Structural Pattern Recognition

GREYC – CNRS UMR 6072, University of Caen, ENSICAEN
Luc.Brun@ensicaen.fr
Aims of Pattern Recognition

Design “Intelligent Systems”

- Intelligence ← inter legere (pick out, discern)
- Torture by removal of inputs
- One Def.:

  The aggregate or global capacity of the individual to act purposefully, to think rationally, **and to deal effectively with his environment.**

Aims of Pattern Recognition

Why is it challenging?

- A large part of our brain is devoted to the analysis of perceptions.
- Computer Science will be present everywhere: Houses, Cars, iTowns, Phones, laptop, ...
- Human interactions should overpass the screen/keyword interactions.

⇒ Computer should understand their environments and react accordingly... and somehow become intelligent.
Songs

- Recognition of simple commands:
  - Devices (phones, car, TV, house, fridge,...)
  - Standards of operators

- Identification of songs/Musics (Audio Fingerprint)
  - Copyright protection for You Tube/Dalymotion...
  - New services for mobile devices

- Voice/Song representation:
  - A vector (set of Fourier coefficients, Wavelet transform, ...)
  - A function
  - A string
## Shapes

### Confusion matrix

<table>
<thead>
<tr>
<th>Classes</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>♂</td>
<td>♀</td>
<td>▶</td>
<td>♂</td>
<td>♀</td>
<td>▶</td>
<td>♂</td>
<td>♀</td>
</tr>
<tr>
<td>♂</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>♀</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▶</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>♂</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>♀</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▶</td>
<td>1</td>
<td>9</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>♂</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>♀</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▶</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Examples
- Strings
- Bipartite Assignment
- Graph Terminology
- Graph Matching
- Graph Edit distance
Shapes

- Applications:
  - Character recognition,
  - Docking,
  - Identification of signatures,
  - Pose estimation
  - Detection & Characterization of spines

- Shape representations:
  - Vector of features (e.g. Legendre’s moments),
  - String representation of the boundary
  - Graph representation of the skeleton.
Image/Video indexation

- **Image Classification/Retrieval**
  
  All images related to a subject

- **Objects Detection**
  
  Learn: object’s appearances and relationships with context.

- **Visual Words**
  
  Intermediate representation adapted to object representations.

(a) classic research engine, (b) [CVPR10]
Brain

- Decode brain activity.
  1. Submit stimuli to subjects,
  2. Measure brain activities,
  3. Retrieve stimuli from brain activities.
- Activity measurements:
  1. Set of signals (for each active zone),
  2. Graph of zones with correlated signals.
Document Analysis

- Applications:
  - Structuring of document,
  - Retrieval and identification of signatures,
  - Analysis of Technical Drawings,
  - Reading of addresses,
  - Analysis of old documents.

- Tools:
  - Graphs,
  - Shape descriptors,
  - Markov RF,
Biometry

- People identification by:
  - Fingers ++,
  - Hands +,
  - Eyes +++,
  - Face +,
  - Voice,

- Features:
  - Set of points (minuties),
  - Measure of texture,...
Video Analysis

- **Aims:**
  - Surveillance/tracking of people on a single/network of camera,
  - Detection of abnormal trajectories,

- **Applications:**
  - Security,
  - Sports,
  - iTowns

- **Main tools:**
  - Background subtraction,
  - histograms, vectors, graphs, strings...
Chemoinformatics

Building a new molecule requires many attempts and many tests: Time consuming, expensive.

- **Aims**
  - Predict physical/biological properties of molecules (Virtual Screening)
  - Regression (Continuous properties),
  - Classification (Discrete Properties),
    - Cancerous/not cancerous,
    - Active against some diseases (Aids, Depression, ...).

- **Tools**
  - Vector of properties,
  - Molecular graph.
Different Levels/Steps of Recognition

**Level 0:** Hand made classification (Expert Systems)

If \( x > 0.3 \) and \( y < 1.5 \) then TUMORS

**Level 1:** Design of feature vectors/(di)similarity measures. Automatic Classification

**Level 2:** Automatic design of pertinent features / metric from huge amount of examples.

We will mainly study Level 1 systems within the structural pattern recognition framework.
Statistical vs Structural Pattern Rec.

- **Statistical pattern recognition**
  - Based on numerical descriptions of objects,
  - Focused on individual descriptions.

- **Structural Pattern recognition**
  - Based on both numerical/symbolic description of objects,
  - Focused on the relationships between objects.

- **Pros and Cons :**
  - **Statistical**
    - Many efficient algorithms exists to manipulate numerical values,
    - Individual description of objects may lead to poor descriptions,
  - **Structural**
    - Nicely describe both the individual objects and their relationships,
    - Based on complex structure (Graphs, Strings) which can not be readily combined with numerical algorithms.
Strings

- Let $L$ be an alphabet ($L : \mathbb{R}^p$, a sequence of symbols, combination of both,..)
- A finite string of length $n$ is an element of $L^n$
- An infinite string is an element of $L^\infty$.
- A circular string of $L^n$ is a string such that $s[n + 1] = s[1]$.
- Appear between any sequential relationships between objects:
  - Temporal (trajectories, song, video,...),
  - Spatial (DNA, text, shape’s boundaries...)
- Examples:
  - $s=”Hello” : Text$
  - $s=”AGATACA” : DNA$
  - $s=”.2 .4 .04 1.0” Song fingerprint.$
Let $s_1$ and $s_2$ denote two strings. The string edit distance (Levenshtein distance) is the minimum number of edits needed to transform $s_1$ into $s_2$, with the allowable edit operations being insertion, deletion, or substitution of a single character.

Let $s_1 = "restauration$$", s_2 = "restaurant$$"

- restauration $\rightarrow$ restaurantion (insertion of n)
- restaurantion $\rightarrow$ restaurantio (removal of n)
- restaurantio $\rightarrow$ restaurant (removal of i and o)
- d("restauration","restaurant")=4

This edit distance is readily extended by associating different costs to each operation. In this case, the cost of a transformation is the sum of the costs of elementary operations and the edit distance is the transformation with minimal cost.
Dynamic Programming

A computer science problem may be (efficiently) solved by dynamic programming if it has:

**Optimal substructure**: The optimum we search for may be deduced from the optimal solutions of sub-problems.

**Overlapping sub-problems**: The naive recursive decomposition of a problem into sub-problems leads to solve many times the same problem.

Example: The Pascal triangle:

\[
\binom{n}{p} = \binom{n-1}{p-1} + \binom{n-1}{p}
\]

\[
\begin{align*}
\binom{10}{7} &= \binom{9}{6} + \binom{9}{7} \\
\binom{9}{7} &= \binom{8}{6} + \binom{8}{7}.
\end{align*}
\]

In case of non overlapping problem the strategy is called divide and conquer.
Dynamic programming:

- Let us suppose that
  - We want to compute the edit distance between $s_1$ and $s_2$ up to indexes $i$ and $j$
  - we know the optimal solution for any $(k, l)$ such that $k + l < i + j$. 

$$d(s_1[1 \ldots, i], s_2[1 \ldots, j]) = \min\left(\begin{array}{c}
d(s_1[1 \ldots, i-1], s_2[1 \ldots, j]) + c_{\text{supp}}(s_1[i]), \\
d(s_1[1 \ldots, i], s_2[1 \ldots, j-1]) + c_{\text{add}}(s_2[j]), \\
d(s_1[1 \ldots, i-1], s_2[1 \ldots, j-1]) + c_{\text{sub}}(s_i[i], s_2[j])
\end{array}\right)$$
String Edit distance

**Algorithm**

```plaintext
function LEVENSHTEINDISTANCE(char s1[1..m], char s2[1..n])
    $d[0, 0] \leftarrow 0$
    for $i = 1 \rightarrow m$
        $d[i, 0] \leftarrow d[i - 1, 0] + c_{supp}(s1[i])$
    end for
    for $j = 1 \rightarrow n$
        $d[0, j] \leftarrow d[0, j - 1] + c_{add}(s2[j])$
    end for
    for $j = 1 \rightarrow n$
        for $i = 1 \rightarrow m$
            $d[i, j] \leftarrow min\left(\begin{array}{c}
                d[i - 1, j] + c_{supp}(s1[i]),
                d[i, j - 1] + c_{add}(s2[j]),
                d[i - 1, j - 1] + c_{sub}(s1[i], s2[j])
            \end{array}\right)$
        end for
    end for
    return $d[m, n]$
end function
```

▷ Removal of $s_1$ prefixes

▷ Insertion of $s_2$ prefixes
String Edit distance

Illustration: all costs equal to 1

Example:

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>a</th>
<th>t</th>
<th>u</th>
<th>r</th>
<th>d</th>
<th>a</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sat</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>u</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>n</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Exercise:

<table>
<thead>
<tr>
<th></th>
<th>E</th>
<th>c</th>
<th>o</th>
<th>n</th>
<th>o</th>
<th>m</th>
<th>i</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eng</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
Sting Edit distance and Global Alignment

Let \( x = (x_1, \ldots, x_n) \) and \( y = (y_1, \ldots, y_m) \) denote 2 strings.

An alignment \( \pi \) between \( x \) and \( y \) is defined by two vectors \( (\pi_1, \pi_2) \) of length \( p < n + m - 1 \) such that:

\[
\begin{align*}
1 &\leq \pi_1(1) \leq \cdots \leq \pi_1(p) = n \\
1 &\leq \pi_2(1) \leq \cdots \leq \pi_2(p) = m
\end{align*}
\]

with unitary increasing and no repetitions:

\[
\forall i \in \{1, \ldots p - 1\} \left( \begin{array}{c}
\pi_1(i + 1) - \pi_1(i) \\
\pi_2(i + 1) - \pi_2(i)
\end{array} \right) \in \left\{ \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}
\]

The Dynamic Time Warping (DTW) distance is then defined as:

\[
DTW(x, y) = \min_{\pi \in \mathcal{A}(n, m)} \sum_{i=1}^{\lvert p \rvert} c_{sub}(x_{\pi_1(i)}, y_{\pi_2(i)})
\]

where \( \mathcal{A}(n, m) \) denotes the set of alignments between strings of length \( n \) and \( m \).
Global Alignment

- Example:

```
restauraton
restauran
```

\[ \pi_1 : 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 12 \ 13 \]
\[ \pi_2 : 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \]

- Connection with edit Distance:

```
restauraton
restauran
```

edit script \[ S \ S \ S \ S \ S \ S \ S \ S \ D \ D \ D \ D \ S \ I \]
• Complexity in $O(nm)$

• Extensions:
  - Circular string edit distance,
  - (Circular) string edit distance with rewritings, ...

• Conclusion:
  - May satisfy all axioms of a distance (according to $c_{sub}, c_{add}, c_{supp}$),
  - Efficient algorithm on small/usual pattern recognition tasks,
Due to its complexity, string edit distance is a bit useless on very long strings (songs, DNA, long texts, ...).

→ need to localize quickly promising parts of 2 strings with a low edit distance or to bound (quickly) the edit distance between two string.

Given $s_1[1 \ldots, n]$ and $s_2[1 \ldots, m]$ ($n >> m$) find all sub strings $s$ of $s_1$ such that $d(s, s_2) \leq k$.

A q gram is a word of length q.

The set of q grams of a string $s$ may be found in $|s|$ and stored in a databased together with its location.

$Q$ grams distance between strings :

$$D_q(x, y) = \sum_{s \in \Sigma_q} |H_s(x) - H_s(y)|$$

$$= \sum_{s \in Gr(x) \cap Gr(y)} |H_s(x) - H_s(y)| + \sum_{s \in Gr(x) - Gr(y)} |H_s(x)| + \sum_{s \in Gr(y) - Gr(x)} |H_s(y)|$$

$Gr(x)$: Set of q grams of $x$. 

Q grams
Fundamental theorem: Let $x$ be a pattern string and $y$ a text (song, DNA,...) :

$$d(x, y) < k \Rightarrow Gr(x) \cap Gr(y) \geq |x| - q + 1 - kq$$

Given any part $y$ of a text $T$. $Gr(y)$ may be computed linearly and if $Gr(x) \cap Gr(y) < |x| - q + 1 - kq$ then $d(x, y) \geq k$ and $y$ may be rejected without computing $d(x, y)$. 
An application to audio Fingerprint

- Compute the fingerprints of a database of songs.
  - Each fingerprint is a string
  - One element of a string \( \approx \) every 5/10ms.
  - Each fingerprint is a string of approx. 18,000 elements.
  - Several thousands of songs in the database.
- Let \( I \) be an input song of 5 seconds. For each q gram of \( I \), let \( S_q \) denote the set of fingerprints \( D \) of the database such that \( q \in Gr(D) \).

\[
\text{score}_{I,D}(q) = \sum_{k=1}^{p} \sum_{l=1}^{q} S(I[i_k, i_k + m], D[j_l, j_l + m])
\]

\( i_k, j_l \): occurrences of \( q \) in \( I \) and \( D \). \( m \) : small integer, \( S \) similarity measure.
- Score between \( I \) and \( D \):

\[
\text{score}(I, D) = \sum_{q \in Gr(x) \cap Gr(y)} \text{score}_{I,D}(q)
\]
- Let \( D_{\text{max}} \) the database fingerprint with the maximal score, and \( i_{\text{max}}, j_{\text{max}} \) the locations in \( I \) and \( D_{\text{max}} \) providing the highest q gram score:

\[
\text{score}(I) = \text{score}(I[i_{\text{max}}, i_{\text{max}} + M], D[j_{\text{max}}, j_{\text{max}} + M])
\]
Bipartite Assignment

- Let us consider two sets $S$ (Engineers) and $P$ (Projects) of size $n$ and a cost function from $S \times T$ to $\mathbb{R}^+$. 
- $c(i, j)$ is the cost of engineer $i$ for project $j$.
- The aim of bipartite assignment algorithm is to determine a bijective mapping function $\psi : S \rightarrow T$ (a permutation) from Engineers to Projects which minimizes:
  $$\sum_{i \in S} c(i, \psi(i))$$  
  $\psi$ is an optimal assignment minimizing the mapping of Engineers to projects.
- This problem may be formalize as the one of determining a mapping on a bipartite weighted graph $G = (S \cup T, E, )$ where $E \subset S \times T$.
- This problem is solved in $O(n^3)$
Mathématical model

To each permutation \( \psi \) we associate the permutation matrix:

\[
x_{ij} = \begin{cases} 
1 & \text{If } \psi(i) = j \\
0 & \text{otherwise}
\end{cases}
\]

