Graph Representation for Intelligent Information Processing - Fundamentals and Algorithms for Classification and Clustering

Prof. Horst Bunke
Institute of Informatics and Applied Mathematics
University of Bern, Switzerland

bunke@iam.unibe.ch
http://www.iam.unibe.ch/groups/fki
Contents

I General Methodology 2

1 Introduction to Pattern Recognition with Symbolic Data Structures 3

2 Pattern Recognition with Graphs 5
   2.1 Basic Definitions ............................. 5
   2.2 Basic Algorithms .............................. 8
      2.2.1 Graph and Subgraph Isomorphism ............ 9
      2.2.2 Maximum Common Subgraph .................. 12
   2.3 Graph Edit Distance ........................... 16
   2.4 Other Graph Distance Measures ................ 19
   2.5 Applications ................................ 20
      2.5.1 Character Recognition ...................... 20
      2.5.2 Diatom Classification ....................... 24
      2.5.3 Classification of Web Documents ............ 28
   2.6 Other Algorithms ................................ 31
      2.6.1 Network based Approach ..................... 31
      2.6.2 Decision Tree based Method .................. 34
   2.7 The Median Graph Problem ........................ 39
   2.8 Suboptimal Graph Edit Distance Computation .. 43
   2.9 Graphs with Unique Node Labels ................ 46
   2.10 Hypergraphs .................................. 48
   2.11 Information Retrieval with Graphs ............. 51
   2.12 Graph Databases for Experimental Work .......... 54

II Kernel Functions for Symbolic Data Structures 62

3 Graph Kernel Functions 64
   3.1 Graph Kernels based on Common Substructures ....... 64
   3.2 Walk Kernels ................................... 67
   3.3 Dissimilarity Based Graph Embedding ............. 71
4 General Kernel Theory

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>General Introduction</td>
<td>78</td>
</tr>
<tr>
<td>4.2</td>
<td>Nearest-Neighbor Classifier</td>
<td>83</td>
</tr>
<tr>
<td>4.3</td>
<td>Minimum Distance Classifier</td>
<td>83</td>
</tr>
<tr>
<td>4.4</td>
<td>Perceptron Algorithm</td>
<td>85</td>
</tr>
<tr>
<td>4.5</td>
<td>Support Vector Machine</td>
<td>88</td>
</tr>
<tr>
<td>4.6</td>
<td>K-means Clustering</td>
<td>89</td>
</tr>
<tr>
<td>4.7</td>
<td>Properties of Kernel Functions</td>
<td>92</td>
</tr>
</tbody>
</table>
Literature


Part I

General Methodology
Chapter 1

Introduction to Pattern Recognition with Symbolic Data Structures

The standard representation used in pattern recognition and related fields is feature vectors:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = (x_1, \ldots, x_n)' \in \mathbb{R}^n$$

Advantages of features vectors include:

- simple; low computational complexity
- many algorithms available for classification, clustering, dimensionality reduction, etc.

However, there are also disadvantages:

- no relations can be represented
- dimensionality is fixed for a particular application

In the following we consider symbolic data structures for object representation, especially strings and graphs:

- string $x = x_1 \ldots x_n, x_i \in A$, where $A$ is a finite alphabet, i.e. a finite set of symbols
Advantages of symbolic data structures (especially graphs) are:

- more powerful than feature vectors; relations can be represented
- dimensionality not fixed

Disadvantages of symbolic data structures include:

- high computational complexity
- many mathematical operations are not defined; many algorithms are not available
Chapter 2

Pattern Recognition with Graphs

Graphs do not only allow each object to have its individual size, but they are also able to represent relationships between different parts of an object. In fact, graphs include, as special cases, strings and feature vectors. On the other hand, we face the problems mentioned in Section 1.

2.1 Basic Definitions

We consider graphs with labels on their nodes and edges. \( L_V \) and \( L_E \) denote the alphabet of node and edge labels, respectively. Graphs with unlabelled nodes and edges are obtained, as special cases, if \( |L_V| = 1 \) and \( |L_E| = 1 \), respectively.

A graph is a 4-tuple \( g = (V, E, \alpha, \beta) \), where

- \( V \) is the finite set of nodes
- \( E \subseteq V \times V \) is the set of edges
- \( \alpha : V \rightarrow L_V \) is the node labelling function
- \( \beta : E \rightarrow L_E \) is the edge labelling function

Undirected graphs are obtained as a special case if for each edge \( e = (u, v) \) there exists an edge \( (v, u) \in E \) in the opposite direction with \( \beta(u, v) = \beta(v, u) \).
Let $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, E_2, \alpha_2, \beta_2)$ be graphs: $g_1$ is a subgraph of $g_2$, $g_1 \subseteq g_2$, if

- $V_1 \subseteq V_2$
- $E_1 = E_2 \cap (V_1 \times V_1)$
- $\alpha_1(v) = \{ \alpha_2(v), \text{if } v \in V_1 \\ \text{undefined, otherwise} $ \)
- $\beta_1(e) = \{ \beta_2(e), \text{if } e \in E_1 \\ \text{undefined, otherwise} $ \)

In the literature, the second condition is sometimes replaced by the weaker condition $E_1 \subseteq E_2$, and graphs that satisfy the stronger definition given above are called induced subgraphs. Most of the algorithms considered in the following can be adapted to either case. However, the definition given above is more useful in many pattern recognition applications.

Let $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, E_2, \alpha_2, \beta_2)$ be two graphs. A bijective function $f : V_1 \to V_2$ is called graph isomorphism from $g_1$ to $g_2$ if

- $\alpha_1(u) = \alpha_2(f(u))$ for all $u \in V_1$
- for every 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 every edge $e_2 = (x, y) \in E_2$ there exists an edge $e_1 = (f^{-1}(x), f^{-1}(y))$ with $\beta_1(e_1) = \beta_2(e_2)$.

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 from $g_1$ to $g$.

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 graph $g = (V, E, \alpha, \beta)$ is a common subgraph of $g_1$ and $g_2$ if there exist subgraph isomorphisms $f_1 : V \to V_1$ and $f_2 : V \to V_2$. A common subgraph $g$ of $g_1$ and $g_2$ is called a maximum common subgraph, if $g$ is
a common subgraph of \( g_1 \) and \( g_2 \), and there exists no other common subgraph of \( g_1 \) and \( g_2 \) with more nodes than \( g \).

Note that the maximum common subgraph of two graphs is not necessarily unique.

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 graph \( g = (V, E, \alpha, \beta) \) is a common supergraph of \( g_1 \) and \( g_2 \) if there exist subgraph isomorphisms \( f_1 : V_1 \rightarrow V \) and \( f_2 : V_2 \rightarrow V \). A common supergraph \( g \) of \( g_1 \) and \( g_2 \) is called minimum common supergraph if \( g \) is a common supergraph of \( g_1 \) and \( g_2 \) and there exists no other common supergraph with fewer nodes than \( g \).

