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 YouTube/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>
<th>🐮</th>
<th>🐄</th>
<th>🐐</th>
</tr>
</thead>
<tbody>
<tr>
<td>🐘</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>🐳</td>
<td></td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>🐐</td>
<td></td>
<td></td>
<td>11</td>
<td></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>10</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>11</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>11</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>1</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>9</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>
<td>11</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>
<td></td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>🐉</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>
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.
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 (minutiae),
  - 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.
Statistical vs Structural Pattern Recognition

▶ 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.
Basic objects of Structural Pattern Recognition

- 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.
String Edit distance

- Let $s_1$ ans $s_2$ 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 → restaurantion (insertion of n)
  - restaurantion → restaurantio (removal of n)
  - restaurantio → restaurant (removal of i and o)
  - $d(\text{"restauration"}, \text{"restaurant"}) = 4$

- 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:

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 \( k + l < i + j \).

\[
d(s_1[1 \ldots , i], s_2[1 \ldots , j]) =
\begin{bmatrix}
d(s_1[1 \ldots , i - 1], s_2[1 \ldots , j]) + c_{\text{supp}}(s_1[i]), \\
\text{Min}
\begin{bmatrix}
d(s_1[1 \ldots , i], s_2[1 \ldots , j - 1]) + c_{\text{add}}(s_2[j]), \\
d(s_1[1 \ldots , i - 1], s_2[1 \ldots , j - 1]) + c_{\text{sub}}(s_i[i], s_2[j])
\end{bmatrix}
\end{bmatrix}
\]
String Edit distance

function LEVENSHTEINDISTANCE(char s1[1..m], char s2[1..n])
    for i = 1 → m do
        d[i, 0] ← i
    end for
    for j = 1 → n do
        d[0, j] ← j
    end for
    for j = 1 → n do
        for i = 1 → m do
            d[i, j] ← minimum(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 for
    end for
    return d[m, n]
end function
# String Edit Distance

**Example:**

<table>
<thead>
<tr>
<th></th>
<th>S a t u r d a y</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>0 1 2 3 4 5 6 7 8</td>
</tr>
<tr>
<td>u</td>
<td>1 0 1 2 3 4 5 6 7</td>
</tr>
<tr>
<td>n</td>
<td>2 1 1 2 2 3 4 5 6</td>
</tr>
<tr>
<td>d</td>
<td>3 2 2 2 3 3 4 5 6</td>
</tr>
<tr>
<td>a</td>
<td>4 3 3 3 3 4 3 4 5</td>
</tr>
<tr>
<td>y</td>
<td>5 4 3 4 4 4 3 4 3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Ing e g n e r i a</th>
</tr>
</thead>
<tbody>
<tr>
<td>S a t u r d a y</td>
<td>0 1 2 3 4 5 6 7 8</td>
</tr>
</tbody>
</table>

**Exercise:**

<table>
<thead>
<tr>
<th></th>
<th>Ing e g n e r i a</th>
</tr>
</thead>
<tbody>
<tr>
<td>E c o n o m i a</td>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
</tr>
<tr>
<td>E c o n o m i a</td>
<td>1 2 3 4 5 6 7 8 9 10</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:
  \[
  \begin{cases}
  1 \leq \pi_1(1) \leq \cdots \leq \pi_1(p) = n \\
  1 \leq \pi_2(1) \leq \cdots \leq \pi_2(p) = m
  \end{cases}
  \]
  with unitary increasing and no repetitions:
  \[
  \forall i \in \{1, \ldots, p-1\} \quad \begin{pmatrix}
  \pi_1(i + 1) - \pi_1(i) \\
  \pi_2(i + 1) - \pi_2(i)
  \end{pmatrix} \in \left\{ \begin{pmatrix}
  0 \\ 1
  \end{pmatrix}, \begin{pmatrix}
  1 \\ 0
  \end{pmatrix}, \begin{pmatrix}
  1 \\ 1
  \end{pmatrix} \right\}
  \]
- The Dynamic Time Warping (DTW) distance is then defined as:
  \[
  DTW(x, y) = \min_{\pi \in A(n, m)} \sum_{i=1}^{\vert p \vert} c_{sub}(x_{\pi_1(i)}, y_{\pi_2(i)})
  \]
  where $A(n, m)$ denotes the set of alignments between strings of length $n$ and $m$.  