The assignment problem is then translated into the search of a permutation matrix \( x^* \) such that:

\[
x^* = \arg\min_x \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}
\]

such that:

- Each line of \( x \) sums to 1:

\[
\forall i \in \{1, \ldots, n\} \sum_{j=1}^{n} x_{ij} = 1
\]

- Each column of \( x \) sums to 1:

\[
\forall j \in \{1, \ldots, n\} \sum_{i=1}^{n} x_{ij} = 1
\]

- \( x \) is composed of 0 and 1: \( \forall (i, j) \in \{1, \ldots, n\}^2 x_{ij} \in \{0, 1\} \)
Problem reformulation

Let $A$ be the $2n \times n^2$ matrix defined by:

$$
\begin{pmatrix}
1 & 1 & 1 & \ldots & 1 \\
1 & 1 & 1 & \ldots & 1 \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
n & 1 & \ldots & 1 & 1 \\
n + 1 & 1 & \ldots & 1 & 1 \\
2n & 1 & \ldots & 1 & 1 \\
\end{pmatrix}
$$

Using $c$ and $x$ as vectors the problem becomes:

$$
\min_x c^t x \text{ such that } Ax = 1
$$

Note that $A$ is unimodular (the determinant of any squared sub matrix is 0, +1, or -1). Since $A$ is unimodular and 1 is a vector of integer, $x$ is a permutation matrix.
Dual problems

Our initial problem is equivalent to:

\[
\begin{align*}
\text{Min} & \quad c^T x \quad \Leftrightarrow \quad \text{Max} 1^T y \\
A x = 1, x \geq 0 & \quad A^T y \leq c
\end{align*}
\]

Let \( y = \begin{pmatrix} u \\ v \end{pmatrix} \) denotes the vector of \( 2n \) variables. Our problem is thus equivalent to:

\[
\text{Max } \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j \text{ with } u_i + v_j \leq c_{ij} \forall (i, j) \in \{1, \ldots, n\}^2
\]

Algorithms may be decomposed into:

1. Primal methods
2. Dual methods
3. Primal/Dual methods
A dual method

- Let us call a function \( y : (S \cup T) \to \mathbb{R} \) a potential if:

\[
y_i + y_j \leq c_{i,j} \quad \forall i \in S, j \in T.
\]

- The value of potential \( y \) is \( \sum_{i \in S \cup T} y_i \).
- The cost of each perfect matching is at least the value of each potential.
- The Hungarian method finds a perfect matching and a potential with equal cost/value which proves the optimality of both.
- An edge \( ij \) is called tight for a potential \( y \) if \( y_i + y_j = c_{i,j} \).
- Let us denote the subgraph of tight edges by \( G_y \).
- The cost of a perfect matching in \( G_y \) (if there is one) equals the value of \( y \).
- All edges in \( G_y \) are initially oriented from \( S \) to \( T \).
- Edges added to the matching \( M \subset G_y \) are oriented from \( T \) to \( S \). Initial value of \( M = \emptyset \).
- \( y \) is initialised to 0.
A dual method

Computation in terms of Bipartite Graph : Defs (2/2)

- We maintain the invariant that all the edges of $M$ are tight. We are done if $M$ is a perfect matching.
- In a general step,
  - let $R_S \subseteq S$ and $R_T \subseteq T$ be the vertices not covered by $M$.
  - Let $Z$ be the set of vertices reachable in $G_y$ from $R_S$ by a directed path only following edges that are tight.
  - If $R_T \cap Z \neq \emptyset$, then reverse the orientation of a directed path in $G_y$ from $R_S$ to $R_T$. Thus the size of the corresponding matching increases by 1.
  - If $R_T \cap Z = \emptyset$, then let:
    \[ \Delta := \min \{ c_{i,j} - y_i - y_j : i \in Z \cap S, j \in T \setminus Z \}. \]
    - Increase $y$ by $\Delta$ on the vertices of $Z \cap S$ and decrease $y$ by $\Delta$ on the vertices of $Z \cap T$. The resulting $y$ is still a potential. The graph $G_y$ changes, but it still contains $M$.
- We repeat these steps until $M$ is a perfect matching, in which case it gives a minimum cost assignment.
A dual method: Example

Defs.

- Let $R_S \subseteq S$ and $R_T \subseteq T$ be the vertices not covered by $M$.
- Let $Z$ be the set of vertices reachable in $\overrightarrow{G_y}$ from $R_S$ by a directed path only following edges that are tight.
- If $R_T \cap Z \neq \emptyset$, reverse the orientation of a directed path in $\overrightarrow{G_y}$ from $R_S$ to $R_T$.
- If $R_T \cap Z = \emptyset$, then let:
  \[
  \Delta := \min \{c_{i,j} - y - i - y - j : i \in Z \cap S, j \in T \setminus Z\}.
  \]
- Increase $y$ by $\Delta$ on $Z \cap S$ and decrease $y$ by $\Delta$ on $Z \cap T$.

Example

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>7</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>d</td>
<td>6</td>
<td>4</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>
A dual method: Example

Defns

- let $R_S \subseteq S$ and $R_T \subseteq T$ be the vertices not covered by $M$.
- Let $Z$ be the set of vertices reachable in $G_y$ from $R_S$ by a directed path only following edges that are tight.
- If $R_T \cap Z \neq \emptyset$, reverse the orientation of a directed path in $G_y$ from $R_S$ to $R_T$.
- If $R_T \cap Z = \emptyset$, then let:
  $$\Delta := \min \{ c_{i,j} - y_i - y_j : i \in Z \cap S, j \in T \setminus Z \}.$$
- Increase $y$ by $\Delta$ on $Z \cap S$ and decrease $y$ by $\Delta$ on $Z \cap T$.

Example: Iteration 0

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>7</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>d</td>
<td>6</td>
<td>4</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>
A dual method: Example

Defs

- Let $R_S \subseteq S$ and $R_T \subseteq T$ be the vertices not covered by $M$.
- Let $Z$ be the set of vertices reachable in $\overrightarrow{G_y}$ from $R_S$ by a directed path only following edges that are tight.
- If $R_T \cap Z \neq \emptyset$, reverse the orientation of a directed path in $\overrightarrow{G_y}$ from $R_S$ to $R_T$.
- If $R_T \cap Z = \emptyset$, then let:
  \[ \Delta := \min\{c_{i,j} - y_i - y_j : i \in Z \cap S, j \in T \setminus Z\}. \]
- Increase $y$ by $\Delta$ on $Z \cap S$ and decrease $y$ by $\Delta$ on $Z \cap T$.

Example: Iteration 1

\begin{align*}
E_y &= (c, 1); \\
R_S &= S; \\
R_T &= T; \\
Z &= S \cup \{1\};
\end{align*}

\begin{tabular}{|c|cccc|}
\hline
 & 1 & 2 & 3 & 4 \\
\hline
a & 4 & 6 & 8 & 10 \\
b & 3 & 2 & 5 & 9 \\
c & 1 & 7 & 8 & 3 \\
d & 6 & 4 & 10 & 5 \\
\hline
\end{tabular}
A dual method: Example

**Defs**

- let $R_S \subseteq S$ and $R_T \subseteq T$ be the vertices not covered by $M$.
- Let $Z$ be the set of vertices reachable in $\overrightarrow{G_y}$ from $R_S$ by a directed path only following edges that are tight.
- If $R_T \cap Z \neq \emptyset$, reverse the orientation of a directed path in $\overrightarrow{G_y}$ from $R_S$ to $R_T$.
- If $R_T \cap Z = \emptyset$, then let:
  \[
  \Delta := \min \{ c_{i,j} - y_i - y_j : i \in Z \cap S, j \in T \setminus Z \}.
  \]
- Increase $y$ by $\Delta$ on $Z \cap S$ and decrease $y$ by $\Delta$ on $Z \cap T$.

**Example: Iteration 2**

\[
\begin{array}{cccc}
 E_y & = & (1, c); \\
 R_S & = & \{a, b, d\}; \\
 R_T & = & \{2, 3, 4\}; \\
 Z & = & \{a, b, d\}; \\
 \Delta & = & 1 \\
\end{array}
\]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>7</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>d</td>
<td>6</td>
<td>4</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>
A dual method: Example

**Defs**

- let \( R_S \subseteq S \) and \( R_T \subseteq T \) be the vertices not covered by \( M \).
- Let \( Z \) be the set of vertices reachable in \( \overrightarrow{G_y} \) from \( R_S \) by a directed path only following edges that are tight.
- If \( R_T \cap Z \neq \emptyset \), reverse the orientation of a directed path in \( \overrightarrow{G_y} \) from \( R_S \) to \( R_T \).
- If \( R_T \cap Z = \emptyset \), then let:
  \[
  \Delta := \min\{c_{i,j} - y_i - y_j : i \in Z \cap S, j \in T \setminus Z\}.
  \]
- Increase \( y \) by \( \Delta \) on \( Z \cap S \) and decrease \( y \) by \( \Delta \) on \( Z \cap T \).

**Example: Iteration 3**

\[
\begin{array}{c|cccc}
 & 1 & 2 & 3 & 4 \\
\hline
a & 4 & 6 & 8 & 10 \\
b & 3 & 2 & 5 & 9 \\
c & 1 & 7 & 8 & 3 \\
d & 6 & 4 & 10 & 5 \\
\end{array}
\]
A dual method: Example

**Defs**

- Let \( R_S \subseteq S \) and \( R_T \subseteq T \) be the vertices not covered by \( M \).
- Let \( Z \) be the set of vertices reachable in \( \overrightarrow{G_y} \) from \( R_S \) by a directed path only following edges that are tight.
- If \( R_T \cap Z \neq \emptyset \), reverse the orientation of a directed path in \( \overrightarrow{G_y} \) from \( R_S \) to \( R_T \).
- If \( R_T \cap Z = \emptyset \), then let:

\[
\Delta := \min\{c_{i,j} - y_i - y_j : i \in Z \cap S, j \in T \setminus Z\}.
\]

- Increase \( y \) by \( \Delta \) on \( Z \cap S \) and decrease \( y \) by \( \Delta \) on \( Z \cap T \).

**Example: Iteration 4**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>7</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>d</td>
<td>6</td>
<td>4</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

\( E_y = \{(1, c), (2, b)\} \);  
\( R_S = \{a, d\} \);  
\( R_T = \{3, 4\} \);  
\( Z = \{a, d\} \);  
\( \Delta = 2 \)
A dual method: Example

**Defs**

- Let \( R_S \subseteq S \) and \( R_T \subseteq T \) be the vertices not covered by \( M \).
- Let \( Z \) be the set of vertices reachable in \( \overrightarrow{G_y} \) from \( R_S \) by a directed path only following edges that are tight.
- If \( R_T \cap Z \neq \emptyset \), reverse the orientation of a directed path in \( \overrightarrow{G_y} \) from \( R_S \) to \( R_T \).
- If \( R_T \cap Z = \emptyset \), then let:

  \[
  \Delta := \min \{ c_{i,j} - y_i - y_j : i \in Z \cap S, j \in T \setminus Z \}.
  \]

- Increase \( y \) by \( \Delta \) on \( Z \cap S \) and decrease \( y \) by \( \Delta \) on \( Z \cap T \).

**Example: Iteration 5**

\[
\begin{align*}
E_y &= \{(1, c), (2, b)\}; \\
R_S &= \{a, d\}; \\
R_T &= \{3, 4\}; \\
Z &= \{a, d, 1, c, 2, b\}; \\
\Delta &= 1
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>7</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>d</td>
<td>6</td>
<td>4</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>
Defns

- Let $R_S \subseteq S$ and $R_T \subseteq T$ be the vertices not covered by $M$.

- Let $Z$ be the set of vertices reachable in $\overrightarrow{G_y}$ from $R_S$ by a directed path only following edges that are tight.

- If $R_T \cap Z \neq \emptyset$, reverse the orientation of a directed path in $\overrightarrow{G_y}$ from $R_S$ to $R_T$.

- If $R_T \cap Z = \emptyset$, then let:

$$
\Delta := \min \{ c_{i,j} - y_i - y_j : i \in Z \cap S, j \in T \setminus Z \}.
$$

- Increase $y$ by $\Delta$ on $Z \cap S$ and decrease $y$ by $\Delta$ on $Z \cap T$.

Example: Iteration 6

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_y$</td>
<td>{(1, c), (2, b), (d, 4)}</td>
<td>4</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>$R_S$</td>
<td>{a, d};</td>
<td>\textbf{b}</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>$R_T$</td>
<td>{3, 4};</td>
<td>c</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>$Z$</td>
<td>{a, d, 1, c, 2, b, 4};</td>
<td>\textbf{d}</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>
A dual method: Example

Defs

- let $R_S \subseteq S$ and $R_T \subseteq T$ be the vertices not covered by $M$.
- Let $Z$ be the set of vertices reachable in $G_y$ from $R_S$ by a directed path only following edges that are tight.
- If $R_T \cap Z \neq \emptyset$, reverse the orientation of a directed path in $G_y$ from $R_S$ to $R_T$.
- If $R_T \cap Z = \emptyset$, then let:
  \[
  \Delta := \min \{c_{i,j} - y_i - y_j : i \in Z \cap S, j \in T \setminus Z\}.
  \]
- Increase $y$ by $\Delta$ on $Z \cap S$ and decrease $y$ by $\Delta$ on $Z \cap T$.

Example: Iteration 7

\[
\begin{array}{c}
S & T \\
\bullet a(5) & \bullet 1(-1) & \bullet 2(-1) & \bullet 3(0) & \bullet 4(0) \\
\bullet b(3) & \bullet 3(0) & \bullet 2(-1) & \bullet 1(-1) \\
\bullet c(2) & \bullet 3(0) & \bullet 2(-1) & \bullet 1(-1) \\
\bullet d(5) & \bullet 4(0) & \bullet 3(0) & \bullet 2(-1) \\
\end{array}
\]

\[
R_S = \{a\};
R_T = \{3\};
Z = \{a, 1, c\};
T - Z = \{2, 3, 4\};
\Delta = 1
\]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>7</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>d</td>
<td>6</td>
<td>4</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>
A dual method: Example

**Defs**

- let $R_S \subseteq S$ and $R_T \subseteq T$ be the vertices not covered by $M$.
- Let $Z$ be the set of vertices reachable in $\overrightarrow{G_y}$ from $R_S$ by a directed path only following edges that are tight.
- If $R_T \cap Z \neq \emptyset$, reverse the orientation of a directed path in $\overrightarrow{G_y}$ from $R_S$ to $R_T$.
- If $R_T \cap Z = \emptyset$, then let:

$$\Delta := \min \{c_{i,j} - y_i - y_j : i \in Z \cap S, j \in T \setminus Z\}.$$

- Increase $y$ by $\Delta$ on $Z \cap S$ and decrease $y$ by $\Delta$ on $Z \cap T$.