Note that the minimum common supergraph of two graphs is not necessarily unique.
Example 2.1.1: Given are graphs $g_1$ and $g_2$ from Fig. 2.1

- $V_1 = \{1, 2, 3, 4\}, V_2 = \{5, 6, 7, 8\}$, a.s.o.
- $E_1 = \{(1, 2), (1, 3), (1, 4), (2, 4)\}, E_2 = \{(5, 6), (5, 7), (5, 8), (8, 7)\}$, a.s.o.
- $\alpha_1 : 1 \rightarrow A, 2 \rightarrow A, 3 \rightarrow C, 4 \rightarrow B$
- $\alpha_2 : 5 \rightarrow A, 6 \rightarrow C, 7 \rightarrow B, 8 \rightarrow A$
- $\beta_1, \beta_2$ are constant, i.e. there are no edge labels.

Obviously, the following statements hold true:

- there exists a graph isomorphism between $g_1$ and $g_2$
- $g_3 \subseteq g_1$
- there exists a subgraph isomorphism from $g_3$ to $g_2$
- $g_4 \not\subseteq g_1$
- $\text{mcs}(g_1, g_4) = g_5$
- $\text{mcs}(g_1, g_2) = g_1$
- $\text{MCS}(g_1, g_4) = g_6$

2.2 Basic Algorithms

Comparing two graphs with each other or computing the dissimilarity of graphs is referred to as graph matching. Algorithms for finding graph isomorphisms, subgraph isomorphisms, or graph edit distance are all instances of graph matching algorithms.
2.2.1 Graph and Subgraph Isomorphism

- Given: graphs $g_1 = (V_1, E_1, \alpha_1, \beta_1), g_2 = (V_2, E_2, \alpha_2, \beta_2)$
- Wanted: are $g_1$ and $g_2$ isomorphic?

Classical algorithm for graph and subgraph isomorphism: Tree search procedure as shown below. One can use either breadth first, depth first, best first, or some other strategy for traversal of the search tree (Fig. 2.2).
In order to recognize "dead ends" in the search, the following condition can be used; see Fig. 2.3.

**Structure preservation condition**

\[
\begin{align*}
\text{if} & \quad f(u_1) = u_2 \\
\text{then} & \quad \text{Each node } v_1 \in V_1 \text{ with } (u_1, v_1) \in E_1 \text{ can be mapped only to a node } v_2 \in V_2 \text{ with } (u_2, v_2) \in E_2 \text{ and vice versa; furthermore it is required that } \alpha_1(v_1) = \alpha_2(v_2) \text{ and } \beta_1((u_1, v_1)) = \beta_2((u_2, v_2)); \text{ the analog condition must hold for } (u_1, v_1) \notin E_1, (v_1, u_1) \in E_1 \text{ and } (v_1, u_1) \notin E_1.
\end{align*}
\]

If all nodes of \( g_1 \) can be mapped to nodes of \( g_2 \) without violating the structure preservation condition, then

- if \( |V_1| < |V_2| \) then \( g_1 \subseteq g_2 \)
- if \( |V_1| = |V_2| \) then \( g_1 \) and \( g_2 \) are isomorphic

In general, there may exist more than one graph- or subgraph isomorphism for two given graphs \( g_1 \) and \( g_2 \). If the task is to find only one, the search can be terminated after the first one has been found. Otherwise, if one wants to find all, the search has to go on until the complete search tree has been processed.

The time and space complexity of the tree search procedure is \( O(nm) \), where \( |V_1| = n \) and \( |V_2| = m \). It is known that the subgraph isomorphism problem is NP-complete. For the isomorphism problem it is not
yet known whether it belongs to NP or P. Anyway, all available (optimal) algorithms for both isomorphism and subgraph isomorphism are of exponential complexity.

In order to speed up the computation, a heuristic look-ahead technique can be applied. This technique is based on the structure preservation condition. It uses a table, the so-called future match table $F(u, v)$. This table has $n$ rows and $m$ columns, where

- each row represents a node in $g_1$
- each column represents a node in $g_2$

Initialization of $F(u, v)$:

- $F(u, v) = \begin{cases} 
1 & \text{if } \alpha_1(u) = \alpha_2(v) \text{ and } \in/-\text{outdegree}(u) \leq \in/-\text{outdegree}(v) \\
0 & \text{otherwise} 
\end{cases}$

$\in/-\text{outdegree}(u)$ is the number of incoming/outgoing edges of a node $u$. An entry $F(u, v) = 1$ means that $v \in V_2$ is a potential candidate to which $u \in V_1$ can be mapped, i.e. $f(u) = v$ is possible. If $F(u, v) = 0$ then $u$ can not be mapped to $v$.

In the tree search procedure, if we want to map $u \in V_1$ to a node in $V_2$, we need only consider nodes $v \in V_2$ with $F(u, v) = 1$. Once the mapping $f(u) = v$ has been established, $F(u, v)$ is updated according to the following procedure:

1. Copy $F(u, v)$ from the predecessor node in the search tree
2. Set all entries in the row corresponding to $u$ to 0, except for $F(u, v) = 1$
3. Set all entries in the column corresponding to $v$ to 0, except for $F(u, v) = 1$
4. for all nodes \( x \in V_1 \) that have not yet been mapped to \( V_2 \)
   for all nodes \( y \in V_2 \) with \( F(x, y) = 1 \)
     if structure preservation condition is violated
     then \( F(x, y) = 0 \)
     if there exists a complete row in \( F(u, v) \),
        containing only 0’s
     then BACKTRACK
   endfor
endfor

We can additionally speed up the method by always choosing the node \( u \in V_1 \) with the fewest non-zero entries in \( F(u, v) \) to be mapped to \( V_2 \) next.

Example 2.2.1: See Figures 2.4 and 2.5. The complete search tree becomes linear when the node with the fewest entries is always selected first. \( \square \)

2.2.2 Maximum Common Subgraph

- Given: graphs \( g_1 = (V_1, E_1, \alpha_1, \beta_1), g_2 = (V_2, E_2, \alpha_2, \beta_2) \)
- Wanted: \( \text{mcs}(g_1,g_2) \), the maximum common subgraph of \( g_1 \) and \( g_2 \)

This problem can be solved by means of a search procedure, similarly to Section 2.2.1. There exists another solution based on the so-called association graph of \( g_1 \) and \( g_2 \). The association graph of \( g_1 \) and \( g_2 \) is an undirected unlabelled graph \( G = (V, E) \) defined as follows

- \( V = \{(u_1, u_2) \mid u_1 \in V_1 \land u_2 \in V_2 \land \alpha_1(u_1) = \alpha_2(u_2)\} \)
- \( E \): There exists an edge between nodes \( (u_1, u_2) \) and \( (v_1, v_2) \) if the mapping \( f(u_1) = u_2, f(v_1) = v_2 \) satisfies the structure preservation condition.
This means that the set of nodes of the association graph is the Cartesian product of the nodes of $g_1$ and $g_2$, considering only pairs of nodes with identical labels. There exists an edge between two nodes, $(u_1, u_2)$ and $(v_1, v_2)$ iff nodes $u_1, v_1 \in V_1$ and $u_2, v_2 \in V_2$ have the same relation (i.e. edge or no-edge).