Global Alignment

▶ Example:

\[
\begin{array}{cccccccccccc}
\text{restauraton} & - & \text{re} & \text{sta} & \text{aur} & \text{at} & \text{ion} & - \\
\text{restaurat} & - & - & - & - & \text{n} & \text{t}
\end{array}
\]

\[
\begin{array}{cccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\
\pi_1: & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 12 & 13 \\
\pi_2: & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10
\end{array}
\]

▶ Connection with edit Distance:

\[
\begin{array}{cccccccccccc}
\text{restauraton} & - & \text{re} & \text{sta} & \text{aur} & \text{at} & \text{ion} & - \\
\text{restaurat} & - & - & - & - & \text{n} & \text{t}
\end{array}
\]

\[
\text{edit script} \quad S \quad S \quad S \quad S \quad S \quad S \quad S \quad S \quad S \quad D \quad D \quad D \quad D \quad S \quad I
\]
String Edit distance

- Complexity in $\mathcal{O}(nm)$
- 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 $s_2[1 \ldots, m]$ ($n \gg m$) 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 $x$. 
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 \( q \in Gr(D) \).

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

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

\[
score(I, D) = \sum_{q\in Gr(x) \cap Gr(y)} score_{I,D}(q)
\]
- Let \( D_{max} \) the database fingerprint with the maximal score, and \( i_{max}, j_{max} \) the locations in \( I \) and \( D_{max} \) providing the highest q gram score.
Unfortunately not all problems may be formulated using unidimensional 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.**
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 vertice)), 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, ...
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 $\mathcal{O}(n^3)$
Mathematical model

To each permutation $\psi$ we associate the permutation matrix:

$$x_{ij} = \begin{cases} 
1 & \text{If } \psi(i) = j \\
0 & \text{otherwize}
\end{cases}$$

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:

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

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:

$$\min_{Ax=1, x \geq 0} c^t x \iff \max_{A^t y \leq c} 1^t y$$

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

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

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

- 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 $\overrightarrow{G_y}$ 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 $\overrightarrow{G_y}$ 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

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 $\vec{G_y}$ 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 $\vec{G_y}$ 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

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 0

<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>$\emptyset$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_S$</td>
<td>$S$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R_T$</td>
<td>$T$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Z$</td>
<td>$S$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\Delta$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
1(0) 2(0) 3(0) 4(0)
1 2 3 4
```