**Example: Iteration 8**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_S$</td>
<td>${a}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_T$</td>
<td>${3}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Z$</td>
<td>${a, 1, c, 4, d, 2}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T-Z$</td>
<td>${3}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\Delta$</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- $S$: a(6), b(3), c(3), d(5)
- $T$: 1(-2), 2(-1), 3(0), 4(0)

- $R_S = \{a\}$
- $R_T = \{3\}$
- $Z = \{a, 1, c, 4, d, 2\}$
- $T-Z = \{3\}$
- $\Delta = 2$
A dual method: Example

Defs

- let $R_S \subseteq S$ and $R_T \subseteq T$ be the vertices not covered by $M$.
- Let $Z$ be the set of vertices reachable in $\overrightarrow{G_y}$ from $R_S$ by a directed path only following edges that are tight.
- If $R_T \cap Z \neq \emptyset$, reverse the orientation of a directed path in $\overrightarrow{G_y}$ from $R_S$ to $R_T$.
- If $R_T \cap Z = \emptyset$, then let:

$$\Delta := \min\{c_{i,j} - y_i - y_j : i \in Z \cap S, j \in T \setminus Z\}.$$ 

- Increase $y$ by $\Delta$ on $Z \cap S$ and decrease $y$ by $\Delta$ on $Z \cap T$.

Example: Iteration 9

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>$b$</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>$c$</td>
<td>1</td>
<td>7</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>$d$</td>
<td>6</td>
<td>4</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>
Defns

- let $R_S \subseteq S$ and $R_T \subseteq T$ be the vertices not covered by $M$.
- Let $Z$ be the set of vertices reachable in $\overrightarrow{G_y}$ from $R_S$ by a directed path only following edges that are tight.
- If $R_T \cap Z \neq \emptyset$, reverse the orientation of a directed path in $\overrightarrow{G_y}$ from $R_S$ to $R_T$.
- If $R_T \cap Z = \emptyset$, then let:
  \[
  \Delta := \min\{c_{i,j} - y_i - y_j : i \in Z \cap S, j \in T \setminus Z\}.
  \]
- Increase $y$ by $\Delta$ on $Z \cap S$ and decrease $y$ by $\Delta$ on $Z \cap T$.

Example: Iteration 10

\[
C = C(c, 1) + C(b, 2) + C(d, 4) + C(a, 3)
\]
\[
= 1 + 2 + 5 + 8
\]
\[
= 16
\]
Pimal/Dual methods

A pair of solutions feasible both in the primal and the dual is optimal if and only if:

\[ x_{ij}(c_{ij} - u_i - v_j) = 0 \]

Moreover, given \((u_i)_{i \in \{1,\ldots,n\}}\) and \((v_j)_{j \in \{1,\ldots,n\}}\) solving LSAP for \(c_{ij}\) is equivalent to solve it for \(\overline{c}_{ij} = c_{ij} - u_i - v_j\). Indeed:

\[
\sum_{i=1}^{n} \sum_{j=1}^{n} (c_{ij} - u_i - v_j)x_{ij} = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}x_{ij} - \sum_{i=1}^{n} u_i \sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} v_j \sum_{i=1}^{n} x_{ij} \\
= \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}x_{ij} - \sum_{i=1}^{n} u_i - \sum_{j=1}^{n} v_j \text{ (constant)}
\]

The main difference is that if \(\overline{c}_{ij}\) contains a sufficient amount of 0, the primal problem reduces to find a set of \(n\) independent 0.
A primal/Dual method

**Algorithm**

**Initialisation:** Subtract the minimum of each line and then the minimum of each column.

### Example

<table>
<thead>
<tr>
<th>$u_i$</th>
<th>$v_j$</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>b</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>c</td>
<td>1</td>
<td>7</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>d</td>
<td>6</td>
<td>4</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$u_i$</th>
<th>$v_j$</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>a</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>c</td>
<td>0</td>
<td>6</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>d</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$u_i$</th>
<th>$v_j$</th>
<th>0</th>
<th>0</th>
<th>3</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>a</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>c</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>d</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
A primal/Dual method

**Hungarian/Munkres algorithm**

**Algorithm**

**Initialisation:** Subtract the minimum of each line and then the minimum of each column.

**Step 1:** Mark each 0 not in the same row or column of a 0 already marked. If $n$ zeros marked: Stop.

**Example**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>3</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>c</td>
<td>0</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>d</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

36 / 155
A primal/Dual method

Hungarian/Munkres algorithm

Algorithm

Step 2: Cover each column containing a selected zero.
- For each non covered zero, mark it by a prime
  - If there is a selected zero on the line uncover the column and cover the line
  - If there is no selected zero on the line, we do not have selected enough independent zero. Goto step 3.
- If there is no more uncovered zero. Goto step 4.

Example

<table>
<thead>
<tr>
<th></th>
<th>×</th>
<th>×</th>
<th>×</th>
<th></th>
<th></th>
<th>×</th>
<th>×</th>
<th>×</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4 a</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>4 a</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2 b</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>2 b</td>
<td>1</td>
<td>0</td>
<td>0’</td>
<td>6</td>
</tr>
<tr>
<td>1 c</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td>1 c</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4 d</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>4 d</td>
<td>2</td>
<td>0’</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

|    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| 0  | 0  | 3 | 1  | 0  | 0  | 3 | 1  |
| 1  | 2 | 3 | 4  | 1  | 2 | 3 | 4  |

<table>
<thead>
<tr>
<th></th>
<th>×</th>
<th>×</th>
<th>×</th>
<th></th>
<th></th>
<th>×</th>
<th>×</th>
<th>×</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4 a</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>4 a</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2 b</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>2 b</td>
<td>1</td>
<td>0</td>
<td>0’</td>
<td>6</td>
</tr>
<tr>
<td>1 c</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td>1 c</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4 d</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>4 d</td>
<td>2</td>
<td>0’</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|
| 0  | 0  | 3 | 1  | 0  | 0  | 3 | 1  |
| 1  | 2 | 3 | 4  | 1  | 2 | 3 | 4  |
Algorithm

**Step 2:** Cover each column containing a selected zero.

**Step 4:** Take the minimal value of uncovered elements found in step 2. Add this value to each covered line and subtract it from each non-covered column. Return to step 1.

Example

\[
\begin{array}{cccc}
\times & 0 & 0 & 3 & 1 \\
0 & 2 & 1 & 5 \\
1 & 2 & 3 & 4 \\
4 & 0 \\
\hline
2 & 1 & 0 & 0' & 6 \\
1 & 0 & 6 & 4 & 1 \\
4 & 2 & 0' & 3 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
\times & 0 & 0 & 3 & 1 \\
1 & 2 & 3 & 4 \\
4 & 0 & 2 & 1 & 5 \\
\hline
1 & 2 & 1 & 1 & 7 \\
1 & 0 & 6 & 4 & 1 \\
3 & 3 & 1 & 4 & 1 \\
\times & 3 & 3 & 1 & 4 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 1 & 4 & 2 \\
1 & 2 & 3 & 4 \\
4 a & 0 & 1 & 0 & 4 \\
\hline
1 b & 2 & 0 & 0 & 6 \\
1 c & 0 & 5 & 3 & 0 \\
3 d & 3 & 0 & 3 & 0 \\
\times & 3 d & 3 & 0 & 3 & 0 \\
\end{array}
\]
**Algorithm**

**Step 2:** Cover each column containing a selected zero.
- For each non covered zero, mark it by a prime
  - If there is a selected zero on the line uncover the column and cover the line
  - If there is no selected zero on the line, we do not have selected enough independent zero. Goto step 3.
- If there is no more uncovered zero. Goto step 4.

**Example**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>4</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td>0</td>
<td>1</td>
<td>0’</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>2</td>
<td>0</td>
<td>0’</td>
</tr>
<tr>
<td>1</td>
<td>c</td>
<td>0’</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>d</td>
<td>3</td>
<td>0’</td>
<td>3</td>
</tr>
</tbody>
</table>

0 1 0’ 4
×
1 0 0’ 6
×
0’ 5 3 0
×
A primal/Dual method

Hungarian/Munkres algorithm

Algorithm

Step 2: Cover each column containing a selected zero.

Step 3: Let \( z_0 \) be the only uncovered 0 and \( z_1 \) the selected 0 on its column. Let \( z_i \) (i odd ) the 0’ on the line of \( z_{i-1} \) and \( z_i \) (i even) the selected 0 on the column of \( z_{i-1} \) if it exists (otherwise we stop). The serie \( z_i \) contains one more 0’ than selected 0. Exchange 0’ and selected 0. Removes the primes of zeros and the lines and columns covering. Return to step 1

Example

\[
\begin{array}{c|cccc}
& 0 & 1 & 4 & 2 \\
\hline
1 & 1 & 2 & 3 & 4 \\
4 & a & 0 & 1 & 0' \\
1 & b & 2 & 0 & 0 \\
1 & c & 0' & 5 & 3 \\
3 & d & 3 & 0 & 3 \\
\end{array}
\]

\[
\begin{array}{c|cccc}
& 0 & 1 & 4 & 2 \\
\hline
1 & 1 & 2 & 3 & 4 \\
4 & a & 0 & 1 & 0 \\
1 & b & 2 & 0 & 0 \\
1 & c & 0 & 5 & 3 \\
3 & d & 3 & 0 & 3 \\
\end{array}
\]

\[
\begin{array}{c|cccc}
& 0 & 1 & 4 & 2 \\
\hline
1 & 1 & 2 & 3 & 4 \\
4 & a & 0 & 1 & 0 \\
1 & b & 2 & 0 & 0 \\
1 & c & 0 & 5 & 3 \\
3 & d & 3 & 0 & 3 \\
\end{array}
\]

\[
\begin{array}{c|cccc}
& 0 & 1 & 4 & 2 \\
\hline
1 & 1 & 2 & 3 & 4 \\
4 & a & 0 & 1 & 0 \\
1 & b & 2 & 0 & 0 \\
1 & c & 0 & 5 & 3 \\
3 & d & 3 & 0 & 3 \\
\end{array}
\]

\[
\begin{array}{c|cccc}
& 0 & 1 & 4 & 2 \\
\hline
1 & 1 & 2 & 3 & 4 \\
4 & a & 0 & 1 & 0 \\
1 & b & 2 & 0 & 0 \\
1 & c & 0 & 5 & 3 \\
3 & d & 3 & 0 & 3 \\
\end{array}
\]

\[
\begin{array}{c|cccc}
& 0 & 1 & 4 & 2 \\
\hline
1 & 1 & 2 & 3 & 4 \\
4 & a & 0 & 1 & 0 \\
1 & b & 2 & 0 & 0 \\
1 & c & 0 & 5 & 3 \\
3 & d & 3 & 0 & 3 \\
\end{array}
\]

\[
\begin{array}{c|cccc}
& 0 & 1 & 4 & 2 \\
\hline
1 & 1 & 2 & 3 & 4 \\
4 & a & 0 & 1 & 0 \\
1 & b & 2 & 0 & 0 \\
1 & c & 0 & 5 & 3 \\
3 & d & 3 & 0 & 3 \\
\end{array}
\]

\[
\begin{array}{c|cccc}
& 0 & 1 & 4 & 2 \\
\hline
1 & 1 & 2 & 3 & 4 \\
4 & a & 0 & 1 & 0 \\
1 & b & 2 & 0 & 0 \\
1 & c & 0 & 5 & 3 \\
3 & d & 3 & 0 & 3 \\
\end{array}
\]

\[
\begin{array}{c|cccc}
& 0 & 1 & 4 & 2 \\
\hline
1 & 1 & 2 & 3 & 4 \\
4 & a & 0 & 1 & 0 \\
1 & b & 2 & 0 & 0 \\
1 & c & 0 & 5 & 3 \\
3 & d & 3 & 0 & 3 \\
\end{array}
\]
**Bipartite Assignment algorithm: Matrix version**

**Summary**

**Initialisation:** Substract the minimum of each line and then the minimum of each column.

**Step 1:** Mark each 0 not in the same row or column of a 0 already marked. If $n$ zeros marked: Stop.

**Step 2:** Cover each column by a selected zero.
- For each non covered zero, mark it by a prime
  - If there is a selected zero on the line uncover the column and cover the line
  - If there is no selected zero on the line, we do not have selected enough independent zero. Goto step 3.
- If there is no more uncovered zero. Goto step 4.

**Step 3:** Let $z_0$ be the only uncovered 0 and $z_1$ the selected 0 on its column. Let $z_i$ (i odd) the 0' on the line of $z_{i-1}$ and $z_i$ (i even) the selected 0 on the column of $z_{i-1}$ if it exists (otherwise we stop). Exchange 0’ and selected 0. Removes the primes of zeros and the lines and columns covering. Return to step 1.

**Step 4:** Take the minimal value of uncovered elements found in step 2. Add this value to each covered line and substract it to each non covered column. Return to step 1.
Unfortunately not all problems may be formulated using uni dimensional relationships between objects.

We need a more global description of relationships between objects.
Graphs

- $G = (V, E)$
  - $V$ set of vertices (or nodes): set of objects
  - $E$ set of edges: set of relationships between objects

- Let $G = (V, E)$ be a graph describing a scene coming from a segmentation, a skeleton, a document...

- Let $G' = (V', E')$ be a graph describing a model
  - Mean object of a class,
  - Perfect theoretical object (known or obtained during previous acquisitions).

