Graphs in Pattern Recognition

a one hour trip in the history

Prof. Mario Vento
University of Salerno
Outline

- Historical overview of GB methods in PR
  - The birth of graphs: promises and hopes
  - Pure Period: methods work in the graph space
    - Exact and inexact matching methods
    - Graph distance
  - Impure Period: the first steps toward vectors
  - Extreme Period: Graphs turn into vectors
- A brief survey of main methods and algorithms
- Conclusions and perspectives
The birth of graphs
Graphs vs Vectors

- SPR is based on a well founded mathematical framework: Vector Spaces
- Describing patterns by a vector of a set of measures has an immediate meaning
  - The pattern is a point in a Vector Space
  - (If the features are good) the distance of the points stands as a similarity measure between the corresponding patterns
  - Plenty of learning and classification methods
Are vectors really effective?

- The world is made of complex patterns
- Patterns generally contains subparts which are related each other
- Descriptions based on a set of features aren’t effective: it is expected that the more complex an object is the more articulated its description be
Angela in a vector…

… and Silvio

How Similar are Silvio and Angela?

$\hat{d} = 1.9$

<table>
<thead>
<tr>
<th></th>
<th>F_W</th>
<th>F_H</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>24.8</td>
<td>38.7</td>
<td>18.61</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>F_W</th>
<th>F_H</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>25.2</td>
<td>39.5</td>
<td>20.33</td>
</tr>
</tbody>
</table>
From FV to Graphs

- infact drawbacks pop up when the patterns have structural relationships
- and statistical distribution of a feature set is not so effective to catch the differences
- CONS: Inadequacy to deal with the decomposition into parts
Graphs vs Vectors

Graph-based representations decompose the object into parts
- Single parts are individually described
- Relations between the parts are represented
- FV are added to edges and nodes
- Different information for different nodes

Object comparison exploits contextual knowledge (inside a single object)
Structural comparison

\( d = 5.7 \)
Preserving the semantic value of the context
Pure Methods
Working into “pure” graph domain

Graph Comparison: compatibility between the structure of the input patterns

- Morphism between graph structures
- Node and Edge compatibilities

Symbolic graph prototypes:

- Manually
- Learned automatically ??
Exact Graph Matching

- Mapping between the nodes of two graphs:
  - edge-preserving: if two nodes in the first graph are linked by an edge, the corresponding nodes of the second graph must have an edge too
  - satisfying some general constraints:
    - Isomorphism
    - Sub graph isomorphism
    - Monomorphism
    - Homomorphism
Exact Graph Matching
Optimal, Tree based Search
EGM: Tree Search

- Use of State Space Representation (SSR): Each state represents a Partial Match (consistent with the matching type)
- A partial match is iteratively expanded by adding to it new pairs of matched nodes.
- The candidate pair to add, is chosen using some necessary conditions ensuring its consistency with the matching type.
- Use some heuristic condition to prune as early as possible unfruitful search paths.
EGM: Tree Search Example

\[ M(s0) = \{} \]
\[ M(s1) = \{(1,A)\} \]
EGM: Tree Search Example

\[ M(s0) = \{0\} \]
\[ M(s1) = \{(1,A)\} \]
\[ M(s2) = \{(1,B)\} \]
**EGM: Tree Search Example**

- $M(s0) = \{0\}$
- $M(s1) = \{(1,A)\}$
- $M(s2) = \{(1,B)\}$
- $M(s3) = \{(1,C)\}$
EGM: Tree Search Example

M(s0) = {0}
M(s1) = {(1,A)}
M(s2) = {(1,B)}
M(s3) = {(1,C)}
M(s4) = {(1,D)}
EGM: Tree Search Example

\[ M(s0) = \{0\} \]
\[ M(s1) = \{(1,A)\} \]
\[ M(s2) = \{(1,B)\} \]
\[ M(s3) = \{(1,C)\} \]
\[ M(s4) = \{(1,D)\} \]
\[ M(s5) = \{(1,A), (2,B)\} \]
EGM: Tree Search Example

\[
\begin{align*}
M(s_0) &= \{0\} \\
M(s_1) &= \{(1,A)\} \\
M(s_2) &= \{(1,B)\} \\
M(s_3) &= \{(1,C)\} \\
M(s_4) &= \{(1,D)\} \\
M(s_5) &= \{(1,A), (2,B)\} \\
M(s_6) &= \{(1,A), (2,C)\}
\end{align*}
\]
EGM: Tree Search Example

- $M(s_0) = \{0\}$
- $M(s_1) = \{(1,A)\}$
- $M(s_2) = \{(1,B)\}$
- $M(s_3) = \{(1,C)\}$
- $M(s_4) = \{(1,D)\}$
- $M(s_5) = \{(1,A), (2,B)\}$
- $M(s_6) = \{(1,A), (2,C)\}$
- $M(s_7) = \{(1,A), (2,D)\}$
EGM: Tree Search Example

\[
\begin{align*}
M(s_0) &= \{0\} \\
M(s_1) &= \{(1,A)\} \\
M(s_2) &= \{(1,B)\} \\
M(s_3) &= \{(1,C)\} \\
M(s_4) &= \{(1,D)\} \\
M(s_5) &= \{(1,A), (2,B)\} \\
M(s_6) &= \{(1,A), (2,C)\} \\
M(s_7) &= \{(1,A), (2,D)\} \\
\ldots
\end{align*}
\]
EGM: Tree Search Example

\[
\begin{align*}
M(s0) &= \{0\} \\
M(s1) &= \{(1, A)\} \\
M(s2) &= \{(1, B)\} \\
M(s3) &= \{(1, C)\} \\
M(s4) &= \{(1, D)\} \\
M(s5) &= \{(1, A), (2, B)\} \\
M(s6) &= \{(1, A), (2, C)\} \\
M(s7) &= \{(1, A), (2, D)\} \\
\ldots
\end{align*}
\]
EGM: Tree Search Example