```
4(0) 6(0) 8(0) 10(0)
```

```
3(0) 2(0) 5(0) 9(0)
```

```
[1] 7 8 3
```
A dual method

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

<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

Def:

1. Let $R_S \subseteq S$ and $R_T \subseteq T$ be the vertices not covered by $M$.
2. 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.
3. If $R_T \cap Z \neq \emptyset$, reverse the orientation of a directed path in $\overrightarrow{G_y}$ from $R_S$ to $R_T$.
4. If $R_T \cap Z = \emptyset$, then let:
   $$\Delta := \min\{c_{i,j} - y_i - y_j : i \in Z \cap S, j \in T \setminus Z\}.$$ 
5. Increase $y$ by $\Delta$ on $Z \cap S$ and decrease $y$ by $\Delta$ on $Z \cap T$.

Example: Iteration 2

<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
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

\[
\begin{align*}
E_y &= \{(1, c), (b, 2)\}; a &| &1 & 2 & 3 & 4 \\
R_S &= \{a, b, d\}; &b & 3 & \underline{2} & 5 & 9 \\
R_T &= \{2, 3, 4\}; &c & 1 & 7 & 8 & 3 \\
Z &= \{a, b, d, 2\}; &d & 6 & 4 & 10 & 5
\end{align*}
\]
A dual method

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

$$E_y = \{(1, c), (2, b)\}; a \begin{bmatrix} 4 & 6 & 8 & 10 \\ b & 3 & 2 & 5 & 9 \end{bmatrix}; c \begin{bmatrix} 1 & 7 & 8 & 3 \end{bmatrix}; d \begin{bmatrix} 6 & 4 & 10 & 5 \end{bmatrix}.$$
A dual method

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 5

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

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 6

\[
\begin{array}{c|cccc}
& 1 & 2 & 3 & 4 \\
\hline
E_y & (1, c), (2, b), (a, 4) & 4 & 6 & 8 & 10 \\
R_S & \{a, d\} & & & \\
R_T & \{3, 4\} & b & 3 & 2 & 5 & 9 \\
Z & \{a, d, 1, c, 2, b, 4\} & c & 1 & 7 & 8 & 3 \\
\hline
\end{array}
\]

\[
\begin{array}{c|cccc}
& 1 & 2 & 3 & 4 \\
\hline
a & 6 & 4 & 10 & 5 \\
\end{array}
\]
A dual method

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 $\vec{G}_y$ 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 $\vec{G}_y$ 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

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

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

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

$$
\begin{align*}
R_S &= \{a\}; \\
R_T &= \{3\}; \\
Z &= \{a, 1, c, 4, d, 2\}; \\
T - Z &= \{3\}; \\
\Delta &= 2 \\
\end{align*}
$$

<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>
A dual method

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 9

<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>a</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>$R_T$</td>
<td>{3};</td>
<td>b</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>$Z$</td>
<td>{a, 1, c, 4, 3};</td>
<td>c</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>
A dual method

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 $\vec{G}_y$ 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 $\vec{G}_y$ 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

\[ C = C(c, 1) + C(b, 2) + C(d, 4) + C(a, 3) \]
\[ = 1 + 2 + 5 + 8 \]
\[ = 16 \]
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 \(\bar{c}_{ij} = c_{ij} - u_i - v_j\). Indeed:

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

The main difference is that if \(\bar{c}_{ij}\) 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></td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<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></td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<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

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

\[
\begin{array}{|c|cccc|}
\hline
 & 0 & 3 & 1 & 4 \\
\hline
1 & 2 & 3 & 4 & \\
\hline
4 & a & 0 & 2 & 1 & 5 \\
2 & b & 1 & 0 & 0 & 6 \\
1 & c & 0 & 6 & 4 & 1 \\
4 & d & 2 & 0 & 3 & 0 \\
\hline
\end{array}
\]
A primal/Dual method
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>
</tr>
</thead>
<tbody>
<tr>
<td>4 a</td>
<td>0</td>
<td>2</td>
<td>1 5</td>
</tr>
<tr>
<td>2 b</td>
<td>1</td>
<td>0</td>
<td>0 6</td>
</tr>
<tr>
<td>1 c</td>
<td>0</td>
<td>6</td>
<td>4 1</td>
