wesley, 1974.

[2] R. Ambauen, S. Fischer, and H. Bunke. Graph edit distance with node splitting and merging and its application to diatom identification. In E. Hancock and M. Vento, editors, *Proc. 4th Int. Workshop on Graph Based Representations in Pattern Recognition*, LNCS 2726, pages 95–106. Springer, 2003.

[3] J.C. Bezdek and L. Kuncheva. Nearest prototype classifier designs: An experimental study. *Int. Journal of Intelligent Systems*, 16(12):1445–1473, 2001.

[4] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2008.

[5] K. Borgwardt, C. Ong, S. Schonauer, S. Vishwanathan, A. Smola, and H.-P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(1):47–56, 2005.

[6] A. Brugger, H. Bunke, P. Dickinson, and K Riesen. Generalized graph matching for data mining and information retrieval. In P. Perner, editor, *Advances in Data Mining. Medical Applications, E-Commerce, Marketing, and Theoretical Aspects*, LNCS 5077, pages 298–312. Springer, 2008.

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

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

[9] H. Bunke, P.J. Dickinson, M. Kraetzl, and W.D. Wallis. *A Graph-Theoretic Approach to Enterprise Network Dynamics*, volume 24 of *Progress in Computer Science and Applied Logic (PCS)*. Birkhauser, 2007.

[10] H. Bunke, P. Foggia, C. Guidobaldi, C. Sansone, and M. Vento. A comparison of algorithms for maximum common subgraph on randomly connected graphs. In T. Caelli, A. Amin, R. Duin, M. Kamel, and D. de Ridder, editors, *Structural, Syntactic, and Statistical Pattern Recognition*, pages 85–106. Springer, 2002. LNCS 2396.

[11] H. Bunke, X. Jiang, and A. Kandel. On the minimum common supergraph of two graphs. *Computing*, 65(1):13–25, 2000.

[12] H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3):255–259, 1998.

[13] T. Caelli and S. Kosinov. Inexact graph matching using eigen-subspace projection clustering. *Int. Journal of Pattern Recognition and Artificial Intelligence*, 18(3):329–355, 2004.

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

[15] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *Int. Journal of Pattern Recognition and Artificial Intelligence*, 18(3):265–298, 2004.

[16] D. Cook and L. Holder, editors. *Mining Graph Data*. Wiley-Interscience, 2007.

[17] L.P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 26(20):1367–1372, 2004.

[18] P.J. Dickinson, H. Bunke, A. Dadej, and M. Kraetzl. Matching graphs with unique node labels. *Pattern Analysis and Applications*, 7(3):243–254, 2004.

[19] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Wiley-Interscience, 2nd edition, 2000.

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

[21] M.-L. Fernandez and G. Valiente. A graph distance metric combining maximum common subgraph and minimum common supergraph. *Pattern Recognition Letters*, 22(6–7):753–758, 2001.

[22] M.A. Fischler and R.A. Elschlager. The representation and matching of pictorial structures. *IEEE Trans. on Computers*, 22(1):67–92, 1973.

[23] R.A. Fisher. The statistical utilization of multiple measurements. In *Annals of Eugenics*, volume 8, pages 376–386, 1938.

[24] P. Frasconi, M. Gori, and A. Sperduti. A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks*, 9(5):768–786, 1998.

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

[26] T. Gartner. *Kernels for Structured Data*. World Scientific, 2008.

[27] T. Gartner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In B. Schəlkopf and M. Warmuth, editors, *Proc. 16th Annual Conf. on Learning Theory*, pages 129–143, 2003.

[28] S. Gold and A. Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(4):377–388, 1996.

[29] M. Gori, M. Maggini, and L. Sarti. Exact and approximate graph matching using random walks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1100–1111, 2005.

[30] E.R. Hancock and J. Kittler. Discrete relaxation. *Pattern Recognition*, 23(7):711–733, 1990.

[31] Z. Harchaoui and F. Bach. Image classification with segmentation graph kernels. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[32] D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, University of California, Santa Cruz, 1999.

[33] G. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. *IEEE Trans. on Pattern Analysis ans Machine Intelligence*, 25(5):530–549, 2003.

[34] J.E. Hopcroft and J. Wong. Linear time algorithm for isomorphism of planar graphs. In *Proc. 6th Annual ACM Symposium on Theory of Computing*, pages 172–184, 1974.

[35] B. Huet and E.R. Hancock. Shape recognition from large image libraries by inexact graph matching. *Pattern Recognition Letters*, 20(11–13):1259–1269, 1999.

[36] A. Jain and D. Zongker. Feature selection: Evaluation, application, and small sample performance. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(2):153–158, 1997.

[37] B. Jain and F. Wysotzki. Automorphism partitioning with neural networks. *Neural Processing Letters*, 17(2):205–215, 2003.

[38] X. Jiang and H. Bunke. Optimal quadratic-time isomorphism of ordered graphs. *Pattern Recognition*, 32(17):1273–1283, 1999.