- Questions:
  - Does $G, G'$ encode a same object?
  - Does $G$ describe a part of $G'$?
Graph matching terminology (1/5)

- Let X and Y denote objects/scenes, we want to know if:

\[ X \cong Y \text{ or } X \subseteq Y. \]

- Graph isomorphism \( G = (V, E), G' = (V', E') \) (\( X \cong Y \))
  - \( |V| = |V'| \) and
  - it exists \( \phi : V \to V' \) bijective such that:
    \[
    (v_1, v_2) \in E \iff (\phi(v_1), \phi(v_2)) \in E'
    \]

- Partial sub graph isomorphism (\( X \subseteq Y \))
  - \( |V| \leq |V'| \) and
  - it exists \( \phi : V \to V' \) injective such that:
    \[
    (v_1, v_2) \in E \Rightarrow (\phi(v_1), \phi(v_2)) \in E'
    \]
Graph matching terminology (2/5)

- Sub graph isomorphism
  - Same as partial sub graph isomorphism but with the additional constraint:
    \[ \forall (v_1, v_2) \in V^2 \ (v_1, v_2) \notin E \Rightarrow (\phi(v_1), \phi(v_2)) \notin E' \]
    
    We have thus:
    \[ (\phi(v_1), \phi(v_2)) \in E' \iff (v_1, v_2) \in E \]
  - A partial sub graph isomorphism which is not a sub graph isomorphism.
Graph matching terminology (3/5)

- Let $X$ (image) and $Y$ (model), we want to know if it exists $Z$ such that:

  $$Z \subseteq X \text{ and } Z \subseteq Y$$

- Maximum common partial sub graph (mcps).
  - Graph of maximal size (in terms of number of vertices), being a partial sub graph of $G$ and $G'$.

- Maximum common sub graph (mcs)
  - Same than mcps but isomorphism of sub graph instead of partial sub graph isomorphism.
Maximal vs maximum sub graph

- (Partial) maximum or maximal sub graph?
  - A (partial) common sub graph of $G$ and $G'$, is said to be **maximal**, if we cannot add to it any vertex or edge without breaking the isomorphism property.
  - A (partial) common sub graph is said to be **maximum** if any (partial) common sub graph of $G$ and $G'$ contains less vertices.
(Sub)graph isomorphisms: typology of approaches

- Graph/Sub graph isomorphism
  - NP complete problem
- Heuristics to obtain solution (may be sub optimal)
- Two approaches:
  - Symbolic or algorithmic:
    - Traverse the set of potential solutions with some heuristics to reject a priori some solutions.
    - Good control over the result.
    - Main actors: Horst Bunke, Marcello Pellilo, Mario Vento, Pasquale Foggia, …
  - Numerical approaches:
    - Define the problem in terms of minimization/maximization of a function/energy.
    - All the tools of numerical optimization are available.
    - Main actors: Edwin Hancock, Kitler, Sven Dickinson, …
From bipartite Assignment to graph matching.

Rem : Image taken from B. Raynal PhD.
Labeled Graphs

- **Definitions:**
  - **A labeled graph** \( G = (V, E, \mu, \nu, L_v, L_e) \)
    \[
    \begin{align*}
    \mu &: V \rightarrow L_v \quad \text{vertex’s label function} \\
    \nu &: E \rightarrow L_e \quad \text{edge’s label function}
    \end{align*}
    \]
  - **A labeled sub graph** \( G_s = (V_s, E_s, \mu_s, \nu_s, L_v, L_e) \) of \( G = (V, E, \mu, \nu, L_v, L_e) \).
    - \( \mu_s \) and \( \nu_s \) restriction of \( \mu \) and \( \nu \) to \( V_s \subset V \) and \( E_e \subset E \) (with \( E_s = E \cap V_s \times V_s \)).
  - The **adjacency matrix** \( M = (m_{i,j}) \) of a graph \( G = (V, E, \mu, \nu, L_v, L_e) \) is defined by:
    1. \( \forall i \in \{1, \ldots, n\} \) \( m_{i,i} = \mu(v_i) \)
    2. \( \forall (i, j) \in \{1, \ldots, n\}^2, i \neq j \)
      \[
      m_{i,j} = \begin{cases} 
      \nu((v_i, v_j)) & \text{if } (v_i, v_j) \in E \\
      0 & \text{else}
      \end{cases}
      \]
### Adjacency/Permutation matrices

Consider two graphs with vertices labeled `a`, `b`, and `c`, and edges labeled `e1`, `e2`, and `e3`.

**Graph 1:**
- Vertices: `a, b, c`
- Edges: `a - e1 - b, e2 - e2 - c`

**Graph 2:**
- Vertices: `a, b, c`
- Edges: `a - e1 - b, e2 - e2 - c`

**Adjacency Matrices:**
- **Graph 1:**
  
  \[
  M = \begin{pmatrix}
  1 & 2 & 3 \\
  1 & a & e_2 & e_1 \\
  2 & e_2 & b & 0 \\
  3 & e_1 & 0 & c \\
  \end{pmatrix}
  \]

- **Graph 2:**
  
  \[
  M' = \begin{pmatrix}
  1 & 2 & 3 \\
  1 & 0 & 1 & 0 \\
  2 & 1 & 0 & 0 \\
  3 & 0 & 0 & 1 \\
  \end{pmatrix}
  \]

**Permutation Matrix:**
- **Graph 1:**
  
  \[
  P = \begin{pmatrix}
  1 & 2 & 3 \\
  1 & 2 & 3 \\
  \end{pmatrix}
  \]

**Permutation and Adjacency Matrices:**
- **Graph 1:**
  
  \[
  M' = PMP^t = \begin{pmatrix}
  1 & 2 & 3 \\
  1 & b & e_2 & e_1 \\
  2 & e_2 & a & 0 \\
  3 & e_1 & 0 & c \\
  \end{pmatrix}
  \]
Graph Isomorphism and Permutation matrices

- Two graphs $G_1$ and $G_2$ with matrices $M_1$ and $M_2$ are said to be **isomorphic** iff it exists a permutation matrix $P$ such that:

  $$M_2 = P M_1 P^t$$

- It exists a sub graph isomorphism between $G_1$ and $G_2$ iff it exists $S \subset G_2$ such that $G_1$ and $S$ are isomorphic ($S = (S, E \cap S \times S, \mu|_S, \nu|_S, L_v, L_e)$).

- Let $M = (m_{i,j})$ be a $n \times n$ permutation matrix

  $$\forall (k, m) \in \{1, \ldots, n\}^2 \ S_{k,m}(M) = (m_{i,j})_{i \in \{1, \ldots, k\}, j \in \{1, \ldots, m\}}$$

  - $S_{k,k}(M)$ is the adjacency matrix of the sub graph restricted to the $k$ first vertices.
Let $G_1$ and $G_2$ be two graphs with adjacency matrices $M_1$ and $M_2$

- $M_1 : m \times m,$
- $M_2 : n \times n$ with $m \leq n.$

It exists a sub graph isomorphism between $G_1$ and $G_2$ iff it exists a $n \times n$ permutation matrix $P$ such that:

$$M_1 = S_{m,m}(PM_2P^t)$$

Remark:

$$M_1 = S_{m,m}(PM_2P^t) = S_{m,n}(P)M_2 (S_{m,n}(P))^t$$
- A state: A partial matching
- Method: explore successively the different states
- Distinction between different methods:
  - Transition from one state to an other
  - Heuristics to avoid infinite loops (consider twice the same state),
  - Heuristics to restrict the search space
Examples Strings Bipartitite Assignment Graph Terminology Graph Matching Graph Edit distance

SSR : Algorithme de Cordella 2001 (VF2)

- **Notations:**
  - $G_1 = (V_2, E_2)$, $G_2 = (V_2, E_2)$ two *oriented* graphs,
  - We search for either:
    - an isomorphism between $G_1$ and $G_2$,
    - a sub graph isomorphism between $G_2$ and $G_1$ ($|V_2| \leq |V_1|$)
  - state $s$,
  - $M(s)$ partial matching associated to $s$,
  - $M_1(s)$ vertices of $M(s)$ in $V_1$,
  - $M_2(s)$ vertices of $M(s)$ in $V_2$
  - $P(s)$ set of couples (in $V_1 \times V_2$) candidates to an inclusion in $s$,
  - $F(s, n, m)$ predicate: does the addition of $(n, m)$ to $s$ defines a partial isomorphism?
  - $T_1^{in}(s)(T_1^{out}(s))$ set of vertices of $G_1$ predecessors (successors) of a vertex of $M_1(s)$.
  - $T_2^{in}(s)(T_2^{out}(s))$ set of vertices of $G_2$ predecessors (successors) of a vertex of $M_2(s)$. 
VF2 notations: Example

\[ G_1 \]

- \( M(s) = (a, 1) \)
- \( M_1(s) = \{ a \}, M_2(s) = \{ 1 \} \)
- \( T_1^{in}(s) = \{ b \}; T_1^{out}(s) = \{ c \}; \)
- \( T_2^{in}(s) = \{ 2 \}; T_2^{out}(s) = \{ 3 \}; \)

\[ G_2 \]
VF2 algorithm

procedure MATCHING(char s) a matching
\[ s_0 \text{ initial state s.t. } M(s_0) = \emptyset \]

if \( M(s) \) contains all vertices of \( G_2 \) then
  return \( M(s) \)
else
  compute \( P(s) \)
  for each \((n, m) \in P(s)\) do
    if \( F(s, n, m) \) then
      compute \( s' \) after the addition of \((n, m)\) to \( M(s) \)
      MATCHING(s')
    end if
  end for
end if
Restauration of data structures
end procedure
VF2

Examples
Strings
Bipartitite Assignement
Graph Terminology
Graph Matching
VF2

Computation of $P(s)$

- If $T_1^{out}(s)$ and $T_2^{out}(s)$ non empty

$$P(s) = T_1^{out}(s) \times \{\min T_2^{out}(s)\}$$

min: any order relationship: increasing order of insertion of $G_2$’s vertices (avoid to consider $\neq$ paths leading to a same state).

- Else If $T_1^{in}(s)$ et $T_2^{in}(s)$ non empty

$$P(s) = T_1^{in}(s) \times \{\min T_2^{in}(s)\}$$

- Else if $T_1^{in}(s) = T_2^{in}(s) = T_1^{out}(s) = T_2^{out}(s) = \emptyset$

$$P(s) = (V_1 - M_1(s)) \times \{\min(V_2 - M_2(s))\}$$

- Rem: If one of the set $T^{in}(s)$ and $T^{out}(s)$ is empty and not the other $M(s)$ cannot lead to a matching.
Computation of $F(s, n, m)$

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

- $R_{pred}, R_{succ}$: Does $M(s')$ defines a matching?
- $R_{in}, R_{out}$: can we build a matching the step after?
- $R_{new}$: Can I obtain a matching (one day)?
VF2

matching of predecessors: $R_{pred}$

- $R_{pred}(s, m, n)$: predecessors match

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

$$n' \in M_1(s) \iff m' \in M_2(s)$$

\[ \downarrow \]

\[ n \iff m \]
$R_{succ}(s, m, n)$: successors match

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

\[
\begin{array}{c c c}
  n & \leftrightarrow & m \\
  \downarrow & & \downarrow \\
  n' \in M_1(s) & \leftrightarrow & m' \in M_2(s)
\end{array}
\]
VF2

consistency according to $T^{in}(s)$: $R_{in}$

- Successors (predecessors) of $n$ and $m$ must match locally
  - $R_{in}(s, n, m)$

$$
\left( \left| T^{in}_1(s) \cap Succ(G_1, n) \right| \geq \left| T^{in}_2(s) \cap Succ(G_2, m) \right| \right) \land \\
\left( \left| T^{in}_1(s) \cap Pred(G_1, n) \right| \geq \left| T^{in}_2(s) \cap Pred(G_2, m) \right| \right)
$$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$\in G_1(s')$</th>
<th>$n$</th>
<th>$\in G_1(s')$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td>$\uparrow$</td>
<td>$\uparrow$</td>
</tr>
<tr>
<td>$n'$</td>
<td>$\rightarrow$</td>
<td>$n''$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>$\in T^{in}_1(s)$</td>
<td>$\in G_1(s)$</td>
<td>$\in T^{in}_1(s)$</td>
<td>$\in G_1(s)$</td>
</tr>
</tbody>
</table>
VF2

Consistency according to $T_{1}^{out}(s)$: $R_{out}$

- $R_{out}(s, n, m)$

\[
\begin{align*}
(|T_{1}^{out}(s) \cap \text{Succ}(G_1, n)| & \geq |T_{2}^{out}(s) \cap \text{Succ}(G_2, m)|) \land \\
(|T_{1}^{out}(s) \cap \text{Pred}(G_1, n)| & \geq |T_{2}^{out}(s) \cap \text{Pred}(G_2, m)|)
\end{align*}
\]

<table>
<thead>
<tr>
<th>$n$</th>
<th>$\in G_1(s')$</th>
<th>$n \in G_1(s')$</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓</td>
<td>$n'$</td>
<td>$n''$</td>
</tr>
<tr>
<td>$\in T_{1}^{out}(s)$</td>
<td>$\in G_1(s)$</td>
<td>$\in T_{1}^{out}(s)$</td>
</tr>
</tbody>
</table>

- Replace $\geq$ by $=$ for the graph isomorphism
**Predicate $R_{new}$**

- Successors and predecessors must match outside sets $M_i(s), T^{in}_i(s)$ and $T^{out}_i(s)$, $i = 1, 2$.
  - $R_{new}(s, n, m)$
    
    $$(|N_1(s) \cap Succ(G_1, n)| \geq |N_2(s) \cap Succ(G_2, m)|) \land$$
    $$(|N_1(s) \cap Pred(G_1, n)| \geq |N_2(s) \cap Pred(G_2, m)|)$$

- with:
  - $N_1(s) = V_1 - M_1(s) - T^{in}_1(s) \cup T^{out}_1(s)$: all that remains to be seen in $G_1$.
  - $N_2(s) = V_2 - M_2(s) - T^{in}_2(s) \cup T^{out}_2(s)$: all that remains to be seen in $G_2$. 
**Example (1/3)**

\[ G_1 \]

- \( M(s) = (a, 1) \)
- \( M_1(s) = \{a\}, M_2(s) = \{1\} \)
- \( T_1^{in}(s) = \{b\}; T_1^{out}(s) = \{c\}; \)
- \( T_2^{in}(s) = \{2\}; T_2^{out}(s) = \{3\}; \)
- \( P(s) = (c, 3) \)
- \( Pred(G_1, c) = \{a\}; Succ(G_1, c) = \{b, e\} \)
- \( Pred(G_2, 3) = \{1\}; Succ(G_2, 3) = \{2\} \)
- \( F(s, c, 3) = true. \)
\[ G_1 \]