</tr>
<tr>
<td>4 d</td>
<td>2</td>
<td>0</td>
<td>3 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>×</th>
<th>×</th>
<th>×</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 a</td>
<td>0 3 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 2 3 4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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 to each non-covered column. Return to step 1.

Example

<table>
<thead>
<tr>
<th>×</th>
<th>0</th>
<th>0</th>
<th>3</th>
<th>1</th>
<th>4</th>
<th>0</th>
<th>2</th>
<th>1</th>
<th>5</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 4 | 2 | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 105
A primal/Dual method

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>0</td>
<td>0’</td>
<td>6</td>
</tr>
<tr>
<td>0’</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0’</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{array}{c|cccc}
4 & a & 0 & 1 & 0’ & 4 \\
1 & b & 2 & 0 & 0’ & 6 \\
1 & c & 0’ & 5 & 3 & 0 \\
3 & d & 3 & 0’ & 3 & 0 \\
\end{array}
\]

×
**A primal/Dual method**

**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**

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

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

\[ \begin{align*}
\times \quad u_i & \quad v_j \\
4 & a & 4 & 4 & 4 & 4 \\
1 & b & 2 & 0 & 6 & 0 \\
1 & c & 0 & 5 & 3 & 0 \\
3 & d & 3 & 0 & 3 & 0 \\
\end{align*} \]
Bipartite Assignment algorithm: Matrix version

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.

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
From bipartite Assignment to graph matching

Rem : Image taken from B. Raynal PhD.
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. $\forall (i, j) \in \{1, \ldots, n\}^2, i \neq j$

$$m_{i,j} = \begin{cases} \nu((v_i, v_j)) & \text{if } (v_i, v_j) \in E \\ 0 & \text{else} \end{cases}$$

A permutation matrix $n \times n$, $P = (p_{i,j})$ satisfies:

1. $\forall (i, j) \in \{1, \ldots, n\}^2 \ p_{i,j} \in \{0, 1\}$,
2. $\forall j \in \{1, \ldots, n\} \ \sum_{i=0}^{n} p_{i,j} = 1$,
3. $\forall i \in \{1, \ldots, n\} \ \sum_{j=0}^{n} p_{i,j} = 1$. 
Adjacency/Permutation matrices

\[
M = \begin{pmatrix}
1 & 2 & 3 \\
1 & a & e_2 & e_1 \\
2 & e_2 & b & 0 \\
3 & e_1 & 0 & c
\end{pmatrix},
\quad P = \begin{pmatrix}
1 & 2 & 3 \\
1 & 0 & 1 & 0 \\
2 & 1 & 0 & 0 \\
3 & 0 & 0 & 1
\end{pmatrix}
\]

or

\[
M' = PMP^t = \begin{pmatrix}
1 & 2 & 3 \\
1 & b & e_2 & e_1 \\
2 & e_2 & a & 0 \\
3 & e_1 & 0 & c
\end{pmatrix}
\]
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:

\[ M_2 = PM_1P^t \]

- 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.
Graph Isomorphism and Permutation matrices

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 $G_2$ if and only if there exists a $n \times n$ permutation matrix $P$ such that:

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

Remark:

$$M_1 = S_{m,m}(PM_2P^t) = S_{m,n}(P)M_2(S_{m,n}(P))^t$$
Matching by State Space Representation (SSR).

- A state: A partial matching
- Method: explore successively the different states
- Distinction between different methods:
  - Transition from one state to another
  - Heuristics to avoid infinite loops (consider twice the same state),
  - Heuristics to restrict the search space
Notations:

- $G_1 = (V_2, E_2)$, $G_1 = (V_2, E_2)$ 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?
- $T^{in}_1(s)(T^{out}_1(s))$ set of vertices of $G_1$ predecessors (successors) of a vertex of $M_1(s)$.
- $T^{in}_2(s)(T^{out}_2(s))$ 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(character 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
If $T_1^{out}(s)$ and $T_2^{out}(s)$ non empty

$$P(s) = T_1^{out}(s) \times \{minT_2^{out}(s)\}$$

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

$$P(s) = T_1^{in}(s) \times \{minT_2^{in}(s)\}$$

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

- \( R_{pred}, R_{succ} \): 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)?
$R_{\text{pred}}(s, m, n)$: predecessors match

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