M(0) = {0}
M(1) = {(1,A)}
M(2) = {(1,B)}
M(3) = {(1,C)}
M(4) = {(1,D)}
M(5) = {(1,A),(2,B)}
M(6) = {(1,A),(2,C)}
M(7) = {(1,A),(2,D)}
...

Diagram:

- M(s0) = {0}
- M(s1) = {(1,A)}
- M(s2) = {(1,B)}
- M(s3) = {(1,C)}
- M(s4) = {(1,D)}
- M(s5) = {(1,A),(2,B)}
- M(s6) = {(1,A),(2,C)}
- M(s7) = {(1,A),(2,D)}
EGM: Tree Search Example

\[
\begin{align*}
M(s0) &= \{0\} \\
M(s1) &= \{(1,A)\} \\
M(s2) &= \{(1,B)\} \\
M(s3) &= \{(1,C)\} \\
M(s4) &= \{(1,D)\} \\
M(s5) &= \{(1,A), (2,B)\} \\
M(s6) &= \{(1,A), (2,C)\} \\
M(s7) &= \{(1,A), (2,D)\} \\
M(s10) &= \{(1,A), (2,B), (3,C), (4,D)\}
\end{align*}
\]
EGM: Tree Search

- For each couple of nodes requires additional time for feasibility
- But feasibility rules allow the pruning
- The more asymmetric is the graph the more is the reduction of the matching time
Ullmann's algorithm (1976)

- Based on tree search with backtracking
- Finds isomorphism and subgraph isomorphism
- Pruning by a bit matrix M for each state
  - $M_{ij} = 1$ iff the matching of $n_i$ and $n_j$ is possible

Refinement: delete 1's from M of the current state
Then, the state is extended using the node pairs corresponding to the remaining 1's
Ullmann's refinement procedure

1. For each \((i,j)\) such as \(M_{ij} = 1\),
   - For each \(x\) such as \((i, x)\) is an edge of \(G_1\),
     - if there is at least one \(y\) compatible with \(x\) (\(M_{xy} = 1\)) and with a corresponding edge, i.e. \((j, y)\) is an edge of \(G_2\)
       - THEN \(M_{ij}\) remains unchanged
       - ELSE \(M_{ij}\) is set to 0

2. Repeat until \(M\) remains unchanged
1. Ullmann (1976)

Is Ga isomorphic to a subgraph of Gb?

M=?
$M_{ij}^0 = \begin{cases} 
1 & \text{if } \deg(B_j) \geq \deg(A_i), \\
0 & \text{otherwise}
\end{cases}, \forall i, j$
Laboratorio di Macchine Intelligenti per il riconoscimento di Video, Immagini e Audio
Laboratorio di Macchine Intelligenti per il riconoscimento di Video, Immagini e Audio
Laboratorio di Macchine Intelligenti per il riconoscimento di Video, Immagini e Audio
Laboratorio di Macchine Intelligenti per il riconoscimento di Video, Immagini e Audio
ATTENTION: the refinement procedure avoids to exploit all these states by evaluating “neighbour connections”.

\[ (\forall x)((a_{ix} = 1) \Rightarrow (\exists y)(m_{xy} \cdot b_{yj} = 1)) \]
ATTENTION:
the refinement procedure avoids to exploit all these states by evaluating “neighbour connections”.

\[(\forall x)(((a_{ix} = 1) \Rightarrow (\exists y)(m_{xy} \cdot b_{yj} = 1)))\]
ATTENTION:
the refinement procedure avoids to exploit all these states by evaluating “neighbour connections”.

$$(\forall x)((a_{ix} = 1) \Rightarrow (\exists y)(m_{xy} \cdot b_{yj} = 1))$$
ATTENTION: The refinement procedure avoids to exploit all these states by evaluating "neighbour connections".

\[
(\forall x)((a_{ix} = 1) \Rightarrow (\exists y)(m_{xy} \cdot b_{yj} = 1))
\]

where

- \(a_{ix} = 1\)
- \(b_{yj} = 1\)
- \(m_{xy} = 1\)

The image shows a graph with nodes and connections, illustrating the concept of "neighbour connections".
ATTENTION:
the refinement procedure avoids to exploit all these states by evaluating “neighbour connections”.

\[(\forall x)((a_{ix} = 1) \Rightarrow (\exists y)(m_{xy} \cdot b_{yj} = 1))\]
1. Ullmann (1976)

PROS: the solution is found for each pair of graphs (the refinement procedures converges in a finite number of steps);

CONS: exponential time; very high memory requirements;

- The problem is formulated in terms of State Space Representation, where each state represents a partial mapping solution;
- A depth-first search is used;
- Five Feasibility rules are defined in order to prune the state space. These rules take into account:
  - the consistency of the current solution;
  - the consistency of the “future” solutions (1- and 2-look-ahead)
Outline of VF2