**Example 2.2.2:** See Figures 2.6 and 2.7.

Each node $(u_1, u_2)$ in the association graph can be understood as a mapping $f(u_1) = u_2$. If two nodes $(u_1, u_2)$ and $(v_1, v_2)$ in the association graph are connected by an edge, both of the corresponding...
mappings satisfy the *structure preservation condition*. It follows that a completely connected subset of nodes in the association graph represents an isomorphism between a subgraph $g_1 \subseteq g$ and a subgraph $g_2 \subseteq g$. Therefore, each maximal connected subset of nodes in the association graph represents a maximum common subgraph of $g_1$ and $g_2$.

A subset of completely connected nodes is also called a *clique*. Hence, the algorithm introduced above searches for maximal cliques in an association graph. Maximum clique finding is a well-known problem, for which various solutions exist.
In the example given above, there exist two maximal cliques, each of size three. They represent the two maximum common subgraphs.

The maximum clique method can also be used to find graph and subgraph isomorphisms. If all nodes of $g_1$ are present in the maximal clique, a subgraph isomorphism from $g_1$ to $g_2$ has been found. If all nodes of both graphs are present, an isomorphism has been found.

The minimum common supergraph can be computed based on the maximum common subgraph. Hence, the maximum clique method can also be used for computing minimum common supergraphs.
2.3 Graph Edit Distance

We consider six different types of edit operations:

- Node deletion: \( v \rightarrow \varepsilon; \ v \in V \)
- Node insertion: \( \varepsilon \rightarrow v; \ v \in V \)
- Node substitution: \( a \rightarrow b; \ a, b \in L_V \)
- Edge deletion \( e = (u, v): \ e \rightarrow \varepsilon; \ e \in E \)
- Edge insertion \( e = (u, v): \ \varepsilon \rightarrow e; \ e \in E \)
- Edge substitution: \( c \rightarrow d; \ c, d \in L_E \)

The corresponding costs are \( c (v \rightarrow \varepsilon), \ c (\varepsilon \rightarrow v), \ c (a \rightarrow b), \ c (e \rightarrow \varepsilon), \ c (\varepsilon \rightarrow e) \) and \( c (c \rightarrow d) \).

Example 2.3.1:

- original graph:

- node deletion/insertion/substitution (plus implied operations):

- edge deletion/insertion/substitution:
The cost of a sequence of edit operations \( S = s_1, \ldots, s_n \) is defined as
\[
c(S) = \sum_{i=1}^{n} c(s_i).
\]

The edit distance \( d(g_1, g_2) \) of graphs \( g_1 \) and \( g_2 \) is defined as
\[
d(g_1, g_2) = \min \{ c(S) \mid S \text{ is a sequence of edit operations transforming } g_1 \text{ into } g_2 \}.
\]

If, during the transformation of \( g_1 \) into \( g_2 \), a node \( u \) gets deleted, also all of its incident edges need to be deleted. Otherwise, a non-valid graph is generated. Such edge deletions are called implied edge deletions. Sometimes, depending on the underlying application, a different cost is defined for implied and normal edge deletions. A similar comment applies to node/edge insertions.

**Example 2.3.2:**

If all costs are equal to 1, we have \( d(g_1, g_2) = 3 \). \( \square \)
If we define
\[ c(s) = \begin{cases} 
0 & \text{if } s \text{ is a node deletion} \\
\infty & \text{if } s \text{ is any other non-identical edit operation} 
\end{cases} \]
then
\[ d(g_1, g_2) = 0 \Leftrightarrow g_1 \supseteq g_2 \]

Here we assume that implied edge deletions have 0 cost. Consequently, we can solve the subgraph (and graph) isomorphism problem by means of edit distance. It follows that graph edit distance must have an exponential computational complexity.

The computation of \( d(g_1, g_2) \) can be accomplished by a tree search procedure, similarly to subgraph isomorphism (Section 2.2.1.); see Fig. 2.8.

There exists the possibility to speed up the search by taking future costs into account (Algorithm A*). (No details given.)
2.4 Other Graph Distance Measures

(1) \( d_1(g, g') = 1 - \frac{|mcs(g, g')|}{\max(|g|, |g'|)} \)

Properties:
- \( d_1(g, g') = 0 \) if and only if \( g \) and \( g' \) are isomorphic
- \( d_1(g, g') = 1 \) if and only if \( mcs(g, g') = \emptyset \)
- \( d_1(g, g') \in [0, 1] \)
- \( d_1(g, g') \) is a metric

(2) \( d_2(g, g') = 1 - \frac{|mcs(g, g')|}{|MCS(g, g')|} \)

- has the same properties as \( d_1 \)

(3) \( d_3(g, g') = |MCS(g, g')| - |mcs(g, g')| \)

Properties:
- \( d_3(g, g') \in [0, |g| + |g'|] \)
- \( d_3(g, g') \) is a metric

Because \( |MCS(g, g')| = |g| + |g'| - |mcs(g, g')| \), we can rewrite

\( d_3(g, g') = |g| + |g'| - 2|mcs(g, g')| \)

One can show that, with certain edit costs, \( d_3(g, g') = d(g, g') \), where \( d(g, g') \) is the edit distance.

Illustration: See Figure 2.9

Example 2.4.1: Consider graphs in Fig. 2.10.

\[
\begin{align*}
    d_1(g_1, g_4) &= 1 - \frac{2}{\frac{4}{5}} = \frac{1}{\frac{7}{5}} \\
    d_2(g_1, g_4) &= 1 - \frac{3}{\frac{6}{5}} = \frac{1}{\frac{11}{5}}
\end{align*}
\]
$d_3(g_1, g_4) = 5 - 2 = 3$

$d_1(g_1, g_2) = d_2(g_1, g_2) = d_3(g_1, g_2) = 0$

$d_1(g_1, g_3) = 1 - \frac{3}{4} = \frac{1}{4}$

$d_2(g_1, g_3) = 1 - \frac{3}{4} = \frac{1}{4}$

$d_3(g_1, g_3) = 4 - 3 = 1$

\[\square\]

### 2.5 Applications

#### 2.5.1 Character Recognition

In this application example, we consider synthetically generated patterns. The task is to recognize all capital letters of the Roman alphabet that can be written with straight line segments only ($A, E, F, \ldots, Z$). Totally, there exist 15 such letters (see Fig. 2.11). Hence we are facing a classification problem with 15 classes.
Starting with ideal prototype letters, we apply distortions:

- translation of endpoints
- deletion and insertion of lines

There exist five different degrees of distortion \((0.1, 0.3, 0.5, 0.7, 0.9)\). For each letter prototype and each degree of distortion, there exist 150 letter instances. Therefore, in total the data set consists of \(15 \times 150 \times 5 = 11'250\) letters.