- \( M(s') = \{(a, 1), (c, 3)\} \)
- \( M_1(s') = \{a, c\}, M_2(s) = \{1, 3\} \)
- \( T_1^{in}(s) = \{b\}; T_1^{out}(s) = \{b, e\}; \)
- \( T_2^{in}(s) = \{2\}; T_2^{out}(s) = \{2\}; \)
- \( P(s) = (b, 2), (e, 2) \)
- \( \text{Pred}(G_1, e) = \{c\}; \text{Succ}(G_1, e) = \{d\} \)
- \( \text{Pred}(G_2, 2) = \{3\}; \text{Succ}(G_2, 2) = \{1\} \)
- \( (e, 2) \) violate predicate \( R_{\text{succ}}(s', e, 2) \). Indeed:
  - \( 1 \in M_2(s) \cap \text{Succ}(G_2, 2) \) or \( \text{Succ}(G_1, e) \cap M_1(s) = \emptyset \)
VF2

Examples
Strings
Bipartitite Assignement
Graph Terminology
Graph Matching
Graph Edit distance

Example (3/3)

We come up with the matching: \( M(s'') = \{(a, 1), (c, 3), (b, 2)\} \) and we are done since we cover all the vertices of \( G_2 \).
VF2

- Complexity: \( N = |V_1| + |V_2| \)

<table>
<thead>
<tr>
<th>Complexity</th>
<th>VF2 Best case</th>
<th>VF2 Worse case</th>
<th>Ullman Best case</th>
<th>Ullman Worse case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>( O(N^2) )</td>
<td>( O(N!N) )</td>
<td>( O(N^3) )</td>
<td>( O(N!N^2) )</td>
</tr>
<tr>
<td>Space</td>
<td>( O(N) )</td>
<td>( O(N) )</td>
<td>( O(N^3) )</td>
<td>( O(N^3) )</td>
</tr>
</tbody>
</table>

- Usable for large graphs (up to 1000 vertices).
Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the association graph $G = (V, E)$ of $G_1$ and $G_2$ is defined by:

- **vertices:**
  
  $$V = V_1 \times V_2$$

- **Edges:**
  
  $$E = \{(i, h, (j, k)) \in V \times V | i \neq j, h \neq k \text{ and } (i, j) \in E_1 \iff (h, k) \in E_2\}$$
Association graph

### Input Graphs

1. \(1 - a\) connected to \(2 - b\) and \(4 - e\)
2. \(1 - b\) connected to \(2 - b\) and \(4 - b\)
3. \(1 - c\) connected to \(2 - c\) and \(4 - c\)
4. \(1 - d\) connected to \(2 - d\) and \(4 - d\)
5. \(1 - e\) connected to \(2 - e\) and \(4 - e\)

### Association Graph

1. \(1 - \ast\) connects \(1-a\), \(1-b\), \(1-c\), \(1-d\), and \(1-e\)
2. \(2 - \ast\) connects \(2-a\), \(2-b\), \(2-c\), \(2-d\), and \(2-e\)
3. \(1 - \ast\) connects \(1-a\), \(1-b\), \(1-c\), \(1-d\), and \(1-e\)
4. \(4 - \ast\) connects \(4-a\), \(4-b\), \(4-c\), \(4-d\), and \(4-e\)
Sub graph $G$ restricted to $\{(1, a), (2, b), (3, c)\}$ (blue lines) is complete.

- All matching are consistent
MCS and cliques

- A complete sub graph of a graph $G$ is called a **clique** of $G$.
- Maximum/Maximal cliques are defined the same way as common subgraphs.
- The clique-number of $G$, $\omega(G)$ is the size (in vertices) of the maximum clique.
- Théorem: Let $G_1$ and $G_2$ be two graphs and $G$ their association graph. It exists a bijective relationship between:
  - maximal/maximum cliques of $G$ and
  - maximal/maximum common subgraphs of $G_1$ and $G_2$.
- Hence: Computing cliques of the association graph is **equivalent** to compute the common sub graphs.
Based on a pre computation of all cliques of size $i$.
- $i = 1$: vertices,
- $i = 2$: couple of vertices.

Notations:
- Neighborhood:
  \[ N_j = \{ k \in V \mid (j, k) \in E \} \]
- Candidates to the adjunction to clique $K$:
  \[ C_0(K) = \{ j \in V - K \mid \forall k \in K (j, k) \in E \} = \bigcap_{k \in K} N_k \]
**SM^i**: the algorithm

```plaintext
procedure SM^i\(\)\(\)\(\)\(\)\(\)(graph G)\(\)\(\)
  Q: clique with cardinal i, \(KK, K^*, \text{max}\)
  \(K = \emptyset;\)
  \(K^* = \emptyset;\)
  max = 0;
  while Q ≠ \(\emptyset\) do
    H = pop(Q)
    K = H
    while \(C_0(K) \neq \emptyset\) do
      \(l = \arg\max_{j \in C_0(K)} |C_0(K) \cap N_j|\)
      \(K = K \cup \{l\}\)
    end while
    \(K = K \cup K\)
  end while
end procedure
```

if \(|K| > \text{max}\) then
  \(\text{max} = |K|\)
  \(K^* = K\)
end if
$SM^i$ : Comments

- Try to build as many maximal cliques as initial cliques of size $i$.
- Converge toward a local optimum for each initial clique.
- $SM^2$ more efficient but slower than $SM^1$.
- $SM^1_{-SWAP}$ compromise between $SM^1$ and $SM^2$
  - Aim: explore the search space around local optimums.
**SM^1_Swap**: Notations

- Candidate to an exchange:

  \[ C_1(K) = \{ j \in V - K \mid |N_j \cap K| = |K| - 1 \} \]

- \( j \in C_1(K) \) iff only one vertex of \( K \) forbids its integration to the clique.

- Let \( l \in C_1(K) \) and \( k_l \in K \) such that \( (l, k_l) \not\in E \). If we add \( l \) to \( K \), we must remove \( k_l \).

  \[ K = K \cup \{l\} - \{k_l\} \]
$SM^1_{SWAP}$ : Choosing the vertex to add

$$l = \arg \max_{j \in C_0(K)} |C_0(K) \cap N_j| \text{ ou } l = \arg \max_{j \in C_0(K) \cup C_1(K)} |C_0(K) \cap N_j|$$

- The choice of a vertex in $C_1(K)$ allows to move away from the local optimum but:
  - This choice does not necessarily improve the optimum
  - does not allow us to get closer to the convergence
  - may induce infinite loops.
- We restrict ourselves to $C_0(K)$:
  - for a fixed number of iterations ($START_{SWAP}$),
  - when the number of exchanges is greater than a multiple $T$ of $|K|$, 
  - when the selected vertex is the one removed by the last exchange
**SM\(^1\_SWAP\):** selection algorithm

```plaintext
function select(G, K, last_swap)
    l : vertex;
    if n_swap \leq T|K| and |K| \geq START\_SWAP then
        l = \arg \max_{j \in C_0(K) \cup C_1(K)} |C_0(K) \cap N_j|
        if l = last_swap then
            l = \arg \max_{j \in C_0(K)} |C_0(K) \cap N_j|
        end if
    else
        l = \arg \max_{j \in C_0(K)} |C_0(K) \cap N_j|
    end if
    return l
end function
```

- We suppose that \( \arg \max_{j \in C} \) return \( \emptyset \) if \( C \) is empty.
$SM^1_{SWAP}$: Algorithm

```plaintext
procedure $SM^1_{SWAP}$(graph $G = (V, E)$)
    int max = 0;
    list of cliques $\mathcal{K} = \emptyset$,
    optimal clique $K^* = \emptyset$,
    queue $W = V$
    while $W \neq \emptyset$ do
        $h = \text{pop}(W)$
        $K = \{h\}$
        $n\_swap = 0; last\_swap = \emptyset$
        $l = \text{select}(G, K, last\_swap)$
        while $l \neq \emptyset$ do
            if $l \in C_0(K)$ then
                $K = K \cup \{l\}$
            else
                $n\_swap + = 1$
                $last\_swap = k_l$
                $K = K \cup \{l\} - \{k_l\}$
                push(W, $k_l$)
            end if
            $l = \text{select}(G, K, last\_swap)$
        end while
        $K = K \cup \{K\}$
        if $|K| > max$ then
            max = $|K|$
            $K^* = K$
        end if
    end while
end procedure
```

$n\_swap + = 1$
$last\_swap = k_l$
$K = K \cup \{l\} - \{k_l\}$
push(W, $k_l$)

end if
SM1\_SWAP: Performances

- clique size/ max size: \(|K|/\omega(G)|

Execution times (seconds):

\[\begin{array}{c|c|c|c}
\text{density} & 0.1 & 0.2 & 0.3 & 0.4 & 0.5 & 0.6 & 0.7 & 0.8 & 0.9 & 1 \\
\hline
\text{time(s)} & 0 & 0.2 & 0.4 & 0.6 & 0.8 & 1 \\
\end{array}\]
Optimisation methods

1. Encode the graph by a matrix
2. Transform a graph problem into the maximisation/minimisation of some expression (using matrix encoding)
3. Choose an optimisation method
Let us consider $G = (V, E)$ et $C \subset V$ with $|V| = n$

Characteristic vector of $C$:

$$x^C = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \text{ with } x_i = \begin{cases} \frac{1}{|C|} & \text{if } i \in C \\ 0 & \text{otherwise} \end{cases}$$

Any characteristic vector belongs to the simplex of dim $n$:

$$\forall C \subset V \ x^C \in S_n = \{x \in \mathbb{R}^n \mid e^t x = 1 \text{ et } \forall i \in \{1, \ldots, n\} \ x_i \geq 0\}$$
Quadratic Function

- Let $A_G = (a_{i,j})$ the adjacency matrix of a graph $G$:
  \[
  a_{i,j} = \begin{cases} 
  1 & \text{if } (i,j) \in E, \\
  0 & \text{otherwise}. 
  \end{cases}
  \]

- and let the function:
  \[
  g(x) = x^t A_G x + \frac{1}{2} x^t x = x^t A x \\
  \text{with } \left\{ \begin{array}{l}
  A = A_G + \frac{1}{2} I \\
  x \in S_n
  \end{array} \right.
  \]

- $x^*$ is a strict maxima of $g$ iff:
  \[
  \exists \epsilon > 0 \mid \left\{ \begin{array}{l}
  |y - x^*| < \epsilon \\
  |y - x^*| < \epsilon \text{ and } f(y) = f(x^*) \Rightarrow y = x^* \\
  f(y) \leq f(x^*)
  \end{array} \right.
  \]
Motzkin-Straus (65)-Bomze (97) theorem

Theorem: Let $S \subset V$ and $x^S$ its characteristic vector, then:

1. $S$ is a maximum clique of $G$ iff $x^S$ is a global maximum of $g$ over $S_n$. We have then:

$$\omega(G) = \frac{1}{2(1 - g(x^S))}$$

2. $S$ is a maximal clique of $G$ iff $x^S$ is a local maximum of $g$ over $S_n$.

3. Any local maxima (and thus a global one) $x$ of $g$ over $S_n$ is strict and corresponds to a characteristic vector $x^S$ for some $S \subset V$. 

Resolution

- Computation of the maxima of:

\[ g(x) = x^t A x \text{ avec } A = A_G + \frac{1}{2} I \]

- Replication equations:

\[ x_i(t + 1) = x_i(t) \frac{(Ax(t))_i}{g(x)} \text{ note } \sum_{i=1}^{n} x_i(t) = \frac{g(x)}{g(x)} = 1 \]

- \( A \) being symmetric:
  - \( g(x(t)) \) is a strictly increasing function of \( t \),
  - The processus converge to a stationary point \( (x_i(t + 1) = x_i(t)) \) which corresponds to a local maximum of \( g \).
Let us consider $A\overline{G} = (\overline{a}_{ij})$ with $\overline{a}_{i,j} = 1$ if $(i, j) \not\in E$, 0 otherwise.

$$A\overline{G} = ee^t - AG - I$$

Problem transformation:

$$f(x) = x^t \overline{A}x = x^t \left( A\overline{G} + \frac{1}{2}I \right) x$$

$$= x^t (ee^t - AG - I + \frac{1}{2}I) x$$

$$= x^t [ee^t - (AG + \frac{1}{2}I)] x$$

$$= x^tee^tx - x^t (AG + \frac{1}{2}I) x$$

$$= 1 - g(x)$$
Theorem: Let $S \subset V$ and $x^S$ its characteristic , then:

1. $S$ is a maximum clique of $G$ iff $x^S$ is a global minimum of $f$ over $S_n$. We have then:

$$\omega(G) = \frac{1}{2f(x^*)}$$

2. $S$ is a maximal clique of $G$ iff $x^S$ is a local minimum of $f$ over $S_n$.

3. Any local minimum (and hence global minimum) $x$ of $g$ over $S_n$ is strict and corresponds to a characteristic vector $x^S$ for some $S \subset V$. 
Theorem intuition (1/2)

- Problem formulation

\[ f(x) = x^t A x = x^t (A_G^{-1} + \frac{1}{2} I) x \]
\[ = \sum_{i=1}^{n} \frac{1}{2} x_i^2 + \sum_{j \in V \setminus E} x_i x_j \]

- for any characteristic vector \( x^C \) of \( C \subset V \):

\[ f(x^C) = \frac{|C|}{2|C|^2} + \sum_{i=1}^{n} \sum_{j \in V \setminus E} x_i^C x_j^C \]
\[ = \frac{1}{2|C|} + \sum_{(i,j) \in E \setminus C^2} x_i^C x_j^C \]

If \( C \) is a clique:

\[ f(x) = \frac{1}{2|C|} \]

- The more \( C \) is «large» the more, \( f(x) \) is «small». 
Theorem intuition (2/2)

- Let us suppose that we add to $C$ a single vertex adjacent to all vertices of $C$ but one.
- Let $x^{C'}$ the resulting characteristic vector:

$$f(x^{C'}) = \frac{1}{2|C|} + \sum \sum_{(i,j) \in C''} |(i,j) \notin E| x^C_i x^{C'}_j$$

$$= \frac{1}{2(|C|+1)} + \frac{1}{(|C|+1)^2}$$

- $f(x^{C'})$ is larger or smaller than $f(x^C)$?

$$\frac{1}{2(|C|+1)} + \frac{1}{(|C|+1)^2} \quad \nless \quad \frac{1}{2|C|}$$

$$|C| + 1 + 2 \quad \nless \quad \frac{(|C|+1)^2}{|C|} = |C| + 2 + \frac{1}{|C|}$$

$$3 \quad \nless \quad 2 + \frac{1}{|C|}$$
Interest of the reformulation

- Application to valuated graphs
- maximum Clique:

\[ \omega(G) = \max \{|S| \text{ such that } S \text{ is a clique of } G \} \]

- maximum weighted clique
  - Let us consider a vector of weight: \( w \in \mathbb{R}^n \)

\[ \omega(G, w) = \max \{W(S) \text{ such that } S \text{ is a clique of } G\} \]

with

\[ W(S) = \sum_{i \in S} w_i \]
Wighted Cliques

Clique of maximal (resp. maximum) weights correspond to the local (resp. global) minimums of:

\[ f(x) = x^t C(w)x \]

with

\[ C(w)_{i,j} = \begin{cases} 
\frac{1}{2w_i} & \text{if } i = j \\
\frac{1}{2w_i} + \frac{1}{2w_j} & \text{if } i \neq j \text{ and } (i,j) \not\in E \\
0 & \text{otherwise}
\end{cases} \]

- Ability to include a priori informations!
  - distance between points,
  - regions similarities...
Resolution

- Problem: minimise over $S_n$

$$f(x) = x^t A x$$

- Formulation in terms of «Linear Complementarity Problem» (LCP)
  - Found $y, \bar{x}$ such that:

$$y = q_G + M_G x \geq 0, \bar{x} = [x, x_{n+1}, x_{n+2}], x^t y = 0$$

$$q_G = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ -1 \\ 1 \end{pmatrix} \quad \text{et} \quad M_G = \begin{pmatrix} A & -e & e \\ e^t & 0 & 0 \\ -e^t & 0 & 0 \end{pmatrix}$$
LCP: Iterative method based on the choice of a pivot element at each step

Locatelli & Pellilo proposed an heuristic method to choose these pivoting elements \(\rightarrow\) PBH algorithm.