- $M(s)$ is the partial mapping of state $s$, (current set of pairs of matched nodes)
- $P(s)$ is the set of candidate pairs for extending the state $s$;
- $F(s,n,m)$ is the *feasibility predicate*, which indicates whether the addition of pair $(n,m)$ to state $s$ can produce an inconsistent mapping.
In more details, the feasibility rules are defined as follows:

\[ F_{syn}(s, n, m) = R_{pred} \land R_{succ} \land R_{in} \land R_{out} \land R_{new} \]

**\( R_{pred}(s, n, m) \)**

\[ \forall n' \in \text{Pred}(G_1, n) \cap M_1(s) \]
\[ \exists m' \in \text{Pred}(G_2, m) \cap M_2(s) : (n', m') \in M(s) \]
\[ \land \forall m' \in \text{Pred}(G_2, m) \cap M_2(s) \]
\[ \exists n' \in \text{Pred}(G_1, n) \cap M_1(s) : (n', m') \in M(s) \]

**\( R_{succ}(s, n, m) \)**

\[ \forall n' \in \text{Succ}(G_1, n) \cap M_1(s) \]
\[ \exists m' \in \text{Succ}(G_2, m) \cap M_2(s) : (n', m') \in M(s) \]
\[ \land \forall m' \in \text{Succ}(G_2, m) \cap M_2(s) \]
\[ \exists n' \in \text{Succ}(G_1, n) \cap M_1(s) : (n', m') \in M(s) \]

**\( R_{in}(s, n, m) \)**

\[ \text{Card}(\text{Succ}(G_1, n) \cap T_1^{\text{in}}(s)) \geq \text{Card}(\text{Succ}(G_2, m) \cap T_2^{\text{in}}(s)) \land \]
\[ \text{Card}(\text{Pred}(G_1, n) \cap T_1^{\text{in}}(s)) \geq \text{Card}(\text{Pred}(G_2, m) \cap T_2^{\text{in}}(s)) \]

**\( R_{out}(s, n, m) \)**

\[ \text{Card}(\text{Succ}(G_1, n) \cap T_1^{\text{out}}(s)) \geq \text{Card}(\text{Succ}(G_2, m) \cap T_2^{\text{out}}(s)) \land \]
\[ \text{Card}(\text{Pred}(G_1, n) \cap T_1^{\text{out}}(s)) \geq \text{Card}(\text{Pred}(G_2, m) \cap T_2^{\text{out}}(s)) \]

**\( R_{new}(s, n, m) \)**

\[ \text{Card}(\bar{N}_1(s) \cap \text{Pred}(G_1, n)) \geq \text{Card}(\bar{N}_2(s) \cap \text{Pred}(G_2, n)) \land \]
\[ \text{Card}(\bar{N}_1(s) \cap \text{Succ}(G_1, n)) \geq \text{Card}(\bar{N}_2(s) \cap \text{Succ}(G_2, n)) \]

\[ M(s_0) = \{0\} \]

\[ T_1^{\text{in}}(s_0) = \{\} \]
\[ T_1^{\text{out}}(s_0) = \{\} \]
\[ T_2^{\text{in}}(s_0) = \{\} \]
\[ T_2^{\text{out}}(s_0) = \{\} \]
\[ P(s_0) = \{(1,A), (1,B), (1,C), (1,D), \ldots (4,B), (4,C), (4,D)\} \]

\[ M(s0) = \{0\} \]
\[ M(s1) = \{(1,A)\} \]

\[ T_1^{in}(s1) = \{3, 4\} \]
\[ T_1^{out}(s1) = \{2, 4\} \]
\[ T_2^{in}(s1) = \{C, D\} \]
\[ T_2^{out}(s1) = \{B, D\} \]
\[ P(s1) = \{(2,B), (2,D), (4,B), (4,D)\} \]

\[ M(s0) = \{0\} \]
\[ M(s1) = \{(1,A)\} \]
\[ M(s2) = \{(1,A), (2,B)\} \]

\[ T_{1 \text{ in}}(s2) = \{3, 4\} \]
\[ T_{1 \text{ out}}(s2) = \{3, 4\} \]
\[ T_{2 \text{ in}}(s2) = \{C, D\} \]
\[ T_{2 \text{ out}}(s2) = \{C, D\} \]
\[ P(s2) = \{(3,D), (3,C), (4,D), (4,C)\} \]

\[ M_1(s2) = \{1,2\} \]
\[ M_2(s2) = \{A,B\} \]

$M(s0) = \{0\}$

$M(s1) = \{(1, A)\}$

$M(s2) = \{(1, A), (2, B)\}$

$M(s3) = \{(1, A), (2, B), (3, D)\}$

$R_{pred}(s, n, m) \iff$

\[\forall n' \in \text{Pred}(G1, n) \cap M_1(s)\]

\[\exists m' \in \text{Pred}(G2, m) \cap M_2(s) \quad (n', m') \in M(s)\]

\[\land \forall m' \in \text{Pred}(G2, m) \cap M_2(s)\]

\[\exists n' \in \text{Pred}(G1, n) \cap M_1(s) : (n', m') \in M(s)\]