Examples of distortions: Fig. 2.12

Each letter is represented by a graph as follows:

- nodes: endpoints of lines
- edges: lines
- node labels: \((x, y)\)-coordinates of endpoints; i.e. \(L_V = \mathbb{R}^2\)
Ideal prototypes:

Figure 2.11

- edge labels: no edge labels

Cost of edit operations:

- Node deletion and insertion:
  
  \[ c(u \rightarrow \epsilon) = c(\epsilon \rightarrow u) = c_{\text{node}} \]

  where \( c_{\text{node}} \) is a constant

- Edge deletion and insertion:
  
  \[ c(e \rightarrow \epsilon) = c(\epsilon \rightarrow e) = c_{\text{edge}} \]

  where \( c_{\text{edge}} \) is a constant

- Node substitution:
  
  \[ c(u_1 \rightarrow u_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \]

  This is the Euclidean distance of the two endpoints

- Edge substitution: as there are no edge labels, there are no edge substitution costs either
In a classification experiment, the database was divided into three disjoint subsets:

- training set with 3’750 graphs
- test set with 3’750 graphs
- validation set with 3’750 graphs

Each set consists of 50 graphs for each degree of distortion and each letter class, i.e. $15 \times 50 \times 5 = 3’750$ graphs.
To solve the classification task, a nearest-neighbor classifier with graph edit distance is used. The graphs from the training set are the prototypes for this classifier. The graphs from the validation set are used to optimize the parameters $c_{\text{node}}$ and $c_{\text{edge}}$. The test set is used to calculate the recognition rate.

Experimental results:

<table>
<thead>
<tr>
<th>Distortion</th>
<th>1-NN</th>
<th>3-NN</th>
<th>5-NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>97.33%</td>
<td>97.33%</td>
<td>98.00%</td>
</tr>
<tr>
<td>0.3</td>
<td>96.00%</td>
<td>96.00%</td>
<td>98.00%</td>
</tr>
<tr>
<td>0.5</td>
<td>92.66%</td>
<td>92.00%</td>
<td>93.33%</td>
</tr>
<tr>
<td>0.7</td>
<td>89.33%</td>
<td>90.66%</td>
<td>91.33%</td>
</tr>
<tr>
<td>0.9</td>
<td>82.00%</td>
<td>80.66%</td>
<td>81.33%</td>
</tr>
</tbody>
</table>

From these results one can conclude that

- the recognition rate deteriorates with an increasing degree of distortion
- increasing the number of nearest neighbors does not necessarily lead to better recognition rates (only in 6 out of 10 cases an improvement results)

### 2.5.2 Diatom Classification

Diatoms are unicellular algae that exist in water and wet places on earth. There exist probably more than 10,000 different species of diatoms. The occurrence of some of these species is tightly coupled with certain climatic and environmental conditions. Hence diatoms are valuable indicators for environment conditions and their classification is an interesting and important application.

Examples: See Figure 2.13

To classify diatoms one can use either their shape or their internal structure (texture). In the following we consider the second possibility
and describe a nearest-neighbor classifier based on graph edit distance.

First, diatom images are transformed into graphs. This transformation is accomplished by the following steps.

First image segmentation is performed:

1. Apply an edge detector that identifies vertical edges
2. Improve the edge quality by some postprocessing steps
3. Determine the direction of each edge pixel
4. The direction of the nearest edge pixel is assigned to each pixel
5. Apply a region growing procedure that finds regions where the edge direction is nearly constant
6. Do some postprocessing (elimination of small regions, etc.)

Examples see Fig. 2.14, 2.15, and 2.16
Figure 2.14

Figure 2.15

Figure 2.16
Figure 2.17

After segmentation, a diatom image is transformed into a graph as follows:

- **Nodes:** represent the regions; each node has the following attributes:
  - coordinates of the bounding box
  - relative size $A = (\text{size of region})/(\text{size of diatom})$ [in pixel]
  - average direction of the contour pixels in the region
  - density of the contour pixels (number of contour pixels relative to total number of pixels)

- **Edges:** represent the adjacency of regions (details are left out)

Cost of edit operations:

- **Node deletion and insertion:** proportional to size of region
- **Node substitution:** the differences in the node attributes are computed, weighted, and summed up
- **Edge insertion, deletion, and substitution:** details are left out

**Database:** 110 diatom images from 22 classes, i.e. 5 diatoms per class (rather small!)
Classification experiment with 1-nearest-neighbor classifier and leave-one-out protocol.

Recognition results: Fig. 2.17

Confusion matrix: See Fig. 2.18

2.5.3 Classification of Web Documents

The automatic classification of web documents is an actual research topic. We consider the task of content based classification (in contrast with layout based classification).

Traditionally, a web document is represented by a feature vector where the features correspond to words in an underlying dictionary. There are several ways of representation, for example

- binary: a word either occurs in the document or not
- integer: we count the number of occurrences of a word

![Confusion Matrix](image)
real: word counts are weighted with the importance of a word

Example: See Fig. 2.19

Using such a vector representation, any classifier from statistical pattern recognition can be applied.

In the following we discuss a solution based on representing a web document by a graph. We assume that we have a program at our disposal that transform HTML into ASCII format.

Before the graph of a web document is generated, the following pre-processing steps are applied:

- Elimination of stop words: Stop words are frequently occurring words without semantic meaning, for example, *and*, *or*, *the*, *of*,...  
  Stop words are removed based on a list of about 600 frequent stop words.
Figure 2.20

- **Stemming**: Different versions of the same word (singular-plural, present tense - past tense, etc.) are replaced by one standard representation. There exist standard algorithms for stemming.

After preprocessing the graph representation of a document is generated.

- **Nodes**: Each word is represented by a node. If there are multiple occurrences of the same word, only one node is generated. Each node is labeled with the corresponding word.

- **Edges**: If word $y$ immediately follows word $x$, an edge from node $x$ to node $y$ is inserted.

Several variations of this graph representation have been investigated. For example, one may consider neighborhoods larger than one for the edges, and one can distinguish between normal text, hyperlinks, and meta information (details omitted).

Example: See Fig. 2.20

A nearest-neighbor classifier under the leave-one-out strategy is used. This classifier is based on the distance function $d_1$ from Section 2.4. However, we use $|g| = |V| + |E|$ instead of $|g| = |V|$. In this way, more emphasis is put on the edges.

Experimental results: Fig. 2.21
The underlying data set consists of 185 web pages from 10 classes. Better results are obtained with the graph representation, although a much smaller number of words than in the vector representation is used.