The PBH algorithm if formally equivalent to the \(SM^1\) algorithm.
Soft Assign algorithm

- We have to determine a permutation matrix between 2 graphs to check the existence of an isomorphism or a sub graph isomorphisms.
- The Soft assign algorithm is one of the most famous algorithm to that aim.

- We consider two graphs $G$ and $g$ with valuated edges
  - $G_{a,b}$ weight of edge $(a, b)$ in $G$
  - $g_{i,j}$ weight of edge $(i, j)$ in $g$
- We consider:
  - A permutation matrix $M$ ($M_{a,i} = 1$) if $a$ is associated to $i$, 0 otherwise.
  - A similarity function between edges:
    
    $$C_{a,b,i,j} = \begin{cases} 
    0 & \text{Si } G_{a,b} \text{ or } g_{i,j} \text{ is null} \\
    c(G_{a,b}, g_{i,j}) & \text{otherwise}
    \end{cases}$$

    with (for example):

    $$c(G_{a,b}, g_{i,j}) = 1 - 3|G_{a,b} - g_{i,j}|$$
Soft assign problem

- We would like to minimize:

\[
E_{wg}(M) = -\frac{1}{2} \sum_{a=1}^{A} \sum_{i=1}^{I} \sum_{b=1}^{A} \sum_{j=1}^{I} M_{a,i} M_{b,j} C_{a,i,b,j}
\]

with \( A \) (resp. \( I \)) nb vertices in \( G \) (resp. \( g \)).

- i.e. match a maximum of similar edges
SoftAssign : Illustration

- If $G_{a,b}, g_{i,j} \in \{1, NULL\}$, $C_{a,b,i,j} \in \{0, 1\}$ and

$$E_{wg}(M) = -\frac{1}{2} \sum_{a=1}^{A} \sum_{i=1}^{I} \sum_{b=1}^{A} \sum_{j=1}^{I} M_{a,i} M_{b,j} G_{a,b} g_{i,j}$$

- What we are searching for:
SoftAssign: simple problem

- Let us consider the small following problem: Given \( \{X_1, \ldots, X_n\} \) determine \( \{m_1, \ldots, m_n\} \) such that:
  - \( m_i = 1 \) if \( X_i = \max_j X_j \),
  - 0 otherwise.

- Equivalent to maximize:

\[
\sum_{i=1}^{n} m_i X_i \text{ with } \sum_{i=1}^{n} m_i = 1, m_i \in \{0, 1\}
\]

- For any \( \beta > 0 \) let us consider:

\[
m_j(\beta) = \frac{e^{\beta X_j}}{\sum_{i=1}^{n} e^{\beta X_i}}
\]

- We have:

\[
\lim_{\beta \to +\infty} m_j(\beta) = \begin{cases} 
1 & \text{if } X_j = \max X_i \\
0 & \text{otherwise}
\end{cases}
\]
function SOFTASSIGN1(\{X_1, \ldots, X_n\})
\[
\beta \leftarrow \beta_0 \\
\text{while } \beta < \beta_f \text{ do} \\
\quad m_i \leftarrow e^{\beta X_i} \\
\quad m_i \leftarrow \frac{m_i}{\sum_{i=1}^{n} m_i} \\
\quad \text{Do other parts of the algorithm} \\
\quad \text{increase } \beta \\
\text{end while} \\
\text{return } \{m_1, \ldots, m_n\} \\
\text{end function}
\]

- We determine the max softly (hence the name soft assign)
Let now consider a permutation matrix $M$, between two graphs $G$ and $g$ and one variable $X_{a,i}$.

Maximize according to $M$:

$$E_{ass}(M) = \sum_{a=1}^{A} \sum_{i=1}^{I} M_{a,i} X_{a,i}$$

We have no more a single constraint ($\sum_{i=1}^{n} m_i = 1$) but two:

- $\forall a \in \{1, \ldots, A\} \sum_{i=1}^{I} M_{a,i} = 1$
- $\forall i \in \{1, \ldots, I\} \sum_{a=1}^{A} M_{a,i} = 1$
SoftAssign : Algorithm 2

- We normalize iteratively according to lines and columns \( \approx \) we apply algorithm 1 one lines and then on columns

procedure SOFTASSIGN2\((X_{ai})\)

Output : \(M\)

\[
\beta \leftarrow \beta_0
\]

while \(\beta < \beta_f\) do

\[
M_{a,i} \leftarrow e^{\beta X_{ai}}
\]

repeat

\[
M_{a,i} \leftarrow \frac{M_{a,i}}{\sum_{j=1}^{I} M_{a,j}}
\]

\[
M_{a,i} \leftarrow \frac{M_{a,i}}{\sum_{x=1}^{A} M_{x,i}}
\]

until \(M\) converge

Do the remaining part of the algorithm

Increment \(\beta\)

end while

end procedure

- \(M\) converges toward a permutation matrix
SoftAssign: Towards the (1/2)

- Problem: The matching is not a research of a max

\[
\arg\max A \sum_{a=1}^{A} \sum_{i=1}^{I} M_{a,i} X_{a,i} \neq \arg\min -\frac{1}{2} A \sum_{a=1}^{A} \sum_{i=1}^{I} \sum_{b=1}^{A} \sum_{j=1}^{I} M_{a,i} M_{b,j} C_{a,i,b,j}
\]

- Let us consider \( E_{wg}(M) = -\frac{1}{2} \sum_{a=1}^{A} \sum_{i=1}^{I} \sum_{b=1}^{A} \sum_{j=1}^{I} M_{a,i} M_{b,j} C_{a,i,b,j} \) as a function of \( AI \) variables.

- and apply Taylor to order 1:

\[
E_{wg}(M) \approx E_{wg}(M^0) + \sum_{a=1}^{A} \sum_{i=1}^{I} \frac{\partial E_{wg}(M)}{\partial M_{a,i}}(M)|_{M=M^0}(M_{a,i} - M_{a,i}^0)
\]
SoftAssign: Toward the solution

- We have thus:

\[
E_{wg}(M) = -\frac{1}{2} \sum_{a=1}^{A} \sum_{i=1}^{I} \sum_{b=1}^{A} \sum_{j=1}^{I} M_{a,i} M_{b,j} C_{a,i,b,j} \\
\approx E_{wg}(M^0) + \sum_{a=1}^{A} \sum_{i=1}^{I} \sum_{j=1}^{I} \frac{\partial E_{wg}(M^0)}{\partial M_{a,i}}(M^0)(M_{a,i} - M_{a,i}^0)
\]

with:

\[
\frac{\partial E_{wg}(M^0)}{\partial M_{a,i}}(M^0) = - \sum_{b=1}^{A} \sum_{j=1}^{I} M_{b,j}^0 C_{a,i,b,j}
\]
Softassign: toward the solution

- Let's define $Q_{a,i} = -\frac{\partial E_{wg}(M^0)}{\partial M_{a,i}}$
- We have:

$$E_{wg}(M) \approx E_{wg}(M^0) - \sum_{a=1}^{A} \sum_{i=1}^{I} Q_{ai} (M_{a,i} - M_{a,i}^0)$$

$$\approx Cte - \sum_{a=1}^{A} \sum_{i=1}^{I} Q_{a,i} M_{a,i}$$

- Minimizing $E_{wg}(M)$ becomes equivalent to maximize:

$$\sum_{a=1}^{A} \sum_{i=1}^{I} Q_{a,i} M_{a,i}$$

A problem of computation of a max!
Soft Assign: The method

1. Take an initial guess $M$

2. Perform a Taylor decomposition of $E_{wg}(M)$

3. Perform a softassign corresponding to the computation of the max of:

$$\sum_{a=1}^{A} \sum_{i=1}^{I} Q_{a,i} M_{a,i}$$

4. Take the resulting $M$ as result and loop by incrementing $\beta$

- Remark: We add lines and columns to $M$ in order to transform inequalities $\sum_{a=1}^{A} M_{a,i} \leq 1$ and $\sum_{i=1}^{I} M_{a,i} \leq 1$ into equalities $\rightarrow$ matrix $\tilde{M}$.
- allows to encode non matched vertices.
SoftAssign: The algorithm

procedure \textsc{SoftAssign}(G, g, \beta_f, \beta_0)

Output: \( \beta, M \)

\[
\beta \leftarrow \beta_0 \\
\tilde{M}_{a,i} \leftarrow 1 + \epsilon
\]

while \( \beta < \beta_f \) do

repeat

\[
Q_{a,i} \leftarrow -\frac{\partial E_{wg}(M)}{\partial M_{a,i}}
\]

\[
M_{a,i}^0 \leftarrow e^{\beta Q_{a,i}}
\]

repeat

\[
\tilde{M}_{a,i}^1 \leftarrow \sum_{i=1}^{I+1} \tilde{M}_{a,i}^0
\]

\[
\tilde{M}_{a,i}^0 \leftarrow \sum_{a=1}^{A+1} \tilde{M}_{a,i}^1
\]

until \( \tilde{M} \) converge or nb iter > \( I_1 \)

until \( M \) converge or nb iter > \( I_0 \)

\[
\beta \leftarrow \beta_r \beta
\]

end while

threshold \( M_{a,i} \)

end procedure
Extension to vertex’s attributes

- Add a function $C_{a,i}^{(1)}$ encoding distances between vertices

$$E_{arg}(M) = -\frac{1}{2} \sum_{a=1}^{A} \sum_{i=1}^{I} \sum_{b=1}^{A} \sum_{j=1}^{I} M_{a,i} M_{b,j} C_{a,i,b,j}^{(2)} + \alpha \sum_{a=1}^{A} \sum_{i=1}^{I} M_{a,i} C_{a,i}^{(1)}$$

- is equivalent to add $\alpha C_{a,i}^{(1)}$ to $Q_{a,i}$ in previous algorithm.
Graph Edit distance

- Exactly the same formulation than String edit distance.
- \( d(G_1, G_2) \) is the minimal cost of a set of operations transforming \( G_1 \) into \( G_2 \) using vertex/edge insertion/deletion/substitutions.
- Solved with \( A^* \) algorithms but with an exponential complexity in the number of nodes of both graphs.
- Efficient heuristic (and thus sub optimal) algorithms exists based on the bipartite assignment algorithm.
**Definition (Edit path)**

Given two graphs $G_1$ and $G_2$ an **edit path** between $G_1$ and $G_2$ is a sequence of node or edge removal, insertion or substitution which transforms $G_1$ into $G_2$.

A substitution is denoted $u \rightarrow v$, an insertion $\epsilon \rightarrow v$ and a removal $u \rightarrow \epsilon$.

Alternative edit operations such as merge/split have been also proposed[Ambrumen et al., 2003].
All paths go to Roma... However we are usually only interested by the shortest one.

Let $c(.)$ denote the cost of any elementary operation. The cost of an edit path is defined as the sum of the costs of its elementary operations.

- All cost are positive: $c() \geq 0$,
- A node or edge substitution which does not modify a label has a 0 cost: $c(l \rightarrow l) = 0$.

If all costs are equal to 1, the cost of this edit path is equal to 5.
Definition (Graph edit distance)

The graph edit distance between $G_1$ and $G_2$ is defined as the cost of the less costly path within $\Gamma(G_1, G_2)$. Where $\Gamma(G_1, G_2)$ denotes the set of edit paths between $G_1$ and $G_2$.

$$d(G_1, G_2) = \min_{\gamma \in \Gamma(G_1, G_2)} \sum_{e \in \gamma} c(e)$$
Main concepts

Tree search algorithms explore the space $\Gamma(G_1, G_2)$ with some heuristics to avoid to visit unfruitful states. More precisely let us consider a partial edit path $p$ between $G_1$ and $G_2$. Let:

- $g(p)$ the cost of the partial edit path.
- $h(p)$ a lower bound of the cost of the remaining part of the path required to reach $G_2$.

$$\forall \gamma \in \Gamma(G_1, G_2), \gamma = p.q, \quad g(p) + h(p) \leq d_{\gamma}(G_1, G_2)$$

where $d_{\gamma}(G_1, G_2)$ denotes the cost of the edit path $\gamma$. 

Main concepts

Let $UB$ denote the best approximation of the GED found so far. If $g(p) + h(p) > UB$ we have:

$$\forall \gamma \in \Gamma(G_1, G_2), \gamma = p.q, d_\gamma(G_1, G_2) \geq g(p) + h(p) > UB$$

In other terms, all the sons of $p$ will provide a greater approximation of the GED and correspond thus to unfruitful nodes.
Choosing a good lower bound

Let us suppose that \( n_1 \) and \( n_2 \) vertices of respectively \( V_1 \) and \( V_2 \) remain to be assigned. Different choices are possible for function \( h \)[Abu-Aisheh, 2016]:

Null function: \( h(p) = 0 \),

Bipartite Function: \( h(p) = d_{lb}(G_1, G_2) \) (see below). Closer bound but requires a \( \mathcal{O}(\max\{n_1, n_2\}^3) \) algorithm.