$T_{1\text{in}}(s2) = \{3, 4\}$

$T_{1\text{out}}(s2) = \{3, 4\}$

$T_{2\text{in}}(s2) = \{C, D\}$

$T_{2\text{out}}(s2) = \{C, D\}$

$P(s2) = \{(3, D), (3, C), (4, D), (4, C)\}$

\begin{align*}
M(s0) &= \{0\} \\
M(s1) &= \{(1,A)\} \\
M(s2) &= \{(1,A), (2,B)\} \\
M(s4) &= \{(1,A), (2,B), (3,C)\}
\end{align*}

\begin{align*}
T_1^{\text{in}}(s2) &= \{3, 4\} \\
T_1^{\text{out}}(s2) &= \{3, 4\} \\
T_2^{\text{in}}(s2) &= \{C, D\} \\
T_2^{\text{out}}(s2) &= \{C, D\} \\
P(s2) &= \{(3,D), (3,C), (4,D), (4,C)\}
\end{align*}

- Pred (G1,3) = \{2, 4\}
- Pred (G2, C) = \{B, D\}

- M(s0) = \{0\}
- M(s1) = \{(1, A)\}
- M(s2) = \{(1, A), (2, B)\}
- M(s4) = \{(1, A), (2, B), (3, C)\}

- T_1^{in}(s2) = \{3, 4\}
- T_1^{out}(s2) = \{3, 4\}
- T_2^{in}(s2) = \{C, D\}
- T_2^{out}(s2) = \{C, D\}
- P(s2) = \{(3, D), (3, C), (4, D), (4, C)\}

- \exists m' \in \text{Pred}(G_2, m) \cap M_2(s) \land (n', m') \in M(s)
- \forall m' \in \text{Pred}(G_2, m) \cap M_2(s) \land (n', m') \in M(s)
- \exists n' \in \text{Pred}(G_1, n) \cap M_1(s) \land (n', m') \in M(s)
Succ (G1,3) = {1,4}
Succ (G2,C) = {A,D}

M(s0) = {0}
M(s1) = {(1,A)}
M(s2) = {(1,A), (2,B)}
M(s4) = {(1,A), (2,B), (3,C)}

\[ T_1^{\text{in}}(s2) = \{3, 4\} \]
\[ T_1^{\text{out}}(s2) = \{3, 4\} \]
\[ T_2^{\text{in}}(s2) = \{C, D\} \]
\[ T_2^{\text{out}}(s2) = \{C, D\} \]
\[ P(s2) = \{(3,D), (3,C), (4,D), (4,C)\} \]

\[ R_{\text{succ}}(s, n, m) \iff \forall n' \in \text{Succ}(G_1, n) \cap M_1(s), \exists m' \in \text{Succ}(G_2, m) \cap M_2(s), (n', m') \in M(s) \]
\[ \land \forall m' \in \text{Succ}(G_2, m) \cap M_2(s), \exists n' \in \text{Succ}(G_1, n) \cap M_1(s) : (n', m') \in M(s) \]
Succ (G1,3) = {1,4}
Succ (G2,C) = {A,D}

M(s0) = {0}
M(s1) = {(1,A)}
M(s2) = {(1,A), (2,B)}
M(s4) = {(1,A), (2,B), (3,C)}

T1\text{in}(s2) = \{3, 4\}
T1\text{out}(s2) = \{3, 4\}
T2\text{in}(s2) = \{C, D\}
T2\text{out}(s2) = \{C, D\}
P(s2) = \{(3,D), (3,C), (4,D), (4,C)\}

R_{\text{succ}}(s, n, m) \iff
\forall n' \in \text{Succ}(G1, n) \cap M1(s)
\exists m' \in \text{Succ}(G2, m) \cap M2(s) \land (n', m') \in M(s)
\land \forall m' \in \text{Succ}(G2, m) \cap M2(s)
\exists n' \in \text{Succ}(G1, n) \cap M1(s) \land (n', m') \in M(s)
Succ (G1,3) = {1,4}
Succ (G2,C) = {A,D}

M(s0) = {0}
M(s1) = {(1,A)}
M(s2) = {(1,A), (2,B)}
M(s4) = {(1,A), (2,B), (3,C)}

P(s2) = {(3,D), (3,C), (4,D), (4,C)}

Succ (G1,3) = \{1,4\}
Succ (G2,C) = \{A,D\}

Pred (G1,3) = \{2,4\}
Pred (G2,C) = \{B,D\}

\[ \begin{align*}
M(s0) &= \{0\} \\
M(s1) &= \{(1,A)\} \\
M(s2) &= \{(1,A), (2,B)\} \\
M(s4) &= \{(1,A), (2,B), (3,C)\}
\end{align*} \]

\[ \begin{align*}
T_1^{\text{in}}(s2) &= \{3, 4\} \\
T_1^{\text{out}}(s2) &= \{3, 4\} \\
T_2^{\text{in}}(s2) &= \{C, D\} \\
T_2^{\text{out}}(s2) &= \{C, D\} \\
P(s2) &= \{(3,D), (3,C), (4,D), (4,C)\}
\end{align*} \]

\[ \begin{align*}
R_{\text{in}}(s, n, m) &\iff \\
(\text{Card}(\text{Succ}(G_1, n) \cap T_1^{\text{in}}(s))) \geq \text{Card}(\text{Succ}(G_2, m) \cap T_2^{\text{in}}(s))) \land \\
(\text{Card}(\text{Pred}(G_1, n) \cap T_1^{\text{in}}(s))) \geq \text{Card}(\text{Pred}(G_2, m) \cap T_2^{\text{in}}(s)))
\end{align*} \]

