Structural Pattern Recognition

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

Aims of Pattern Recognition

Design "Intelligent Systems"

- Intelligence $\leftarrow$ 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.**

Wechsler, D (1944). The measurement of adult intelligence. Baltimore: Williams & Wilkins. ISBN 0-19-502296-3.

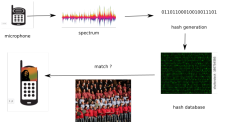
Aims of Pattern Recognition

- 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.

$\Rightarrow$ 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)



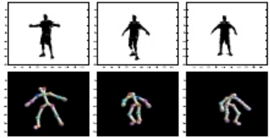
  - Copyright protection for You Tube/Dalymotion...
  - New services for mobile devices
- Voice/Song representation:
  - A vector (set of Fourier coefficients,Wavelet transform,...)
  - A function
  - **A string**

Shapes

Confusion matrix

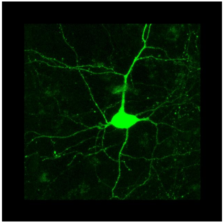
Classes									
	11								
		11							
			11						
				10			1		
					11				
				1		9	1		
							11		
		1						10	

Shapes

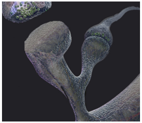


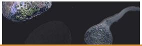
- Applications:
  - Character recognition,
  - Docking,
  - Identification of signatures,
  - Pose estimation
  - Detection & Characterization of spines

⋮



- Shape representations:
  - Vector of features (e.g. Legendre's moments),
  - String representation of the boundary
  - Graph representation of the skeleton.

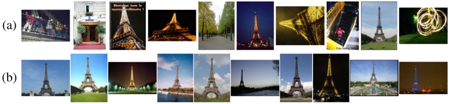




Image/Video indexation

- Image Classification/Retrieval

  All images related to a subject



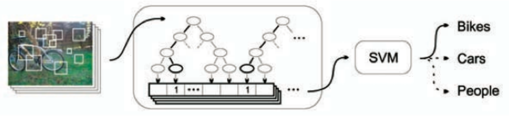
(a) classic research engine, (b) [CVPR10]

- Objects Detection

  Learn : object's appearances and relationships with context.

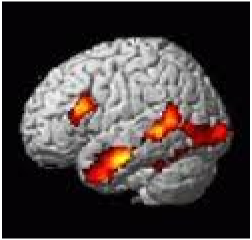
- Visual Words

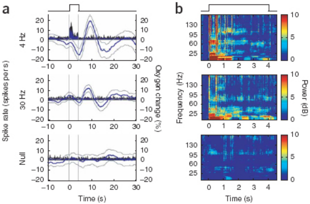
  Intermediate representation adapted to object representations.



Brain

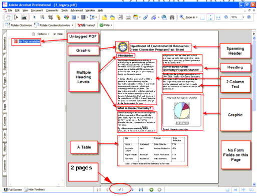
- Decode brain activity.
  1. Submit stimuli to subjects,
  2. Measure brain activities,
  3. Retrieve stimuli from brain activities.
- Activity measurements:
  1. Set of signals (for each active zone),
  2. Graph of zones with correlated signals.

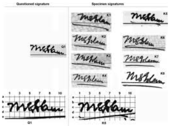


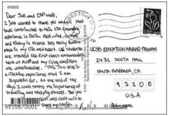


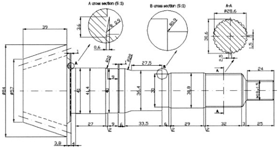
Document Analysis

- 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,



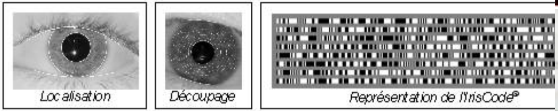






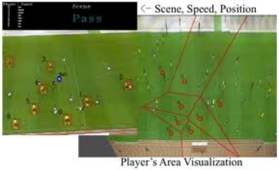
Biometry

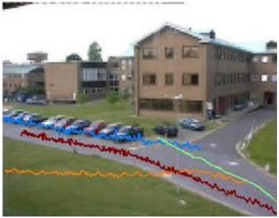
- People identification by:
  - Fingers ++,
  - Hands +,
  - Eyes +++,
  - Face +,
  - Voice,
- Features:
  - Set of points (minuties),
  - Measure of texture,...



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...

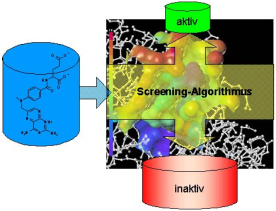


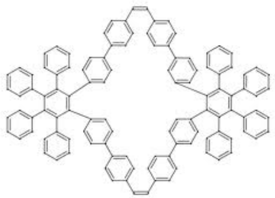


Chemoinformatics

Building a new molecule requires many attempts and many test: 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.



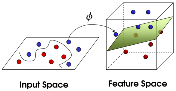


Different Levels/Steps of Recognition

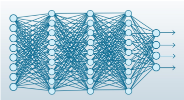
Level 0: Hand made classification (Expert Systems)

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^∞.
- 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(“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. Exemple: The Pascal triangle:

$$\binom{n}{p} = \binom{n-1}{p-1} + \binom{n-1}{p}$$

$\binom{10}{7} = \binom{9}{6} + \binom{9}{7}$ but $\binom{9}{6} = \binom{8}{5} + \binom{8}{6}$ and $\binom{9}{7} = \binom{8}{6} + \binom{8}{7}$.