[39] I. Jolliffe. *Principal Component Analysis*. Springer, 1986.

[40] D. Justice and A. Hero. A binary linear programming formulation of the graph edit distance. *IEEE Trans. on Pattern Analysis ans Machine Intelligence*, 28(8):1200–1214, 2006.

[41] J. Kittler and E.R. Hancock. Combining evidence in probabilistic relaxation. *Int. Journal of Pattern Recognition and Art. Intelligence*, 3(1):29–51, 1989.

[42] R.I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *Proc. 19th Int. Conf. on Machine Learning*, pages 315–322, 2002.

[43] J. Larrosa and G. Valiente. Constraint satisfaction algorithms for graph pattern matching. *Mathematical Structures in Computer Science*, 12(4):403–422, 2002.

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

[45] E.M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and Systems Sciences*, 25:42–65, 1982.

[46] B. Luo and E. Hancock. Structural graph matching using the EM algorithm and singular value decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1120–1136, 2001.

[47] B. Luo, R. Wilson, and E.R. Hancock. Spectral embedding of graphs. *Pattern Recognition*, 36(10):2213–2223, 2003.

[48] P. Mahé, N. Ueda, and T. Akutsu. Graph kernels for molecular structures – activity relationship analysis with support vector machines. *Journal of Chemical Information and Modeling*, 45(4):939–951, 2005.

[49] J.J. McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software Practice and Experience*, 12:23–34, 1982.

[50] B.D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

[51] B.T. Messmer and H. Bunke. A decision tree approach to graph and subgraph isomorphism detection. *Pattern Recognition*, 32:1979–1998, 1008.

[52] A. Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

[53] J. Munkres. Algorithms for the assignment and transportation problems. In *Journal of the Society for Industrial and Applied Mathematics*, volume 5, pages 32–38, March 1957.

[54] R. Myers, R.C. Wilson, and E.R. Hancock. Bayesian graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):628–635, 2000.

[55] M. Neuhaus and H. Bunke. Self-organizing maps for learning the edit costs in graph matching. *IEEE Transactions on Systems, Man, and Cybernetics (Part B)*, 35(3):503–514, 2005.

[56] M. Neuhaus and H. Bunke. Automatic learning of cost functions for graph edit distance. *Information Sciences*, 177(1):239–247, 2007.

[57] M. Neuhaus and H. Bunke. *Bridging the Gap Between Graph Edit Distance and Kernel Machines*. World Scientific, 2007.

[58] M. Neuhaus and H. Bunke. A quadratic programming approach to the graph edit distance problem. In F. Escolano and M. Vento, editors, *Proc.*

6th Int. Workshop on Graph Based Representations in Pattern Recognition, LNCS 4538, pages 92–102, 2007.

[59] M. Neuhaus, K. Riesen, and H. Bunke. Fast suboptimal algorithms for the computation of graph edit distance. In Dit-Yan Yeung, J.T. Kwok, A. Fred, F. Roli, and D. de Ridder, editors, *Proc. 11.th int. Workshop on Strucural and Syntactic Pattern Recognition*, LNCS 4109, pages 163–172. Springer, 2006.

[60] E. Pekalska and R. Duin. *The Dissimilarity Representation for Pattern Recognition: Foundations and Applications*. World Scientific, 2005.

[61] M. Pelillo. Replicator equations, maximal cliques, and graph isomorphism. *Neural Computation*, 11(8):1933–1955, 1999.

[62] K. Riesen and H. Bunke. Graph classification based on vector space embedding. *Int. Journal of Pattern Recognition and Artificial Intelligence*, 2008. accepted for publication.

[63] K. Riesen and H. Bunke. Kernel k-means clustering applied to vector space embeddings of graphs. In L. Prevost, S. Marinai, and F. Schwenker, editors, *Proc. 3rd IAPR Workshop Artificial Neural Networks in Pattern Recognition*, LNAI 5064, pages 24–35. Springer, 2008.

[64] K. Riesen and H. Bunke. Non-linear transformations of vector space embedded graphs. In A. Juan-Ciscar and G. Sanchez-Albaladejo, editors, *Pattern Recognition in Information Systems*, pages 173–186, 2008.

[65] K. Riesen and H. Bunke. On Lipschitz embeddings of graphs. In I. Lovrek, R.J. Howlett, and L.C. Jain, editors, *Proc. 12th International Conference, Knowledge-Based Intelligent Information and Engineering Systems, Part I*, LNAI 5177, pages 131–140. Springer, 2008.

[66] K. Riesen and H. Bunke. Reducing the dimensionality of dissimilarity space embedding graph kernels. *Engineering Applications of Artificial Intelligence*, 22(1):48–56, 2008.

[67] K. Riesen and H. Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, 27(4):950–959, 2009.

[68] K. Riesen and H. Bunke. Dissimilarity based vector space embedding of graphs using prototype reduction schemes. Accepted for publication in Machine Learning and Data Mining in Pattern Recognition, 2009.