$\text{Succ (G1,3)} = \{1, 4\}$
$\text{Succ (G2, C)} = \{A, D\}$

$\text{M(s0)} = \{0\}$
$\text{M(s1)} = \{(1, A)\}$
$\text{M(s2)} = \{(1, A), (2, B)\}$
$\text{M(s4)} = \{(1, A), (2, B), (3, C)\}$

$\text{T}_{1_{\text{in}}}(s2) = \{3, 4\}$
$\text{T}_{1_{\text{out}}}(s2) = \{3, 4\}$
$\text{T}_{2_{\text{in}}}(s2) = \{C, D\}$
$\text{T}_{2_{\text{out}}}(s2) = \{C, D\}$

$\text{P}(s2) = \{(3, D), (3, C), (4, D), (4, C)\}$

$\text{R}_{\text{out}}(s, n, m) \iff$
$\text{(Card } \text{Succ(G1, n)} \cap T_{1_{\text{out}}}(s) \text{) } \geq \text{ Card } \text{Pred(G1, n)} \cap T_{1_{\text{out}}}(s) \text{)}$
$\text{Card } \text{Succ(G2, m)} \cap T_{2_{\text{out}}}(s) \text{) } \land$
$\text{Card } \text{Pred(G2, m)} \cap T_{2_{\text{out}}}(s) \text{) }$

$M(s0) = \emptyset$
$M(s1) = \{(1,A)\}$
$M(s2) = \{(1,A), (2,B)\}$
$M(s4) = \{(1,A), (2,B), (3,C)\}$

$Succ\ (G1, 3) = \{1,4\}$
$Succ\ (G2, C) = \{A, D\}$

$Pred\ (G1, 3) = \{2,4\}$
$Pred\ (G2, C) = \{B, D\}$

$R_{out}(s, n, m) \iff$
$\left( \text{Card}(Succ(G1, n) \cap T_1^{out}(s)) \geq \text{Card}(Succ(G2, m) \cap T_2^{out}(s)) \right) \land$
$\left( \text{Card}(Pred(G1, n) \cap T_1^{out}(s)) \geq \text{Card}(Pred(G2, m) \cap T_2^{out}(s)) \right)$

$1 \geq 1 \text{ and } 1 \geq 1$

$T_1^{in}(s2) = \{3, 4\}$
$T_1^{out}(s2) = \{3, 4\}$
$M_1(s2) = \{1, 2\}$

$T_2^{in}(s2) = \{C, D\}$
$T_2^{out}(s2) = \{C, D\}$
$M_2(s2) = \{A, B\}$

$P(s2) = \{(3,D), (3,C), (4,D), (4,C)\}$

\[
\begin{align*}
M(s0) &= \{0\} \\
M(s1) &= \{(1,A)\} \\
M(s2) &= \{(1,B)\} \\
M(s3) &= \{(1,C)\} \\
M(s4) &= \{(1,D)\} \\
M(s5) &= \{(1,A) , (2,B)\} \\
M(s6) &= \{(1,A) , (2,C)\} \\
M(s7) &= \{(1,A) , (2,D)\} \\
\ldots \\
M(s10) &= \{(1,A) , (2,B) , (3,C) , (4,D)\}
\end{align*}
\]

**PROS:**
- It is applicable to any kind of graphs;
- Linear Memory Complexity;

**CONS**
- Not yet found!
EGM: Other techniques

- The graph matching into a sort of vector-based matching
- From the input graphs a representation (not graph-based of course)
- We compare the two representations (in polynomial time)
- Generally give necessary but not sufficient solutions
Nauty (McKay 1981)

- based on permutation group theory
- only graph isomorphism
- first builds the group of automorphisms for each graph (automorphism = isomorphism of a graph with itself)
- then, the automorphisms groups are used to define a canonical ordering of the nodes
  - once the nodes are reordered, two graphs are isomorphic iff their adjacency matrices are equal
A simple Example (1/4)

Two graphs and their adjacency matrices
A simple Example (2/4)

Finds a canonical labeling for G1 by the permutation groups:

- n1 => n'2
- n2 => n'1
- n3 => n'3

e.g. counting the output degree of the nodes
The same is done for G2:
- m1 => m'1
- m2 => m'3
- m3 => m'2
The two canonically labeled graphs are compared directly using their adjacency matrices.

### A simple Example (4/4)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

=  

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- A new approach for solving the Subgraph isomorphism using constraint programming.
- The idea is to label every node by some invariant property (es. Node degrees), and iteratively extends labels by considering labels of adjacent nodes.
- Labeling is used to define a filtering algorithm for the subgraph isomorphism problem. (Iterative Labeling Filtering – ILF)

- Labeling process allows to filter, for every node in the graph G1, the set of target nodes in G2.
- The node compatibility is evaluated by a partial order, consistent with respect to the subgraph isomorphism function.
- A consistent labeling filters node domains, as strong as possible, without removing solutions to the subgraph isomorphism problem.