2.6 Other Algorithms

In many pattern recognition applications, the objective is to compare an unknown input graph to a number of given graphs (called database graphs in the following). In such a scenario, the high complexity of graph matching is even increased by a factor proportional to the size of the database. However, as the database in known beforehand, one can apply preprocessing steps. These operations potentially lead to a decrease of the complexity at run time. We consider two such preprocessing methods.

2.6.1 Network based Approach

Each DB graph is recursively decomposed into two disjoint parts. Identical parts that belong to two different graphs are stored only once. The resulting data structure is called a network. An unknown
input graph is matched against the network. If certain parts in the network represent subgraphs that occur multiple times in the database, they are matched only once against the input graph. Hence, some computation time can be saved.

The method is explained by means of an example (Fig. 2.22). Graphs $g_1$ and $g_2$ are the DB graphs, while $g_3$ is the unknown input. The network is compiled off-line from $g_1$ and $g_2$. (The network is not uniquely defined, but this is not a disadvantage.) The highest level in the network includes all individual nodes that occur in the DB graphs. The complexity of subgraphs grows as we go to lower levels in the network. At the lowest levels, we have the complete DB graphs represented.

In this example we assume that we want to find out whether one of the DB graphs is contained, as a subgraph, in the input graph $g$. We input the individual nodes of $g$, one after the other, into the network. First the network nodes at the highest level are activated. These
network nodes check the node of the input graph for identical node label. If the node label is identical, the node is stored in a local memory. Afterwards, network nodes at lower levels are activated. They have two input channels and check whether they can combine two subgraphs, one from each input channel, into larger structures. Each larger structure obtained in this way is stored in the local memory of the considered network node. In the end, it is checked whether the local memories of the network nodes at the lowest levels contain any graphs. If this is the case, we know that the corresponding DB graphs are subgraphs of the input graph.

Complexity:

- Worst case: all graphs are disjoint and we get the same complexity as checking each graph separately
- Best case: all graphs in the DB are identical and the complexity is independent of the number of graphs in the DB

Experimental investigation: See Fig. 2.23

For this experiment randomly generated graphs were used. The lower plane is the computation time of the network approach, while the
upper plane corresponds to the conventional approach where all DB graphs are tested sequentially.

2.6.2 Decision Tree based Method

This method uses an even more intensive preprocessing of the DB graphs. Graphs are represented by their adjacency matrix. Let $g = (V, E, \alpha, \beta)$ be a graph with $V = \{v_1, ..., v_n\}$. Then the adjacency matrix of $g$ is an $n \times n$ matrix

$$M = [m_{ij}],$$

where

$$m_{ij} = \begin{cases} \alpha(v_i), & \text{if } i = j \\ \beta(v_i, v_j), & \text{otherwise} \end{cases}$$

If graph $g$ is given by its adjacency matrix, then any permutation of the adjacency matrix is also a valid representation of $g$. In other words, all permutations of an adjacency matrix represent the same graph.

A permutation matrix $P$ is a $n \times n$ matrix with the following properties:

- all elements of $P$ are either 0 or 1
- each row includes exactly one 1
- each column includes exactly one 1

If graph $g$ is given by an $n \times n$ adjacency matrix $M$ and $P$ is an $n \times n$ permutation matrix, then

$$M' = PMP^T$$

is also an adjacency matrix of $g$. ($P^T$ is the transpose of $P$.) If $p_{ij} = 1$, then the $j$-th node in $M$ becomes the $i$-th node in $M'$.

Hence the question whether $g_1$ and $g_2$ are isomorphic is equivalent to the question whether a permutation matrix $P$ exists, such that

$$M_2 = PM_1P^T.$$
The existence of a subgraph isomorphism from $g_1$ to $g_2$ is equivalent to the existence of a permutation matrix $P$, such that the left upper partial matrix of $PM_2P^T$ is identical to $M_1$. $M_1$ and $M_2$ are the adjacency matrices of $g_1$ and $g_2$, respectively.

In the decision tree based method, we generate offline all permutations of the adjacency matrix of the first database graph. We split up each adjacency matrix into so-called row-column elements. All row-column elements are arranged in a decision tree (Fig. 2.24).
We proceed analogously for all other graphs in the DB, using the already existing decision tree (Fig. 2.25).

Some of the decision tree nodes are redundant and can be eliminated (Fig. 2.26).

During run time, we represent an input graph by its row-column elements and compare it with the entries in the decision tree. There are three cases that can happen:

- The input graph has been completely processed and a leave node in the decision tree has been reached. In this case we know that the input graph is isomorphic to the DB graph that is represented
by the leave node.

- The input graph has been completely processed and a non-leave node $u$ in the decision tree has been reached. In this case we know that there exists a subgraph isomorphism from the input graph to all graphs in the DB that are represented by successors of $u$.

- The input graph has not been completely processed and there is no successor node in the search tree. In this case we know that there exists neither a subgraph nor a graph isomorphism from the input graph to any of the database graphs.

Complexity: Let $L$ be the number of graphs in the DB, $n$ be the number of nodes in the input graph, and $m$ be the number of nodes of the largest DB graph. Then

- Time complexity is $O(n^2)$
- Space complexity is $O(Lm^m)$

Note that the time complexity is completely independent of the number of graphs in the DB and their size. However, the space complexity is a limiting factor of this method. If there is one graph in the DB, graphs up to about 20 nodes can be handled. If there are, for example, 30 graph in the DB, each can have up to about 10 nodes.

Experimental investigation: Fig. 2.27

For this experiment randomly generated graphs were used again.
2.7 The Median Graph Problem

Let $U$ be the set of all graphs that can be constructed using labels from $L_V$ and $L_E$, and let $S = \{g_1, \ldots, g_n\} \subset U$.

The set median graph of $S$ is defined as

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

It is a graph from $S$ that minimizes the sum of distances to all members of $S$. The generalized median graph, or median graph, for short, of $S$ is defined as

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

It is a graph from $U$, not necessarily from $S$, that minimizes the sum of distances to all members of $S$. Note that neither the set nor the generalized median graph is necessarily unique for a given set of graphs $S$.

The median or set median graph is a useful concept to represent a whole set of graphs by just a single element. It has applications in clustering, prototype reduction for nearest-neighbor classification, and other tasks. The median is a more general concept than the set median. The set median can be computed in $O(n^2 \delta)$, where $\delta$ is the complexity of the distance computation. The complexity of the median, however, is exponential in $n$. Its computation is a real challenging task (which is only partially solved today).

Note that the median $m$ of a set of numbers $x_1, \ldots, x_n$ has the same property as the median of a set of graphs: it minimizes the sum of distances $\sum_{i=1}^{n} |x_i - m|$.