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

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

\[ \downarrow \quad \downarrow \]

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

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

\[
\begin{array}{c}
\downarrow \\
n' \in M_1(s) & \leftrightarrow & m' \in M_2(s)
\end{array}
\]
Successors (predecessors) of $n$ and $m$ must match locally

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

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

\[
\begin{array}{|c|c|c|}
\hline
n & \in G_1(s') & n \in G_1(s') \\
\downarrow & & \uparrow \\
n' \rightarrow n'' & \in G_1(s) & n' \rightarrow n'' \\
\in T_1^{in}(s) & \in G_1(s) & \in T_1^{in}(s) \in G_1(s) \\
\hline
\end{array}
\]
VF2

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

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

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

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

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

- **with:**
  - $N_1(s) = V_1 - M_1(s) - T_1^{in}(s) \cup T_1^{out}(s)$: all that remains to be seen in $G_1$.
  - $N_2(s) = V_2 - M_2(s) - T_2^{in}(s) \cup T_2^{out}(s)$: all that remains to be seen in $G_2$. 
VF2

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

\[ G_2 \]
\( G_1 \)
- \( M(s') = \{(a, 1), (c, 3)\} \)
- \( M_1(s') = \{a, c\}, M_2(s) = \{1, 3\} \)
- \( T_1^{\text{in}}(s) = \{b\}; T_1^{\text{out}}(s) = \{b, e\}; \)
- \( T_2^{\text{in}}(s) = \{2\}; T_2^{\text{out}}(s) = \{2\}; \)
- \( P(s) = (b, 2), (e, 2) \)
- \( \text{Pred}(G_1, e) = \{c\}; \text{Succ}(G_1, e) = \{d\} \)
- \( \text{Pred}(G_2, 2) = \{3\}; \text{Succ}(G_2, 2) = \{1\} \)
- \( (e, 2) \) violate predicate \( R_{\text{succ}}(s', e, 2) \). Indeed:
  - \( 1 \in M_2(s) \cap \text{Succ}(G_2, 2) \) or \( \text{Succ}(G_1, e) \cap M_1(s) = \emptyset \)
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>( \mathcal{O}(N^2) )</td>
<td>( \mathcal{O}(N!N) )</td>
<td>( \mathcal{O}(N^3) )</td>
<td>( \mathcal{O}(N!N^2) )</td>
</tr>
<tr>
<td>Space</td>
<td>( \mathcal{O}(N) )</td>
<td>( \mathcal{O}(N) )</td>
<td>( \mathcal{O}(N^3) )</td>
<td>( \mathcal{O}(N^3) )</td>
</tr>
</tbody>
</table>

- Usable for large graphs (up to 1000 vertices).
Common sub graph problem

- Connection between graph isomorphisms and common sub graphs
  - Given two graphs $G_1$ and $G_2$, $G$ it exists a common sub graph of $G_1$ et $G_2$ iff it exists:
    - $G \xrightarrow{\varphi} G_1$
    - $G \xrightarrow{\psi} G_2$
  - $\varphi, \psi$ sub graph isomorphisms

- $G$ is maximal if it is not a subgraph of a common subgraph,
- $G$ is maximum if we cannot find a common subgraph of $G_1$ and $G_2$ with a greater number of vertices.
- First idea: use a SSR (e.g. VF2) algorithm and remove rules $R_{in}, R_{out}$ et $R_{ne}$ (we no more search to match all vertices of one of the two graphs).
Use of SSR algorithm

\[ \text{procedure MATCHING}(\text{char } s) \text{a matching} \]
\[ \triangleright s_0 \text{ initial state s.t. } M(s_0) = MCS = \emptyset \]
\[ \text{if } M(s) \text{ contains all vertices of } G_2 \text{ then} \]
\[ \quad \text{return } M(s) \]
\[ \text{else} \]
\[ \quad \text{compute } P(s) \]
\[ \quad \text{for each } (n, m) \in P(s) \text{ do} \]
\[ \quad \quad \text{if } F(s, n, m) \text{ then} \]
\[ \quad \quad \quad \text{compute } s' \text{ after the addition of } (n, m) \text{ to } M(s) \]
\[ \quad \quad \quad \text{if } \text{then} \text{size}(M(s')) > \text{sizeMax} \]
\[ \quad \quad \quad \quad \text{sizeMax} = \text{size}(M(s')) \]
\[ \quad \quad \quad \quad \text{MCS} = M(s') \]
\[ \quad \quad \text{end if} \]
\[ \quad \text{MATCHING}(s') \]
\[ \quad \text{end if} \]
\[ \text{end for} \]
\[ \text{end if} \]
\[ \text{Restauration of data structures} \]
\[ \text{end procedure} \]
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

Association Graph

1-a
1-b
1-c
1-d
1-e

2-a
2-b
2-c
2-d
2-e

4-a
4-b
4-c
4-d
4-e

1
2
3
4

a
b
c
d
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.