Label Assignment Function

\[ \alpha : \text{Node set} \rightarrow \text{Label Set} \]

Partial Order

\[ CC_l = \{(u, v) \in N_p \times N_t \mid \alpha(u) \preceq \alpha(v)\} \]

Set of compatible couples of nodes

- This compatibility relationship may be used to filter the node domain of the graph G1, removing from it every node in G2 not in \( CC_l \)
- A labelling is subgraph isomorphism consistent iff for any subgraph isomorphism function \( f \), we have that for any node \( u \) in \( G1 \) \((u, f(u))\) is in \( CC_l \)

- Subgraph Isomorphism consistent labelling
  - Degree based
  - K-Distance based: nodes at distance $k$
  - K-Clique based

- Degree Based example:
  \[
  \begin{align*}
  \deg(A) &= \deg(B) = \deg(D) = \deg(2) = \deg(4) = 4 \\
  \deg(C) &= \deg(E) = \deg(F) = \deg(G) = \deg(1) = \deg(3) = 3 \\
  \deg(5) &= \deg(6) = 2
  \end{align*}
  \]

\[
CC_{\deg} = \{(u, v) \mid u \in \{2, 4\}, v \in \{A, B, D\}\} \\
\cup \{(u, v) \mid u \in \{1, 3, 5, 6\}, v \in \{A, B, C, D, E, F, G\}\}
\]

The number of compatible nodes is reduced by an iterative labeling strengthening procedure using neighbor information.

This procedure is terminated when the compatible couple set cannot be further reduced, or when a user defined threshold is reached.

**PROS:**
- Different approach in respect to the classical CSP based on Forward Checking or Arc Consistency.
- Time Efficient.

**CONS:**
- The precision and the time of convergence depend on the chosen threshold.
- Only for subgraph isomorphism.
Pure Methods
Inexact Graph Matching
IGM: Tree Search
Main algorithms

• Tsai–Fu, 1979
• Shapiro–Haralick, 1981
• Wong, 1990
• Sasha, 1994
• Allen, 1997
• Berretti, 2000

Gharaman, 1980
Eshera – Fu, 1984
Dumay, 1992
Cordella, 1996
Serratosa, 1999
Gregory – Kittler, 2002
IGM: Continuous Optimization

- Casting graph matching into a *continuous, non linear optimization problem* by removing some constraints.
- The solution is built by successive improvements of a continuous goal function.
- The solution needs to be converted back from continuous domain: this may introduce further approximation.
IGM: Error-correcting

- **Error Correcting Cost**: the matching cost is based on an explicit model of the errors
- **Graph Edit Cost**: based on a set of graph-edit operation (node insertion, edge insertion, edge deletion, etc..); is the cost of the operations sequence to transform one of the graphs into the other.
- **Graph Edit Distance**: the edit cost is used as a measure of dissimilarity of the graphs.
Examples:

G1

G2
Examples:

G1

G2

edge-removal(b,c)
Examples:

G1

G2

attribute-changing(a,f): attributes a of G1 and f of G2 are compatible
Examples:

G1

G2

G1₁

G1₂

G1₃

node-addition(e)
Examples:

G1

G2

g1

g2

g1

g2

g1

g2

edge-addition(b,e)
Examples:

G1

G2

edge-addition(c,e)
Examples:

G1

G2

G1_1

G1_2

G1_3

G1_4
Error-correcting Edit distance

- Uses a tree based search for finding the edit paths which can transform G1 into G2 having the minimum cost.
- The algorithm works into SSR: a path in the tree represents an edit path among the graphs.
- A current edit-path is closed if it allows to reach a same intermediate state with an higher edit-cost.
IGM: Tree Search

- Use tree search with backtracking.
- Direct the search by the cost of the Partial Matching.
- Use, as heuristic, estimate of the matching cost for the remaining nodes.
- Use A* like algorithm to prune unfruitful paths.
IGM: Tree Based Edit-Paths

Each state represents a possible partial matching by considering the needed edit operations and a cost is associated to.

$M(s_0) = \{0\}$
$M(s_1) = \{(1, A)\}$
$M(s_2) = \{(1, B)\}$
$M(s_3) = \{(1, C)\}$
$M(s_4) = \{(1, D)\}$
$M(s_5) = \{(1, A), (2, B)\}$
Impure Methods
Basic operations in a vector space

- Impure methods report in the graph space the main operations defined in a vector space.

- The idea is to reuse on graphs the basic methods in SPR:
  - The first important achievement is the concept of distance between graphs.
Impure Graph Matching and Learning

Once the graph distance has been defined we can reuse effective SPR methods for learning and classifying graphs, as:

- NN, K-NN, (K-K’)-NN

The latter allows to solve a supervised classification problem given a set of labelled graphs, only using the distance
Are we ready?

- We have a graph distance (edit cost dist.)
- … not all the properties are always valid:
  - $D(G_1, G_2) \geq 0$
  - $D(G_1, G_2) = D(G_2, G_1)$ Doesn’t hold
  - $D(G_1, G_x) \leq D(G_2, G_x) + D(G_x, G_2)$ Triang disequality holds only using exhaustive A* search of edit-cost searching algorithm
- It’s a similarity measure not a distance
NN Classifier

Given a set $S$ of labeled graphs (reference set)

An input graph $G$ (to classify)

The graph $G_x$ of $S$ having the minimum edit distance from $G$ is determined

$G$ is assigned the class of $G_x$

With simple variants we obtain $K$-NN and $(K-K')$-NN
Other achievements with graph distance

- Many learning procedures in a vector space use the centroid of points as a prototype (K-means clustering)
- Points are graphs and centroids prototypes.
- We are required to define graph centroids:
  - Median graphs
  - Generalized Median Graph
Graph prototype as the median graph