Following are a few bounds of the median listed. They are potentially useful when designing an algorithm for median graph computation:

1. $0 \leq |\bar{g}| \leq \sum_{i=1}^{n} |g_i|$
2. \( SOD(g) \leq \min(SOD(\epsilon), SOD(MCS(g_1, ..., g_n))) \), where \( \epsilon \) denotes the empty graph.

3. \( SOD(g) \geq d(g_1, g_2) + d(g_3, g_4) + ... + d(g_{n-1}, g_n) \) if \( n \) is even and \( d(g, g') \) is a metric.

A formula similar to 3 can be derived for the case where \( n \) is odd. Note that in formula 3, the order of \( g_1, ..., g_n \) is arbitrary.

A number of optimal and suboptimal algorithms for median graph computation have been proposed. We briefly sketch a genetic algorithm in the following.

**Chromosome representation:** In order to represent a median graph candidate by a chromosome, one can use a linearization of the adjacency matrix (See Fig. 2.28).

However, such a representation results in a high computational complexity, because the \( SOD \) to all members of set \( S \) has to be computed for each chromosome, i.e. for each candidate. For this reason a more efficient solution has been proposed where there is one position in the chromosome for each node of each of the graphs in \( S \). The position holds the number of the corresponding node in the median graph candidate (See Fig. 2.29).

**Median graph candidate representation:** A median graph candidate is only implicitly represented by a chromosome. However, it can
be fully reconstructed from the corresponding chromosome, including its edges, node labels, and edge labels.

**Genetic operators:** The usual genetic operators mutation and crossover are applied. It can happen that an inconsistent chromosome is generated. In this case a repair operation is carried out. This repair operation consists in randomly changing an inconsistent position until it becomes consistent (see Fig. 2.30).

**Fitness function:** The fitness function is the $SOD$.

Experimental results:
The results shown in Fig. 2.31 are based on randomly generated graphs (with 6 nodes only).

The middle curve is the $SOD$ of the computed median graph, while the upper and lower curves show the upper and lower bounds derived
above (equations 1 to 3).

**An application: median words**

Distorted instances of words are generated by randomly changing the individual letters at each position. Each word is represented by a graph where the nodes represent the letters, and edges connect two adjacent letters (See Fig. 2.32).

It can be seen that the median usually has a lower $SOD$ than the set median. Hence it is a more accurate representative. The method of this experiment can be used to combine the results of different OCR systems to make the final results more precise.
### 2.8 Suboptimal Graph Edit Distance Computation

The high computational complexity of graph edit distance motivates the study of suboptimal algorithms that may return approximate solutions only, but run in polynomial time. There exist several possibilities to design such algorithms. In the following we introduce a method based on the assignment problem.

Assignment problem: Let $A$ and $B$ be two finite sets with $|A| = |B| = n$. Furthermore, let $C = [c_{ij}]$ be an $n \times n$ matrix, where $c_{ij} \geq 0$ denotes the cost of assigning $b_j$ to $a_i$. The assignment problem consists in finding a permutation $p_1, ..., p_n$ of the integers $1, ..., n$ that minimizes $\sum_{i=1}^{n} c_{ip_i}$.

The assignment problem can be solved by a well-known algorithm (by Kuhn-Munkres). This algorithm is also known as the Hungarian algorithm, and the assignment problem is sometimes referred to as bipartite graph matching (See Fig. 2.33). The algorithm runs in $O(n^3)$

<table>
<thead>
<tr>
<th>Original word $w$</th>
<th>graph $(1.85)$</th>
<th>matching $(2.25)$</th>
<th>median $(2.1)$</th>
<th>genetic $(2)$</th>
<th>search $(1.45)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input words</td>
<td>caph</td>
<td>matdhdgd</td>
<td>mbblbmbn</td>
<td>genatbb</td>
<td>scdbdh</td>
</tr>
<tr>
<td></td>
<td>crchb</td>
<td>dauchdg</td>
<td>median</td>
<td>gbanaib</td>
<td>sedbhh</td>
</tr>
<tr>
<td></td>
<td>gcph</td>
<td>mbtedig</td>
<td>bciab</td>
<td>gebetic</td>
<td>seabdh</td>
</tr>
<tr>
<td></td>
<td>gbaph</td>
<td>matdcdig</td>
<td>cedcanb</td>
<td>ganatbc</td>
<td>sardh</td>
</tr>
<tr>
<td></td>
<td>grphb</td>
<td>matdlnndd</td>
<td>cedian</td>
<td>genaabb</td>
<td>scdbdh</td>
</tr>
<tr>
<td></td>
<td>grbhb</td>
<td>ddceding</td>
<td>medici</td>
<td>gennaica</td>
<td>dardh</td>
</tr>
<tr>
<td></td>
<td>gcahcb</td>
<td>mbchnbg</td>
<td>mbdiab</td>
<td>genetbb</td>
<td>scdbdh</td>
</tr>
<tr>
<td></td>
<td>grch</td>
<td>mbchnbg</td>
<td>mbdiab</td>
<td>genetbb</td>
<td>scdbdh</td>
</tr>
<tr>
<td></td>
<td>craphh</td>
<td>matchin</td>
<td>cbdiab</td>
<td>enbtiic</td>
<td>bddrdeh</td>
</tr>
<tr>
<td></td>
<td>grabh</td>
<td>mbddbbbg</td>
<td>mcrcan</td>
<td>gebetica</td>
<td>dcarch</td>
</tr>
<tr>
<td>Set median $\hat{S}$</td>
<td>grachh $(2)$</td>
<td>matdcding $(2.4)$</td>
<td>cedian $(2.2)$</td>
<td>genatbb $(2)$</td>
<td>scdbdh $(1.4)$</td>
</tr>
<tr>
<td>Computed generalized median $\overline{S}$</td>
<td>graph $(1.85)$</td>
<td>matching $(2.25)$</td>
<td>mbddb $(2.05)$</td>
<td>genatbb $(1.9)$</td>
<td>scdbdh $(1.4)$</td>
</tr>
</tbody>
</table>
time. (Details are left out.) The algorithm is optimal, i.e. it always returns the optimal solution. Note, however, that there may exist more than one optimal solution.

Consider graphs $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, E_2, \alpha_2, \beta_2)$ with $V_1 = \{u_1, ..., u_n\}$ and $V_2 = \{v_1, ..., v_n\}$, i.e. we assume that $|V_1| = |V_2|$. A simple suboptimal solution to the graph edit distance problem consists in defining

$$c_{ij} = c(\alpha_1(u_i) \rightarrow \alpha_2(v_j)),$$

i.e. the element $c_{ij}$ in the cost matrix $C$ is defined as the cost of substituting the label of node $u_i$ in $g_1$ with the label of $v_j$ in $g_2$. Then we set $A = V_1$ and $B = V_2$ and run the Hungarian algorithm. It will give us the following assignment:

$$u_1 \rightarrow v_{p_1}, u_2 \rightarrow v_{p_2}, ..., u_n \rightarrow v_{p_n}$$

Given this node assignment we can compute the edit distance $d(g_1, g_2)$ by summing up the rest of all node substitutions and all edit operations on the edges implied by the node assignment. The assignment returned by the Hungarian algorithm is optimal w.r.t. cost matrix $C$. However, it is only suboptimal w.r.t. the graph edit distance, because all edit operation on the edges have been neglected when determining the optimal assignment, and only node substitutions (but no node deletions and insertions) have been considered. Hence the
edit distance obtained through the Hungarian algorithm is equal to, or larger, than the true edit distance.