Hausdorff distance \( h(p) \) is set to the Hausdroff distance between the sets of \( n_1 \) and \( n_2 \) vertices (including their incident edges). Requires \( \mathcal{O}((n_1 + n_2)^2) \) computation steps.
A* algorithm

1: **Input:** Two graphs $G_1$ and $G_2$ with $V_1 = \{u_1, \ldots, u_n\}$ and $V_2 = \{v_1, \ldots, v_m\}$
2: **Output:** A minimum edit path between $G_1$ and $G_2$
3: OPEN=$\{u_1 \rightarrow \epsilon\} \cup \bigcup_{w \in V_2} \{u_1 \rightarrow w\}$
4: repeat
5: \[ p = \text{arg min}_{q \in \text{OPEN}} \{g(q) + h(q)\} \]
6: if $p$ is a complete edit path then
7: \[ \text{return } p \]
8: end if
9: Complete $p$ by operations on $u_{k+1}$ or on remaining vertices of $V_2$.
10: Add completed paths to OPEN
11: until end of times
A* algorithm: discussion

- 😊 If algorithm $A^*$ terminates it always returns the optimal value of the GED.
- 😞 The set OPEN may be as large as the number of edit paths between $G_1$ and $G_2$.
- 🙁 The algorithm do not return any result before it finds the optimal solution.
Depth first search algorithm

1: **Input:** Two graphs $G_1$ and $G_2$ with $V_1 = \{u_1, \ldots, u_n\}$ and $V_2 = \{v_1, \ldots, v_m\}$
2: **Output:** $\gamma_{UB}$ and $UB$ a minimum edit path and its associated cost
3:
4: $(\gamma_{UB}, UBCOST) = GoodHeuristic(G_1, G_2)$
5: initialize OPEN
6: while OPEN ≠ ∅ do
7: $p = OPEN.popFirst()$
8: if $p$ is a leaf (i.e. if all vertices of $V_1$ are mapped) then
9: complete $p$ by inserting pending vertices of $V_2$
10: update $(\gamma_{UB}, UBCOST)$ if required
11: else
12: Stack into OPEN all sons $q$ of $p$ such that $g(q) + h(q) < UBCOST$.
13: end if
14: end while
The number of pending edit paths in OPEN is bounded by $|V_1|.|V_2|$, the initialization by an heuristic allows to discard many branches, this algorithm quickly find a first edit path. It may be tuned into [Abu-Aisheh, 2016]:

- An any time algorithm,
- A Parallel or distributed algorithm.

The computation of the optimal value of the Graph Edit distance may anyway require long processing times.
Independent Edit paths

An element $\gamma \in \Gamma(G_1, G_2)$ is potentially infinite by just doing and undoing a given operation (e.g. insert and then delete a node). All cost being positive such an edit path can not correspond to a minimum:

**Definition (Independent Edit path)**

An independent edit path between two labeled graphs $G_1$ and $G_2$ is an edit path such that:

1. No node nor edge is both substituted and removed,
2. No node nor edge is simultaneously substituted and inserted,
3. Any inserted element is never removed,
4. Any node or edge is substituted at most once,
Proposition

The elementary operations of an independent edit path between two graphs $G_1$ and $G_2$ may be ordered into a sequence of removals, followed by a sequence of substitutions and terminated by a sequence of insertions.

Note that $\hat{G}_1 \cong_s \hat{G}_2$
Decomposition of an Edit Path

May be reordered into:

\[
\begin{align*}
G_1 & \rightarrow \hat{G}_1 \rightarrow S \rightarrow \hat{G}_2 \rightarrow I \\
& \rightarrow R
\end{align*}
\]
GED Formulation based on $\hat{G}_1$

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. We have:

$$d(G_1, G_2) = \sum_{v \in V_1 \setminus \hat{V}_1} c_{vd}(v) + \sum_{e \in E_1 \setminus \hat{E}_1} c_{ed}(e) + \sum_{v \in \hat{V}_1} c_{vs}(v) + \sum_{e \in \hat{E}_1} c_{es}(e)$$

$$+ \sum_{v \in V_2 \setminus \hat{V}_2} c_{vi}(v) + \sum_{e \in E_2 \setminus \hat{E}_2} c_{ei}(e)$$

If all costs are constants we have:

$$d(G_1, G_2) = (|V_1| - |\hat{V}_1|)c_{vd} + (|E_1| - |\hat{E}_1|)c_{ed} + V_f c_{vs} + E_f c_{es}$$

$$+ (|V_2| - |\hat{V}_2|)c_{vi} + (|E_2| - |\hat{E}_2|)c_{ei}$$

where $V_f$ (resp. $E_f$) denotes the number of vertices (resp. edges) substituted with a non zero cost and $c_{vd}$,
GED Formulation based on $\hat{G}_1$

By grouping constant terms minimize:

$$d(G_1, G_2) = (|V_1| - |\hat{V}_1|)c_{vd} + (|E_1| - |\hat{E}_1|)c_{ed} + V_f c_{vs} + E_f c_{es}$$

$$+ (|V_2| - |\hat{V}_2|)c_{vi} + (|E_2| - |\hat{E}_2|)c_{ei}$$

is equivalent to maximize:

$$M(P) \overset{not.}{=} |\hat{V}_1|(c_{vd} + c_{vi}) + |\hat{E}_1|(c_{ed} + c_{ei}) - V_f c_{vs} - E_f c_{es}$$

We should thus maximize $\hat{G}_1$ while minimizing $V_f$ and $E_f$.

If $c(u \rightarrow v) \geq c(u \rightarrow \epsilon) + c(\epsilon \rightarrow v)$, $M(P)$ is maximum for $V_f = E_f = \emptyset$ and $\hat{G}_1$ is a maximum common sub graph of $G_1$ and $G_2$ [Bunke, 1997, Bunke and Kandel, 2000].
Further restriction

Definition (Restricted edit path)

A restricted edit path is an independent edit path in which an edge cannot be removed and then inserted.

Proposition

If $G_1$ and $G_2$ are simple graphs, there is a one-to-one mapping between the set of restricted edit paths between $G_1$ and $G_2$ and the set of injective functions from a subset of $V_1$ to $V_2$. We denote by $\varphi_0$, the special function from the empty set onto the empty set.
**ε-assignments**

- Let $v_1$ and $V_2$ be two sets, with $n = |v_1|$ and $m = |V_2|$.
- Consider $V_1^\epsilon = V_1 \cup \{\epsilon\}$ and $V_2^\epsilon = V_2 \cup \{\epsilon\}$.

**Definition**

An ε-assignment from $V_1$ to $V_2$ is a mapping $\varphi : V_1^\epsilon \rightarrow \mathcal{P}(V_2^\epsilon)$, satisfying the following constraints:

\[
\begin{align*}
\forall i \in V_1 & \quad |\varphi(i)| = 1 \\
\forall j \in V_2 & \quad |\varphi^{-1}(j)| = 1 \\
\epsilon & \in \varphi(\epsilon)
\end{align*}
\]

An ε assignment encodes thus:

1. **Substitutions**: $\varphi(i) = j$ with $(i, j) \in V_1 \times V_2$.
2. **Removals**: $\varphi(i) = \epsilon$ with $i \in V_1$.
3. **Insertions**: $j \in \varphi^{-1}(\epsilon)$ with $j \in V_2$. 
From assignments to matrices

- An $\epsilon$-assignment can be encoded in matrix form:
\[
X = (x_{i,k})_{(i,k) \in \{1,\ldots,n+1\} \times \{1,\ldots,m+1\}} \text{ with }
\]
\[
\forall (i,k) \in \{1,\ldots,n+1\} \times \{1,\ldots,m+1\} \quad x_{i,k} = \begin{cases} 
1 & \text{if } k \in \varphi(i) \\
0 & \text{else}
\end{cases}
\]

- We have:
\[
\begin{cases}
\forall i = 1,\ldots,n, \quad \sum_{k=1}^{m+1} x_{i,k} = 1 & (|\varphi(i)| = 1) \\
\forall k = 1,\ldots,m, \quad \sum_{j=1}^{n+1} x_{j,k} = 1 & (|\varphi^{-1}(k)| = 1) \\
\forall (i,j) \quad x_{n+1,m+1} = 1 & (\epsilon \in \varphi(\epsilon)) \\
x_{i,j} \in \{0,1\}
\end{cases}
\]
From functions to matrices

The set of permutation matrices encoding $\epsilon$-assignments is called the set of $\epsilon$-assignment matrices denoted by $A_{n,m}$.

Usual assignments are encoded by larger $(n + m) \times (n + m)$ matrices [Riesen, 2015].
Let us consider two simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with $|V_1| = n$ and $|V_2| = m$.

**Proposition**

There is a one-to-one relation between the set of restricted edit paths from $G_1$ to $G_2$ and $A_{n,m}$.

**Theorem**

Any non-infinite value of $\frac{1}{2}x^T \Delta x + c^T x$ corresponds to the cost of a restricted edit path. Conversely the cost of any restricted edit path may be written as $\frac{1}{2}x^T \Delta x + c^T x$ with the appropriate $x$. 
Costs of Node assignments

\[
C = \begin{pmatrix}
  c(u_1 \rightarrow v_1) & \cdots & c(u_1 \rightarrow v_m) & c(u_1 \rightarrow \varepsilon) \\
  \vdots & & \ddots & \vdots \\
  c(u_i \rightarrow v_j) & \cdots & c(u_i \rightarrow \varepsilon) & \vdots \\
  c(u_n \rightarrow v_1) & \cdots & c(u_n \rightarrow v_m) & c(u_n \rightarrow \varepsilon) \\
  c(\varepsilon \rightarrow v_1) & c(\varepsilon \rightarrow v_i) & c(\varepsilon \rightarrow v_m) & 0
\end{pmatrix}
\]

- \( c = \text{vect}(C) \)
Cost of edges assignments

- Let us consider a \((n + 1)(m + 1) \times (n + 1)(m + 1)\) matrix \(D\) such that:

\[
d_{ik,jl} = c_e(i \to k, j \to l)
\]

with:

<table>
<thead>
<tr>
<th>((i, j))</th>
<th>((k, l))</th>
<th>edit operation</th>
<th>cost (c_e(i \to k, j \to l))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\in E_1)</td>
<td>(\in E_2)</td>
<td>substitution of ((i, j)) by ((k, l))</td>
<td>(c((i, j) \to (k, l)))</td>
</tr>
<tr>
<td>(\in E_1)</td>
<td>(\not\in E_2)</td>
<td>removal of ((i, j))</td>
<td>(c((i, j) \to \epsilon))</td>
</tr>
<tr>
<td>(\not\in E_1)</td>
<td>(\in E_2)</td>
<td>insertion of ((k, l)) into (E_1)</td>
<td>(c(\epsilon \to (k, l)))</td>
</tr>
<tr>
<td>(\not\in E_1)</td>
<td>(\not\in E_2)</td>
<td>do nothing</td>
<td>0</td>
</tr>
</tbody>
</table>

- \(\Delta = D\) if both \(G_1\) and \(G_2\) are undirected and
- \(\Delta = D + D^T\) else.

- Matrix \(\Delta\) is symmetric in both cases.
Graph edit distance and assignments

- We have thus:

\[
\text{GED}(G_1, G_2) = \min \left\{ \frac{1}{2} x^T \Delta x + c^T x \mid x \in \text{vec}[A_{\sim m, n}] \right\}
\]

- Matrix $\Delta$ is non convex with several local minima. The problem is thus $\mathcal{NP}$-complete.
Let us approximate

- One solution to solve this quadratic problem consists in dropping the quadratic term. We hence get:

\[ d(G_1, G_2) \approx \min \{ c^T x \mid x \in \text{vec}[A_{n,m}] \} = \min \sum_{i=1}^{n+m} \sum_{j=1}^{n+m} c_{i,j} x_{i,j} \]

- This problem is an instance of a bipartite graph matching problem also called linear sum assignment problem.
Bipartite Graph matching

- Approximations of the Graph edit distance defined using bipartite Graph matching methods are called bipartite Graph edit distances (BP-GED).
- Different methods, such as Munkres or Jonker-Volgerand [Burkard et al., 2012] allow to solve this problem in cubic time complexity.
- The general procedure is as follows:
  1. compute:
     \[ x = \arg \min c^T x \]
  2. Deduce the edge operations from the node operations encoded by \( x \).
  3. Return the cost of the edit path encoded by \( x \).
The matrix $C$ defined previously only encodes node information.

Idea: Inject edge information into matrix $C$. Let

$$d_{i,j} = \min_x \sum_{k=1}^{n+m} c(i_k \rightarrow j_l)x_{k,l}$$

the cost of the optimal mapping of the edges incident to $i$ onto the edge incident to $j$.

Let:

$$c^*_{i,j} = c(u_i \rightarrow v_j) + d_{i,j} \text{ and } x = \arg\min(c^*)^T x$$

The edit path deduced from $x$ defines an edit cost $d_{ub}(G_1, G_2)$ between $G_1$ and $G_2$. 
We may alternatively consider:

\[
d_{lb}(G_1, G_2) = \sum_{i=1}^{n+m} \sum_{j=1}^{n+m} \left( c(u_i \rightarrow v_j) + \frac{1}{2} d_{i,j} \right) x_{i,j}
\]

The resulting distances satisfy [Riesen, 2015]:

\[
d_{lb}(G_1, G_2) \leq d(G_1, G_2) \leq d_{ub}(G_1, G_2)
\]
Matching larger structure

- The upper bound provided by $d_{ub}(G_1, G_2)$ may be large, especially for large graphs. So a basic idea consists in enlarging the considered substructures:
  1. Consider both the incident edges and the adjacent nodes.
  2. Associate a bag of walks to each vertex and define $C$ as the cost of matching these bags [Gaüzère et al., 2014].
  3. Associate to each vertex a subgraph centered around it and define $C$ as the optimal edit distance between these subgraphs [Carletti et al., 2015].
  4. ...

- All these heuristics provide an upper bound for the Graph edit distance.
- But: up to a certain point the linear approximation of a quadratic problem reach its limits.
One idea to improve the results of the linear approximation of the GED consists in applying a post processing stage.

Two ideas have been proposed:

1. By modifying the initial cost matrix and recomputing a new assignment.
2. By swapping elements in the assignment returned by the linear algorithm.
• $q$ consecutive trials to improve the initial guess provided by a bipartite algorithm.