Given a set $S$ of graphs of a same class, a prototype $C$ of $S$ is the graph which minimizes the sum of its distances from the graph in $S$ (Median Graph)

$$
\hat{S} = \text{argmin}_{g_1 \in S} \sum_{g_2 \in S} d(g_1, g_2)
$$

Differences with a space vector: the space is discrete and the median graph belonging to $S$!
From Median to Generalized Median Graph
Improving graph prototypes

- The graph Generalized Median of a set $S$ of graphs graph is obtained by preliminarily building $S'$
- $S'$ is a very wide set of graphs obtained by the graphs in $S$ changing in any possible mode all the (edge and node) labels to any of the graphs in $S$

\[
\hat{S} = \arg\min_{g_1 \in S} \sum_{g_2 \in S'} d(g_1, g_2)
\]
From Median to Generalized Median Graph
Another achievement

- Up to now:
  - NN Classifiers
  - Supervised Clustering

- Going ahead:
  - Weighted Sum of Graphs: the graph obtained as cG1 + aG2
We can reuse LVQ

A simple LVQ algorithm determining the vector centroids of n classes $C_1, C_n$, is:

1) Pick a graph $G_{in}$ randomly from TS and calculate the nearest quantization graph $C_w$

2) Move $C_w$ (median graph) towards $G_{in}$ by a fraction of the distance, if the class of $C_w$ is correct, otherwise move in the opposite

Repeat until the total error on the training becomes stable
Extending LVQ to Graphs

Given two graphs $G_1$ and $G_2$ we need to transform $G_1$ into another graph $G'$ so as to reduce the distance between $G'$ and $G_2$. 
Graph LVQ: What we need? 2/2

- The distance between graphs gives the edit operations $e_1, e_2, \ldots, e_k$ that transform $G_1$ into $G_2$ with minimum cost.

- Dropping the $k$ first edit operations, we have a new sequence that transforms $G'$ into another graph $G_2$.

- It derives that $G'$ is more similar than $G_1$ to $G_2$ by the amount given by the sum of the edit cost dropped.
Example:
Further acquisitions

Any other learning algorithm based on distance can be used:

- Unsupervised Self Organizing Maps
Considerations

- The extension to graphs of vector space operations has a practical huge impact:
  - Standard learning and classification methods

- The theoretical framework isn’t robust:
  - the graph distance is discrete, as related to edit operations!
  - This distance is affected by uncontrollable errors as the edit distance depends on domain-based edit cost assignment
Graph Kernels
Kernels in PR

- They allow to extend basic linear algorithms to complex non-linear ones
- Non-linear regularities in the data is inherently treated
- Under some conditions kernel methods give better results in difficult PR tasks than traditional methods (Svm Vapnik)
Kernel trick: Mercer’s theorem

Given a vector space $V_1$ with a Kernel function $K$, there is at least another vector space $V_2$, usually having a higher dimensionality and a function $F$ mapping $V_1$ to $V_2$, such as:

- For every $a, b$ in $V_1$:
- $K(a, b) = F(a) \cdot F(b)$
Implications

- If the problem is not linearly separable it may became linear in the transformed space using a suited (unknown) kernel.
- There is no guarantee that this happens even considering thousands of kernels.
- Given a kernel there are no criteria to check whether it linearizes the problem.
Graph Kernel: meaning

- Informally, a graph kernel can be considered as a measure of the similarity between two graphs;
- However, its formal properties allow a kernel to replace the dot product in several vector-based algorithms using it (and other functions related to dot product, such as the Euclidean norm).
Implications

- The Kernel stands as dot product between the considered points in $V_1$
- With this assumption it’s possible to reuse the Kernel machines:
  - SVM
  - PCA
  - Perceptron
Kernels on Graph

A graph kernel is a function that maps a couple of graphs onto a real number

\[ k : G \times G \to R \]

\[ k(G_1, G_2) = k(G_2, G_1) \quad \forall \ G_1, G_2 \in G \]

\[ \sum_{i=1}^{n} \sum_{j=1}^{n} c_i \cdot c_j \cdot k(G_1, G_2) \geq 0 \]
Examples of graph Kernels

- $K(G_1,G_2) = 1$
- $K(G_1,G_2) = n_1 n_2$
- $K(G_1,G_2) = (n_1 + e_1) (n_2 + e_2)$
- $K(G_1,G_2) = \text{Diam}(G_1) \text{Diam}(G_2)$

Are these examples good?
Basic criteria for building (hopely) a good graph kernel

A Kernel should catch the similarity between graphs (the one good for the application at hand). For any graph:

If $K(G_1, G_2) > K(G_3, G_4)$, it is worth that:

Similarity($G_1, G_2$) < Similarity($G_3, G_4$)

Note that the latter is not trivial;
semantic cannot be simply included
Diffusion Kernels

- Applicable when we are dealing with a known finite set $T$ of $n$ graphs
- Defined on the basis of a similarity measure $s$ (any kind on the graph space)
- From $s$ a pairwise similarity matrix $S$ is obtained:
  - $S=S_{ij}=s(G_i,G_j)$, $i,j=1,n$
Diffusion Kernels

- Exponential diffusion Kernel \( (\lambda < 1) \)

\[
K = \sum_{k=0}^{\infty} \frac{1}{k!} \lambda^k S^k = \exp(\lambda S)
\]

- Neumann diffusion Kernel

\[
K = \sum_{k=0}^{\infty} \lambda^k S^k.
\]
Diffusion Kernels

- The sum is cut for practical issues
- Given $G_i$ and $G_j$ of the set $T$, their Kernel is the element $(i,j)$ of the matrix
- Note that in the evaluation of $K$ also other graphs will contribute (power of $K$)
- Heavy limitations for PR: closed sets of graphs
Convolution Kernels