A more general solution is obtained if we assume \( V_1 = \{u_1, ..., u_n\} \) and \( V_2 = \{v_1, ..., v_m\} \) and define the cost matrix \( C \) as an \((n+m) \times (n+m)\) matrix (Fig. 2.34)

![Figure 2.34](image)

Obviously, the left upper corner of the cost matrix represents the costs of all possible node substitutions, the diagonal of the right upper corner the costs of all possible node deletions, and the diagonal of the bottom left corner the costs of all possible node insertions. Note that each node can be deleted or inserted at most once. Therefore any non-diagonal element of the right-upper and left-lower part is set to \( \infty \). The bottom right corner of the cost matrix is set to zero since substitutions of the form \((\epsilon \rightarrow \epsilon)\) should not cause any costs.

With this extended cost matrix one can run the Hungarian algorithm. It will return us an assignment where also node deletions and insertions are possible. Moreover it is not required any longer that \( g_1 \) and \( g_2 \) have the same number of nodes. However, edit operations on the edges are still neglected when computing the optimal node assignment.
A third, more refined version is obtained by taking edit operation on the edges in the cost matrix into account. For this purpose a lower bound estimate of the cost of the edge edit operations adjacent to each node are added to each entry $c_{ij}$ in the cost matrix.

In a number of experiments it has been shown that the execution of graph edit distance can be substantially sped up without losing accuracy in a nearest-neighbor classifier (see Fig. 2.35).

### 2.9 Graphs with Unique Node Labels

One way to deal with the high computational complexity of graph matching is to resort to suboptimal algorithms. Another is to restrict the focus to special classes of graphs. There are in fact several special classes of graphs for which the matching problem can be solved in polynomial time. In this section we will consider the class of graphs with unique node labels.

A graph with unique node labels (gun) is a graph $g = (V, E, \alpha, \beta)$ where for each pair of nodes $u, v \in V$ the following condition holds true:

- If $u \neq v$ then $\alpha(u) \neq \alpha(v)$

It turns out that for a gun nodes and node labels are equivalent. In the following we use the so-called label representation to represent a

<table>
<thead>
<tr>
<th>Database</th>
<th>HEURISTIC-A*</th>
<th>BP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Letter (L)</td>
<td>Acc 91.0 1.00 649'05&quot;</td>
<td>Acc 91.1 0.98 7'52&quot;</td>
</tr>
<tr>
<td>Letter (M)</td>
<td>77.9 1.00 2061'29&quot;</td>
<td>77.6 0.93 6'11&quot;</td>
</tr>
<tr>
<td>Letter (H)</td>
<td>63.0 1.00 4914'45&quot;</td>
<td>61.6 0.97 8'48&quot;</td>
</tr>
</tbody>
</table>

Figure 2.35
Let $g = (V, E, \alpha, \beta)$ be a gun. The label representation of $g$ is given by $l(g) = (L, C, \lambda)$, where

- $L = \{\alpha(x) | x \in V\}$
- $C = \{(\alpha(x), \alpha(y)) | (x, y) \in E\}$
- $\lambda : C \rightarrow L_E$ with $\lambda(\alpha(x), \lambda(y)) = \beta(x, y)$ for all $(x, y) \in E$.

This simply means that we replace each node by its label.

**Properties:** Given two graphs in terms of their label representations $g_1 = (L_1, C_1, \lambda_1)$ and $g_2 = (L_2, C_2, \lambda_2)$

- Graphs $g_1$ and $g_2$ are isomorphic if and only if their label representations are identical, i.e. $L_1 = L_2, C_1 = C_2, \lambda_1 = \lambda_2$.

- Graph $g_1$ is a subgraph of $g_2$ if and only if $L_1 \subset L_2, C_1 \subset C_2$, and $\lambda_1(i, j) = \lambda_2(i, j)$ for all $(i, j) \in C_1$.

- The maximum common subgraph is given by $mcs(g_1, g_2) = (L, C, \lambda)$ where $L = L_1 \cap L_2, E = C_1 \cap C_2, \lambda(i, j) = \lambda_1(i, j)$ for all $(i, j) \in C$.

- The edit distance can be computed in a straightforward way based on $|L_1|, |L_2|, |C_1|, \text{ and } |C_2|$.

All these algorithms run in time that is quadratic in the number of nodes of the underlying graphs.

Also the median of a set of graphs can be computed efficiently. Its complexity is quadratic in the number of nodes in the graphs and linear in the number of graphs in the set.

Computational experiments with randomly generated graphs: See Figure 2.36

Although the class of graphs with unique node labels seems somewhat restricted at first glance, there are applications from the real world where graphs of this kind naturally occur. Examples include
2.10 Hypergraphs

Graphs are a flexible and powerful representation formalism when compared to feature vectors. Nevertheless, there is one limitation arising from the fact that only binary relationships between nodes, but no relations of higher order can be represented. Hypergraphs are a generalization of graphs that provide us with the possibility to overcome this limitation. They allow us to model relations of any order $n \geq 1$ by means of hyperedges. Ordinary graphs are a special case of hypergraphs.

We will not give any formal definitions here, but just mention that a hyperedge $e$ of order $n$ is an $n$-tupel of nodes, i.e. $e(v_1, ..., v_n)$ element $V^n$.

There are several possibilities to graphically represent hypergraphs. One is to represent hyperedges of order $n \geq 3$ by closed curves that include the participating nodes (see Fig 2.37).

Another possible graphical representation of a hypergraph consists in choosing rectangles for the nodes and circles or ellipses for the hyperedges. Arrows point to the nodes that belong to a hyperedge. The arrows can be unlabelled (undirected hypergraph) or labeled with

<table>
<thead>
<tr>
<th></th>
<th>Computational Times in seconds for graphs with $N$ vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$N=100$</td>
</tr>
<tr>
<td>Fan Chung 2.5%</td>
<td>0.01</td>
</tr>
<tr>
<td>Random 2.5%</td>
<td>0.02</td>
</tr>
<tr>
<td>Random 10%</td>
<td>0.02</td>
</tr>
</tbody>
</table>
Next we give an example from image processing. We show an image, its region adjacency graph, and a hypergraph that includes the information that three or more regions touch each other in a common corner (see Fig. 2.39).