$SM^i, i = 1, 2$ algorithms

- 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 $K$:
    \[ C_0(K) = \{ j \in V - K \mid \forall k \in K (j, k) \in E \} = \bigcap_{k \in K} N_k \]
procedure \( SM^i \) (graph \( G \))

\( Q: \) clique with cardinal \( i \), \( \mathcal{K}, \mathcal{K}^*, \max \)

\( \mathcal{K} = \emptyset; \)

\( \mathcal{K}^* = \emptyset; \)

\( \max = 0; \)

while \( Q \neq \emptyset \) do

\( H = \text{pop}(Q) \)

\( \mathcal{K} = H \)

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

\( \mathcal{K} = \mathcal{K} \cup K \)

if \( |K| > \max \) then

\( \max = |K| \)

\( \mathcal{K}^* = K \)

end if

end while

end procedure
$SM^i$: 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 optimaums.
SM\textsuperscript{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 \((l, k_l) \notin E\). 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 improves the optimum
  - does not allows us to get closer from 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
function SELECT\((G, K, last\_\_swap)\)

\[ l : \text{vertex}; \]

if \( n\_\_\text{swap} \leq T|K| \) and \(|K| \geq \text{START\_SWAP} \) then

\[ l = \arg \max_{j \in C_0(K) \cup C_1(K)} |C_0(K) \cap N_j| \]

if \( l = \text{last\_\_swap} \) 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.
procedure

$SM^1_{SWAP}$ (graph $G = (V, E)$)

int max = 0;

list of cliques $K = \emptyset$,
optimal clique $K^* = \emptyset$,
queue $W = V$

while $W \neq \emptyset$ do

$h = \text{pop}(W)$

$K = \{h\}$

$n_{\text{swap}} = 0; last_{\text{swap}} = \emptyset$

$l = \text{select}(G, K, last_{\text{swap}})$

while $l \neq \emptyset$ do

if $l \in C_0(K)$ then

$K = K \cup \{l\}$

else

$n_{\text{swap}} = n_{\text{swap}} + 1$

$last_{\text{swap}} = k_l$

$K = K \cup \{l\} - \{k_l\}$

push($W, k_l$)

end if

end while

if $|K| > \max$ then

$\max = |K|$

$K^* = K$

end if

end while

$l = \text{select}(G, K, last_{\text{swap}})$

end while

end procedure
$SM^1_{SWAP}$: Performances

- clique size/ max size: $\frac{|K|}{\omega(G)}$

- Execution times (seconds):
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 $C$:

$$x^C = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

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 $n$:

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

$$g(x) = x^t A_G x + \frac{1}{2} x^t x = x^t A x$$

with

$$A = A_G + \frac{1}{2} I$$

$x \in S_n$

$x^*$ is a strict maxima of $g$ iff:

$$\exists \epsilon > 0 \mid \begin{cases} 
|y - x^*| < \epsilon \\
|y - x^*| < \epsilon \text{ and } f(y) = f(x^*)
\end{cases} \Rightarrow f(y) \leq f(x^*)$$

$$\Rightarrow y = x^*$$
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:

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

- 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 $A_{\overline{G}} = (\overline{a}_{i,j})$ with $\overline{a}_{i,j} = 1$ if $(i, j) \notin E$, 0 otherwise.

$$A_{\overline{G}} = ee^t - A_G - I$$

Problem transformation:

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

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

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

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

$$= 1 - g(x)$$
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

\[ f(x) = x^t A x = x^t (A_G + \frac{1}{2} I) x \]
\[ = \sum_{i=1}^{n} \frac{1}{2} x_i^2 + \sum_{j \mid (i,j) \notin E} x_i x_j \]

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

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

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:

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

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

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

$\Leftrightarrow$

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

$\Leftrightarrow$

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

- Application to valued 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

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