Convolution kernels infer the similarity of composite objects from the similarity of their parts:

- Similarity between the parts are simpler to define
- Given the similarities between the simpler parts a convolution operation is applied in order to turn them into a kernel function
Convolution Kernels

- From G we obtain the set D of all its decompositions (e.g. the simplest is its decomposition into the set of its nodes)
- A kernel $K_{ji}$ is required for each element of D (e.g. the distance between nodes)

\[ \kappa(g, g') = \sum \sum \prod_{i=1}^{d} \kappa_i(g_i, g'_i). \]
Pros and Cons

- Well established theoretical assumption
- Usually they have a high computational complexity (it mainly depends on how the graph is decomposed)
- An example: decomposing the graph in its walks
Random walk kernel

- They measure the similarity of two graphs by evaluating the number of random walks in both graphs sharing some properties.

- Note: The number of matching walks in $G_1$ and $G_2$ can be computed by means of the direct product of the two graphs, without explicitly enumerate the walks.
Other Recents

- Laplacian Kernel: an evolution of the diffusion kernels
- Treetlet Kernel: a particular case of the convolution kernels; the graphs are decomposed in terms of treetlet (a node with its neighborhood)
Kernel on Graph Distance

- By Bunke: Graph distance as kernel; requires a 0-pattern (difficult to define)
- Other kernels: sum and the product of the previous over a set of 0-patterns
- Same theoretical properties, but more robust with respect to the choice of 0-pattern
Kernels: Conclusions?

- Mercer's theorem doesn’t hold for graph kernels! Even the (few) guarantees of this theorem are lost.
- The actual performance strongly depend on how appropriate is for task at hand the used similarity measure included in the graph kernel.
The Kernel approach

Create on the fly a new Kernel

Apply it

Does it work?

Yes

Problem Solved

No

Publish it
Extreme methods: Graph embedding
Graph embedding is used with two slightly different meanings:

- maps the nodes of a graph onto points in a vector space; nodes having similar structural properties will be mapped onto points which are close in this space;
- maps whole graphs onto points in a vector space, in such a way that similar graphs are mapped onto close points.
Graph Embedding

Graph Domain

Vector Domain

$G_1$, $G_2$, $G_3$, $G_4$
Graph embeddings

- It is easy to define a graph embedding...
  - any function from a graph \( g \) to a vector can be considered an embedding!
- ... But how effective is it?
Graph embeddings

... the embedding should be invariant wrt node permutation...

- ... so the adjacency matrix viewed as a vector is definitely NOT a good embedding

... but this is not enough!

- for instance, the vector \((N_n, N_e)\) with number of nodes and edges is invariant wrt permutations, but it is hardly an effective embedding in most applications
Graph embeddings: better properties

- Graphs that are similar should be mapped to vectors that are close according to a simple metric.
- Graphs that are different should be mapped to vectors that are distant.
- ... but of course, “similar” and “different” depends on what one wants to use the graphs for...
Graph embeddings

- Representation by the eigenvalues of the adjacency matrix in decreasing order
  - the eigenvalues are invariant \( \text{wrt} \) permutation
  - a slight alteration of a graph does not change very much its eigenvalues
- ... this is the basis for several spectral embedding techniques.
Graph embeddings

BUT....

- unfortunately, there are very different graphs with the same eigenvalues...

(+2,0,0,0,-2)
Graph embeddings

Currently the theoretical characterization of the embedding properties is still in its infancy...

- ... no guarantees of suitability for any specific application: you have to try and cross your fingers...
Conclusions

- Pure methods: operations directly in the graph domain
- Impure methods: operations on vectors are turned into operations on graphs
- Extreme methods: graphs are entirely and directly reconducted to vectors
Pure Methods: Conclusions

- **Pros:** Management of contextual semantic information. Solid theoretical framework
- **Cons:** Few learning and classification algorithms available.
- **Open issues:** Reduction of computational complexity, Lack of efficient algorithms for MCS, Need of good symbolic graph learning methods, Theoretical models for evaluating upper-bound of the error for IGM, Exploiting applications in the Bio-informatics
Impure Methods: Conclusions

- **Pros:** Use of well known learning and classification algorithms derived from SPR.

- **Cons:** Lack of a theoretical framework for properly assigning edit cost in a general way. Discrete nature of the graph space. Contextual semantic cannot be directly handled.

- **Open issues:** Improving the edit graph distance. Improvements of weighted sum of graphs (too discrete). Improvements in theoretical assignment of edit cost on a domain.
Kernel Methods: Conclusions

- **Pros**: Kernel machines available
- **Cons**: No theoretical guarantee about the goodness of used Kernel. Contextual semantic cannot be directly handled.
- **Open issues**: Definition of a theoretical framework for evaluating the goodness of a graph kernels. Improvement of graph-based similarity Kernel.
Embedding Methods: Conclusions

- **Pros:** Potentially all SPR methods
- **Cons:** Very young theoretical framework
- **Open issues:** A lot of work to do…
- **A bit of philosophy**…
- Is it more convenient to start from patterns, obtaining graphs and then embed them, OR
to try directly good representations of object into vectors?
Laboratorio di Macchine Intelligenti per il riconoscimento di Video, Immagini e Audio

THANK YOU