1: Build a Cost matrix $C^*$
2: Compute $x = \arg\min (c^*)^T x$
3: Compute $d_{best} = d_x(G_1, G_2)$
4: for $i = 1$ to $q$ do
5: determine node operation $u_i \to v_j$ with highest cost
6: set $c^*_{i,j} = +\infty$ \hspace{1cm} \triangleright$prevent assignment $u_i \to v_j$
7: compute a new edit path $\gamma$ on modified matrix $C^*$.
8: if $d_\gamma(G_1, G_2) < d_{best}$ then
9: $d_{best} = d_\gamma(G_1, G_2)$
10: end if
11: end for
Alternative strategies

- BP-Iterative never reconsiders an assignment.
- Riesen [Riesen, 2015] proposed an alternative procedure called $BP - \text{Float}$ which after each new forbidden assignment:
  1. reconsider the entries previously set to $+\infty$,  
  2. Restore them if the associated edit cost decreases.

- An alternative search procedure is based on Genetic algorithms [Riesen, 2015]:
  1. Build a population by randomly forbidding some entries of $C^*$,  
  2. Select a given percentage of the population whose cost matrix is associated to the lowest edit distance,  
  3. Cross the population by forbidding an entry if this entry is forbidden by one of the two parent.  
  4. Go back to step 2.
Swapping assignments

- Forbidding entries of matrix $C^*$ requires to recompute a new assignment from scratch.

- The basic idea of the swapping strategy consists to replace assignments:

\[
\begin{align*}
&u_i \rightarrow v_k \\
u_j \rightarrow v_l
\end{align*}
\]

by

\[
\begin{align*}
&u_i \rightarrow v_l \\
u_j \rightarrow v_k
\end{align*}
\]

- Or in matrix terms, replacing

\[
\begin{align*}
x_{i,k} &= x_{j,l} = 1; \\
x_{i,l} &= x_{j,k} = 0
\end{align*}
\]

by

\[
\begin{align*}
x_{i,l} &= x_{j,k} = 1; \\
x_{i,k} &= x_{j,l} = 0.
\end{align*}
\]
BP-Greedy-Swap

1: swapped=true
2: while swapped do
3: Search for indices \((i, j, k, l)\) such that \(x_{i,k} = x_{j,l} = 1\)
4: \(cost_{orig} = c_{i,k} + c_{j,l}\)
5: \(cost_{swap} = c_{i,l} + c_{j,k}\)
6: if \(|cost_{orig} - cost_{swap}| < \theta cost_{orig}\) then
7: \(x' = swap(x)\)
8: if \(d_{x'}(G_1, G_2) < d_{best}\) then
9: \(d_{best} = d_{x'}(G_1, G_2)\)
10: best swap =\(x'\); swapped=true
11: end if
12: end if
13: if we swapped for at least one \((i, j) \in V_1^2\) then update \(x\) according to best swap.
14: end while
Discussion

- **BP – Greedy – Swap** as **BP – Iterative** never reconsiders a swap,
- An alternative exploration of the space of solutions may be performed using Genetic algorithms[Riesen, 2015].
- Or by a tree search:
  - This procedure is called **Beam search**
  - Each node of the tree is defined by \((x, q, d_x(G_1, G_2))\) where \(x\) is an assignment and \(q\) the depth of the node.
  - Nodes are sorted according to:
    1. their depth,
    2. their cost.
BP-Beam

1: Build a Cost matrix $C^*$
2: Compute $x = \arg \min (c^*)^T x$ and $d_x(G_1, G_2)$
3: $OPEN = \{ (x, 0, d_x(G_1, G_2)) \}$
4: while $OPEN \neq \emptyset$ do
5: remove first tree node in OPEN: $(x, q, d_x(G_1, G_2))$
6: for $j=q+1$ to $n+m$ do
7: $x' = \text{swap}(x, q+1, j)$
8: $OPEN = OPEN \cup \{ (x', q+1, d_{x'}(G_1, G_2)) \}$
9: if $d_{x'}(G_1, G_2) < d_{best}$ then
10: $d_{best} = d_{x'}(G_1, G_2)$
11: end if
12: end for
13: while size of OPEN $> b$ do
14: Remove tree nodes with highest value $d_x$ from OPEN
15: end while
16: end while

Examples
Strings
Bipartitite Assignement
Graph Terminology
Graph Matching
Graph Edit distance
From linear to quadratic optimization

- Improvements of Bipartite matching explore the space of edit paths around an initial solution.
- In order to go beyond these results, this search must be guided by considering the real quadratic problem.

\[ d(G_1, G_2) = \frac{1}{2} x^T \Delta x + c^T x \]
This work [Justice and Hero, 2006]:

- Proposes a quadratic formulation of the Graph Edit distance,
- Designed for Graphs with unlabeled edges by augmenting graphs with $\epsilon$ nodes,
- Solved by relaxing the integer constraint $x_{i,j} \in \{0, 1\}$ to $x_{i,j} \in [0, 1]$ and then solving it using interior point method.
- Propose a linear approximation corresponding to bipartite matching.
In this work [Neuhaus and Bunke, 2007]

- Also a quadratic formulation but with node operations restricted to substitution with the induced operations on edges.
- Works well mainly for graphs having close sizes with a cost of substitution lower than a cost of removal plus a cost of insertion.
From quadratic to linear problems

- Let us consider a quadratic problem:

\[ x^T Q x = \sum_{i=1}^{n} \sum_{j=1}^{n} q_{i,j} x_i x_j \]

- Let us introduce \( y_{n\times i+j} = x_i x_j \) we get:

\[ x^T Q x = \sum_{k=1}^{n^2} q_k y_k \]

with an appropriate renumbering of \( Q \)'s elements. Hence a Linear Sum Assignment Problem with additional constraints.

- Note that the Hungarian algorithm can not be applied due to additional constraints. We come back to tree based algorithms.

- This approach has been applied to the computation of the exact Graph Edit Distance by [Lerouge et al., 2016]
Franck Wolfe

- Start with a good guess $x_0$:
  \[
  x = x_0
  \]
  
  while a fixed point is not reached do
  
  \[
  b^* = \arg \min_{b} \{x^T \Delta + c^T \}b, b \in \{0, 1\}^{(n+m)^2}
  \]
  
  \[
  t^* = \text{line search between } x \text{ and } b^*
  \]
  
  \[
  x = x + t^*(b^* - x)
  \]
  
  end while

- $b^*$ minimizes a sum of:

  \[
  (x^T \Delta + c^T)_{i,j} = c_{i,j} + \sum_{k,l} d_{i,j,k,l} x_{k,l}
  \]

  which may be understood as the cost of mapping $i$ to $j$ given the previous assignment $x$.

- The distance decreases at each iteration. Moreover,
- at each iteration we come back to an integer solution,
Graduated NonConvexity and Concativity Procedure (GNCCP)

- Consider [Liu and Qiao, 2014]:

\[ S_\eta(x) = (1 - |\zeta|)S(x) + \zeta x^T x \text{ with } S(x) = \frac{1}{2} x^T \Delta x + c^T x \]

where \( \zeta \in [-1, 1] \).

\[
\begin{cases}
\zeta = 1 : \text{Convex objective function} \\
\zeta = -1 : \text{Concave objective function}
\end{cases}
\]

- The algorithm tracks the optimal solution from a convex to a concave relaxation of the problem.
From $\zeta = 1$ to $\zeta = 0$

$\zeta = 1$

$\zeta = 0$
\( \zeta = 1, d = 0.1, x = 0 \)

while \((\zeta > -1)\) and \((x \notin A_{n,m})\) do

\[ Q = \frac{1}{2} (1 - |\zeta|) \Delta + \zeta I \]

\[ L = (1 - |\zeta|) c \]

\[ x = FW(x, Q, L) \]

\[ \zeta = \zeta - d \]

end while
## Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of graphs</th>
<th>Avg Size</th>
<th>Avg Degree</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alkane</td>
<td>150</td>
<td>8.9</td>
<td>1.8</td>
<td>acyclic, unlabeled</td>
</tr>
<tr>
<td>Acyclic</td>
<td>183</td>
<td>8.2</td>
<td>1.8</td>
<td>acyclic</td>
</tr>
<tr>
<td>MAO</td>
<td>68</td>
<td>18.4</td>
<td>2.1</td>
<td></td>
</tr>
<tr>
<td>PAH</td>
<td>94</td>
<td>20.7</td>
<td>2.4</td>
<td>unlabeled, cycles</td>
</tr>
<tr>
<td>MUTAG</td>
<td>$8 \times 10$</td>
<td>40</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
### Experiments

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Alkane</th>
<th></th>
<th></th>
<th>Acyclic</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$d$</td>
<td>$e$</td>
<td>$t$</td>
<td>$d$</td>
<td>$e$</td>
<td>$t$</td>
</tr>
<tr>
<td>$A^*$</td>
<td>15</td>
<td>1.29</td>
<td>17</td>
<td>17</td>
<td>6.02</td>
<td></td>
</tr>
<tr>
<td>[Riesen and Bunke, 2009]</td>
<td>35</td>
<td>18</td>
<td>$\approx 10^{-3}$</td>
<td>35</td>
<td>18</td>
<td>$\approx 10^{-3}$</td>
</tr>
<tr>
<td>[Gaüzère et al., 2014]</td>
<td>33</td>
<td>18</td>
<td>$\approx 10^{-3}$</td>
<td>31</td>
<td>14</td>
<td>$\approx 10^{-2}$</td>
</tr>
<tr>
<td>[Carletti et al., 2015]</td>
<td>26</td>
<td>11</td>
<td>2.27</td>
<td>28</td>
<td>9</td>
<td>0.73</td>
</tr>
<tr>
<td>$\text{IPFP}_{\text{Random init}}$</td>
<td>22.6</td>
<td>7.1</td>
<td>0.007</td>
<td>23.4</td>
<td>6.1</td>
<td>0.006</td>
</tr>
<tr>
<td>$\text{IPFP}_{\text{Init [Riesen and Bunke, 2009]}}$</td>
<td>22.4</td>
<td>7.0</td>
<td>0.007</td>
<td>22.6</td>
<td>5.3</td>
<td>0.006</td>
</tr>
<tr>
<td>$\text{IPFP}_{\text{Init [Gaüzère et al., 2014]}}$</td>
<td>20.5</td>
<td>5</td>
<td>0.006</td>
<td>20.7</td>
<td>3.4</td>
<td>0.005</td>
</tr>
<tr>
<td>[Neuhaus and Bunke, 2007]</td>
<td>20.5</td>
<td>4.9</td>
<td>0.38</td>
<td>25.7</td>
<td>7.6</td>
<td>0.42</td>
</tr>
<tr>
<td>GNCCP</td>
<td>16.7</td>
<td>1.2</td>
<td>0.46</td>
<td>18.8</td>
<td>1.5</td>
<td>0.33</td>
</tr>
</tbody>
</table>
## Experiments

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MAO</th>
<th>PAH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$d$</td>
<td>$t$</td>
</tr>
<tr>
<td>[Riesen and Bunke, 2009]</td>
<td>105</td>
<td>$\approx 10^{-3}$</td>
</tr>
<tr>
<td>[Gaüzère et al., 2014]</td>
<td>49</td>
<td>$\approx 10^{-2}$</td>
</tr>
<tr>
<td>[Carletti et al., 2015]</td>
<td>44</td>
<td>6.16</td>
</tr>
<tr>
<td>IPFP Random init</td>
<td>65.2</td>
<td>0.031</td>
</tr>
<tr>
<td>IPFP Init [Riesen and Bunke, 2009]</td>
<td>59</td>
<td>0.031</td>
</tr>
<tr>
<td>IPFP Init [Gaüzère et al., 2014]</td>
<td>33.6</td>
<td>0.016</td>
</tr>
<tr>
<td>[Neuhaus and Bunke, 2007]</td>
<td>59.1</td>
<td>7</td>
</tr>
<tr>
<td>GNCCP</td>
<td>40.3</td>
<td>4.2</td>
</tr>
</tbody>
</table>
Two characteristics of a Graph edit distance algorithm:

- Mean Deviation:

\[
\text{deviation\_score}^m = \frac{1}{\#\text{subsets}} \sum_{S \in \text{subsets}} \frac{\text{dev}_S^m}{\text{max\_dev}_S}
\]

- Mean execution time:

\[
\text{time\_score}^m = \frac{1}{\#\text{subsets}} \sum_{S \in \text{subsets}} \frac{\text{time}_S^m}{\text{max\_time}_S}
\]
Several Algorithms (all limited to 30 seconds):
- Algorithms based on a linear transformation of the quadratic problem solved by integer programming:
  - F2 (●),
  - F24threads (□),
  - F2LP (◊, relaxed problem)
- Algorithms based on Depth first search methods:
  - DF (▽),
  - PDFS (◁),
  - DFUP (△).
- Beam Search: BS (⊙)
- IPFP [Gaüzère et al., 2014] : QAPE (+)
- Bipartite matching [Gaüzère et al., 2014]: LSAPE (×)
Two settings of editing costs

<table>
<thead>
<tr>
<th></th>
<th>vertices</th>
<th></th>
<th>edges</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$C_s$</td>
<td>$C_d$</td>
<td>$C_i$</td>
<td>$C_s$</td>
</tr>
<tr>
<td>Setting 1</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Setting 2</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Average speed-deviation scores on GREYC’s subsets

### Setting:

<table>
<thead>
<tr>
<th></th>
<th>Vertices</th>
<th></th>
<th></th>
<th>Edges</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$c_s$</td>
<td>$c_d$</td>
<td>$c_i$</td>
<td>$c_s$</td>
<td>$c_d$</td>
<td>$c_i$</td>
</tr>
<tr>
<td>Setting</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Examples

Graph Terminology

Graph Matching

Graph Edit distance

### Figures:

- Acyclic Dataset
- Alkane Dataset
- MAO Dataset
- PAH Dataset
Conclusion

- Bipartite Graph edit distance has re initiated a strong interest on Graph edit Distance from the research community
  1. It is fast enough to process large graph databases,
  2. It provides a reasonable approximation of the GED.
- More recently new quadratic solvers for the GED have emerged.
  1. They remain fast (while slower than BP-GED),
  2. They strongly improve the approximation or provide exact solutions.
Bibliography


Examples

Strings

Bipartite Assignment

Graph Terminology

Graph Matching

Graph Edit distance

Distance Guided by Bipartite Matching of Bags of Walks, pages 73–82. Springer Berlin Heidelberg, Berlin, Heidelberg.