The next example is from computer aided design and shows the wireframe model of a 3-D object together with its graph representation and a hypergraph that includes the relation of collinearity (see Fig. 2.40).

The last example is from social network analysis where we model a group of musicians who play together in different formations (see Fig. 2.41).

It can be shown that the concepts of graph isomorphism, subgraph
isomorphism, maximum common subgraph, minimum common supergraph, median graph, etc. can be generalized from graphs to hypergraphs. Moreover, existing algorithms can be suitably extended. However, the price to be paid for this higher flexibility is an increased computational complexity.

In an experiment it was shown that hypergraphs can improve the performance of a nearest-neighbor classifier when compared to the use of ordinary graphs. The data set used in this experiment is similar to the one used in Section 2.5.1. In the normal graph representation there is no direct information included whether or not two line segments cross each other. There is no straightforward way to include this kind of information in a normal graph, although it can be intuitively expected that it would be useful for classification. However, in a hypergraph it is easy to include this information. One just needs to define a special type of hyperedge of order four (see Fig. 2.42).
With this extended representation the following recognition rates were achieved (see Fig. 2.43).

For strong distortions, the additional relation turns out adverse. This is due to the fact that in many cases this relation gets deleted, and new instances of this relation may be erroneously created.

### 2.11 Information Retrieval with Graphs

In this section we consider the task of information retrieval using graph based representations. For this task the previous concepts need to be extended. In particular we allow

- Multiple edges between the same two nodes
Nodes and edges to have a type; the type defines the particular label (i.e. vector of attributes) that can be assigned to a node.

Example: See Figure 2.44

All concepts defined for graphs can be suitably extended to this kind of graph. However, in the context of information retrieval we are particularly interested in subgraph isomorphism.

The graph given above can be interpreted as a database graph that includes information about a particular domain. For the purpose of information retrieval, the user can present a smaller graph, a so-called...
query graph, to the system and the system will try to find occurrences of the query graph in the database graph (see Fig. 2.45).

However, this kind of information retrieval is very restricted as it allows only binary answers *yes* or *no* to be returned in response to a query.

In order to make the retrieval system more flexible, a number of additional concepts are introduced:

- Variables; see Fig. 2.46 a)
- Don’t cares; ; see Figures 2.46 b) - 2.46 d)
- Constraints; see Fig. 2.46 e)

In principle, such an information retrieval system can be built based on a relational database system as well. However, using a graph based
formalism allows us to include error-tolerance in the retrieval process (allowing, for example, up to $n$ errors in a match). Also, using graphs a number of tools from artificial intelligence, data mining, pattern recognition a.s.o. become available (see also Section 3ff).

### 2.12 Graph Databases for Experimental Work

The objective comparison of different algorithms and methods is an important issue in pattern recognition. For this purpose, databases are needed that can be publicly accessed. In the ideal case, a database should not only include the data, but should be divided into

- a training set
- a test set
- a validation set

For statistical pattern recognition, several such databases exist. A well known example is the *UCI Machine Learning Repository*: 

![Diagram](image-url)
Figure 2.46

(a) (person, (Adam))

(b) (person, (X))

(c) (person, (Adam))

(d) (person, (X))

(e) $U = R$

$V = S + 1h$

$W = T$

55
For graph based pattern recognition, the **TC-15 database** is available:

http://amalfi.dis.unina.it/graph/db/vflib-2.0/

However, this database includes only synthetically generated graphs. Also it is restricted to exact graph matching tasks (subgraph isomorphism, graph isomorphism).

Recently the **IAM Graph Database Repository**, a more general database, has been made publicly available:

www.iam.unibe.ch/fki/databases/iam-graph-database/

It consists of ten datasets and can be used to test graph classification and clustering algorithms. Each data set is divided into the three sets mentioned above. Furthermore, the performance of a simple 1-nearest-neighbor classifier based on graph edit distance is given for each data set.
Figures 2.47 and 2.48 give a summary of the graph data set characteristics.

Following are more details about the individual data sets.

1 - 3. **Letter datasets (low, medium, high)**

These data sets have been described in Section 2.5.1.

4. **Digits**

This data set includes real on-line digits from ten classes 0, 1, ..., 9 (see Figure 2.49). Originally, each digit is given by a sequence of coordinates \((x_1, y_1), \ldots, (x_n, y_n)\), representing the location of the tip of the pen as a function of time. To transform the original data into a graph, a new sequence \((X_1, Y_1), \ldots, (X_m, Y_m)\) is generated by subsampling, where \(m < n\). This new sequence is then transformed into a graph.
• Nodes: new points \((X_i, Y_i)\)
• Edges: connecting lines between two consecutive point \((X_i, Y_i)\) and \((X_{i+1}, Y_{i+1})\)
• Node attributes: \((x, y)\)-coordinates
• Edge attributes: angle of an edge relative to the \(x\)-axis

5. GREC database

This data set includes graphical symbols from architecture and electronic engineering (22 classes; see Fig. 2.50). There exist different levels of distortion (added noise). Morphological operations are applied to an image, and lines are thinned to one pixel width. Then corners, crossings, endpoints and circles are detected. The graph representation is as follows.

• Nodes: end points, corners, and crossings
• Edges: connecting lines
• Node attributes: \((x, y)\)-coordinates
• Edge attributes: line type and line specific parameters (angle, diameter)

6. Fingerprint database

This data set includes images from four different classes (Fig. 2.51). Filtering, binarization, thinning, and noise reduction procedures are applied to transform images into graphs.
• Nodes: ending points, bifurcation points, interior points of ridges
• Edges: ridges connecting two nodes
• Node attributes: \((x, y)\)-coordinates
• Edge attributes: angle of an edge relative to the \(x\)-axis

7. COIL-RAG database

This database consists of images of 100 different objects, obtained under different viewing angles (Fig. 2.52). The images are first segmented into regions of uniform color. From the segmented image a graph is generated.
• Nodes: regions
• Edges: adjacency of regions
• Node attributes: color histogram
• Edge attributes: length of the common border of two regions (in pixel)

8. COIL-DEL database

The images are the same as in the COIL-RAG database. Only the graph extraction process is different. Instead of region segmentation, an interest operator is applied (Harris corner detector). Given the points of interest, a Delaunay triangulation is performed.

• Nodes: points of interest
• Edges: edges of the Delaunay triangulation
• Node attributes: \((x, y)\)-coordinates
• Edge attributes: none

9. Web database

This data set has been described in Section 2.5.3.

10. AIDS database

Graphs represent molecules from two different classes: active and inactive w.r.t. the HIV virus (see Fig.2.53).
Figure 2.54

- Nodes: atoms
- Edges: chemical bonds
- Node attributes: number of chemical element
- Edge attribute: value of bond

11. Mutagenicity database:

This database is similar to the Aids database. There are two classes, *mutagen* and *non-mutagen*.

12. Protein database

The graphs in this database represent proteins from six different classes (Fig. 2.54).