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

with

\[ C(w)_{i,j} = \begin{cases} 
\frac{1}{2w_i} & \text{If } i = j \\
\frac{1}{2w_i} + \frac{1}{2w_j} & \text{If } i \neq j \text{ and } (i,j) \notin E \\
0 & \text{otherwise}
\end{cases} \]

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

- Problem: minimise over $S_n$
  
  \[ f(x) = x^t Ax \]

- Formulation in terms of «Linear Complementarity Problem» (LCP)
  
  Found $y, \bar{x}$ such that:
  
  \[ y = q_G + M_G \bar{x} \geq 0, \bar{x} = [x, x_{n+1}, x_{n+2}], x^t y = 0 \]

  \[
  q_G = \begin{pmatrix}
  0 \\
  \vdots \\
  0 \\
  -1 \\
  1
  \end{pmatrix}
  \text{ et } M_G = \begin{pmatrix}
  A & -e & e \\
  e^t & 0 & 0 \\
  -e^t & 0 & 0
  \end{pmatrix}
  \]
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 → PBH algorithm.

The PBH algorithm is 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
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:
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:

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

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
    \textbf{while } \beta < \beta_f \textbf{ do}
        m_i \leftarrow e^{\beta X_j}
        m_i \leftarrow \frac{m_i}{\sum_{i=1}^{n} m_i}
        \text{Do other parts of the algorithm}
        \text{increase } \beta
    \textbf{end while}
    \textbf{return} \{m_1, \ldots, m_n\}
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 line and then on columns

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

**Output :** \( M \)

\[
\beta \leftarrow \beta_0 \\
\textbf{while} \ \beta < \beta_f \ \textbf{do} \\
\textbf{repeat} \\
\quad M_{a,i} \leftarrow e^{\beta X_{ai}} \\
\quad \textbf{repeat} \\
\quad \quad M_{a,i} \leftarrow \frac{M_{a,i}}{\sum_{j=1}^{T} M_{a,j}} \\
\quad \quad M_{a,i} \leftarrow \frac{M_{a,i}}{\sum_{x=1}^{A} M_{x,i}} \\
\quad \textbf{until} \ M \ \text{converge} \\
\quad \text{Do the remaining part of the algorithm} \\
\quad \text{Increment} \ \beta \\
\textbf{end while} \\
\textbf{end procedure} \\

- \( M \) converges toward a permutation matrix
Problem: The matching is not a research of a max

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

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:

\[E_{wg}(M) \approx E_{wg}(M^0) + \sum_{a=1}^{A} \sum_{i=1}^{I} \frac{\partial E_{wg}(M)}{\partial M_{ai}} (M) |_{M=M^0} (M_{a,i} - M_{a,i}^0)\]
We have thus:

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

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

with:

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

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

we have:

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

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

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 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

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

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

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.
Graph edit distance: An heuristic

1. Let $G_1$ and $G_2$ be two graphs
2. Compute a signature of each node of both graphs,
3. Define a cost function $c$ between nodes.
4. Define $S = V_1 \cup \{\epsilon\}^{|V_2|}$ and $T = V_2 \cup \{\epsilon\}^{|V_1|}$
5. Munkres algorithm applied on $S$ and $T$ provides the optimal matching $\varphi$ between $V_1$ and $V_2$ which minimizes:

$$\sum_{x \in \hat{V}_1} c(x \rightarrow \varphi(x)) + \sum_{x \in V_1 - \hat{V}_1} c(x \rightarrow \epsilon) + \sum_{y \in V_2 - \varphi(\hat{V}_1)} c(\epsilon \rightarrow y)$$

Rem: $\hat{V}_1 \subset V_1$.
6. Define the edit path which:
   - Substitute any $x$ of $\hat{V}_1$ by $\varphi(x)$,
   - Substitute any $(x,y)$ of $E_1 \cap \hat{V}_1 \times \hat{V}_1$ by $(\varphi(x), \varphi(y))$
   - Remove any vertex in $V_1 - \hat{V}_1$
   - Insert any vertex in $V_2 - \varphi(\hat{V}_1)$ and any edge in $V_2 \times V_2 - \varphi(\hat{V}_1) \times \varphi(\hat{V}_1)$.
7. Consider it as the optimal edit path and returns its cost as the edit distance.