[69] A. Robles-Kelly and E.R. Hancock. String edit distance, random walks and graph matching. *Int. Journal of Pattern Recognition and Artificial Intelligence*, 18(3):315–327, 2004.

[70] A. Robles-Kelly and E.R. Hancock. A Riemannian approach to graph embedding. *Pattern Recognition*, 40:1024–1056, 2007.

[71] A. Sanfeliu and K.S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics (Part B)*, 13(3):353–363, 1983.

[72] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[73] K. Schadler and F. Wysotzki. Comparing structures using a Hopfield-style neural network. *Applied Intelligence*, 11:15–30, 1999.

[74] A. Schenker, H. Bunke, M. Last, and A. Kandel. *Graph-Theoretic Techniques for Web Content Mining*. World Scientific, 2005.

[75] B. Schølkopf and A. Smola. *Learning with Kernels*. MIT Press, 2002.

[76] B. Schølkopf, A. Smola, and K.-R. Mʉller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.

[77] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[78] A. Shokoufandeh, D. Macrini, S. Dickinson, K. Siddiqi, and S.W. Zucker. Indexing hierarchical structures using graph spectra. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1125–1140, 2005.

[79] A. Smola and R. Kondor. Kernels and regularization on graphs. In *Proc. 16th. Int. Conf. on Comptuational Learning Theory*, pages 144–158, 2003.

[80] S. Sorlin and C. Solnon. Reactive tabu search for measuring graph similarity. In L. Brun and M. Vento, editors, *Proc. 5th Int. Worksho on Graph-based Representations in Pattern Recognition*, LNCS 3434, pages 172–182. Springer, 2005.

[81] A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

[82] B. Spillmann, M. Neuhaus, H. Bunke, E. Pekalska, and R. Duin. Transforming strings to vector spaces using prototype selection. In Dit-Yan Yeung, J.T. Kwok, A. Fred, F. Roli, and D. de Ridder, editors, *Proc. 11.th int. Workshop on Strucural and Syntactic Pattern Recognition*, LNCS 4109, pages 287–296. Springer, 2006.

[83] P.N. Suganthan, E.K. Teoh, and D.P. Mital. Pattern recognition by graph matching using the potts MFT neural networks. *Pattern Recognition*, 28(7):997–1009, 1995.

[84] P.N. Suganthan, E.K. Teoh, and D.P. Mital. Pattern recognition by homomorphic graph matching using Hopfield neural networks. *Image Vision Computing*, 13(1):45–60, 1995.

[85] P.N. Suganthan, E.K. Teoh, and D.P. Mital. Self-organizing Hopfield network for attributed relational graph matching. *Image Vision Computing*, 13(1):61–73, 1995.

[86] Y. Tian and J.M. Patel. Tale: A tool for approximate large graph matching. In *IEEE 24th International Conference on Data Engineering*, pages 963–972, 2008.

[87] A. Torsello and E. Hancock. Computing approximate tree edit distance using relaxation labeling. *Pattern Recognition Letters*, 24(8):1089–1097, 2003.

[88] K. Tsuda. Support vector classification with asymmetric kernel function. In M. Verleysen, editor, *Proc. 7th European Symposium on Artifical Neural Netweorks*, pages 183–188, 1999.

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

[90] S. Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):695–703, 1988.

[91] M.A. van Wyk, T.S. Durrani, and B.J. van Wyk. A RKHS interpolator-based graph matching algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):988–995, 2003.

[92] J.-P. Vert and M. Kanehisa. Graph-driven features extraction from microarray data using diffusion kernels and kernel CCA. In *Advances in Neural Information Processing Systems*, volume 15, pages 1425–1432. MIT Press, 2003.

[93] 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.

[94] W.D. Wallis, P. Shoubridge, M. Kraetzl, and D. Ray. Graph distances using graph union. *Pattern Recognition Letters*, 22(6):701–704, 2001.

[95] C. Watkins. Dynamic alignment kernels. In A. Smola, P.L. Bartlett, B. Schølkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 39–50. MIT Press, 2000.

[96] R. Wilson and E.R. Hancock. Levenshtein distance for graph spectral features. In J. Kittler, M. Petrou, and M. Nixon, editors, *Proc. 17th Int. Conf. on Pattern Recognition*, volume 2, pages 489–492, 2004.

[97] R.C. Wilson and E. Hancock. Structural matching by discrete relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):634–648, 1997.

[98] R.C. Wilson, E.R. Hancock, and B. Luo. Pattern vectors from algebraic graph theory. *IEEE Trans. on Pattern Analysis ans Machine Intelligence*, 27(7):1112–1124, 2005.

[99] A.K.C. Wong and M. You. Entropy and distance of random graphs with application to structural pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(5):599–609, 1985.

[100] R. Xu and D. Wunsch. Survey of graph clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.

[101] Y. Yao, G.L. Marcialis, M. Pontil, P. Frasconi, and F. Roli. Combining flat and structured representations for fingerprint classification with recursive neural networks and support vector machines. *Pattern Recognition*, 36(2):397–406, 2003.