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

String Edit distance

- 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 $k + l < i + j$.

$$d(s_1[1\ldots,i], s_2[1\ldots,j]) = Min\begin{pmatrix} d(s_1[1\ldots,i-1], s_2[1\ldots,j]) + c_{supp}(s_1[i]), \\ d(s_1[1\ldots,i], s_2[1\ldots,j-1]) + c_{add}(s_2[j]), \\ d(s_1[1\ldots,i-1], s_2[1\ldots,j-1]) + c_{sub}(s_i[i], s_2[j]) \end{pmatrix}$$

- Ex:

$$d(toto, tata) = Min\begin{pmatrix} d(tot, tata) + c_{supp}(o) \\ d(toto, tat) + c_{add}(a) \\ d(tot, tat) + c_{sub}(o, a) \end{pmatrix}$$

String Edit distance

Algorithm

function LEVENSHTEINDISTANCE(char $s_1[1..m]$, char $s_2[1..n]$)

  $d[0,0] \leftarrow 0$

  for $i = 1 \rightarrow m$ **do**

    $d[i,0] \leftarrow d[i-1,0] + c_{supp}(s_1[i])$ ▷ Removal of s_1 prefixes

  end for

  for $j = 1 \rightarrow n$ **do**

    $d[0,j] \leftarrow d[0,j-1] + c_{add}(s_2[j])$ ▷ Insertion of s_2 prefixes

  end for

  for $j = 1 \rightarrow n$ **do**

    for $i = 1 \rightarrow m$ **do**

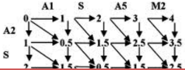
$$d[i,j] \leftarrow min \begin{pmatrix} d[i-1,j] + c_{supp}(s_1[i]), \\ d[i,j-1] + c_{add}(s_2[j]), \\ d[i-1,j-1] + c_{sub}(s_1[i], s_2[j]) \end{pmatrix}$$

    end for

  end for

  return $d[m,n]$

end function



String Edit distance

Illustration : all costs equal to 1

- Example :

		S	a	t	u	r	d	a	y
	0	1	2	3	4	5	6	7	8
S	1	0	1	2	3	4	5	6	7
u	2	1	1	2	2	3	4	5	6
n	3	2	2	2	3	3	4	5	6
d	4	3	3	3	3	4	3	4	5
a	5	4	3	4	4	4	4	3	4
y	6	5	4	4	5	5	5	4	3

- Exercise:

		I	n	g	e	g	n	e	r	i	a
	0	1	2	3	4	5	6	7	8	9	10
E	1										
c	2										
o	3										
n	4										
o	5										
m	6										
i	7										
a	8										

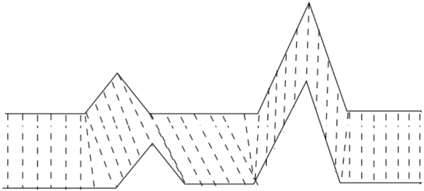
Sting Edit distance and Global Alignment

- Let $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_m)$ denote 2 strings.
- An alignement π between x and y is defined by two vectors (π_1, π_2) of length $p < n + m - 1$ such that:

$$\left\{ \begin{array}{l} 1 \leq \pi_1(1) \leq \cdots \leq \pi_1(p) = n \\ 1 \leq \pi_2(1) \leq \cdots \leq \pi_2(p) = m \end{array} \right.$$

with unitary increasing and no repetitions:

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



Global Alignement

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

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

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

DTW: Algorithm

function DTW(s_1: array[1..m], s_2: array[1..n])
    $DTW \leftarrow array[0,..,m][0,..,n]$
    for $i = 0 \rightarrow m$ **do**
        for $j = 0 \rightarrow n$ **do**
            $DTW[i,j] \leftarrow +\infty$
        end for
    end for
    $DTW[0,0] \leftarrow 0$
    for $i = 1 \rightarrow m$ **do**
        for $j = 1 \rightarrow n$ **do**
            cost$\leftarrow d(s_1[i], s_2[j])$

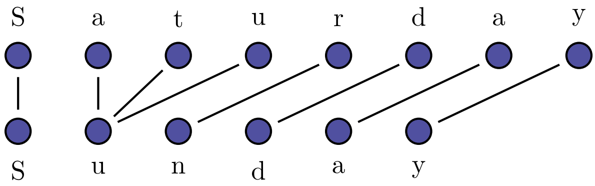
$$DTW[i,j] \leftarrow cost + \min \left(\begin{array}{l} DTW([i-1,j]) \\ DTW([i,j-1]) \\ DTW([i-1,j-1]) \end{array} \right)$$

        end for
    end for
    return $DTW[n,m]$
end function

DTW

- Example :

		S	a	t	u	r	d	a	y
	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
S	$+\infty$	0	1	2	3	4	5	6	7
u	$+\infty$	1	1	2	2	3	4	5	6
n	$+\infty$	2	2	2	3	3	4	5	6
d	$+\infty$	3	3	3	3	4	3	4	5
a	$+\infty$	4	3	4	4	4	4	3	4
y	$+\infty$	5	4	4	5	5	5	4	3



- Exercise:
  - Ingegneria
  - Economia

String Edit distance/DTW

- 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,

Q grams

- Due to its complexity, string edit distance is a bit useless on very long strings (songs, DNA,long texts,...).
- $\rightarrow$ 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 :

$$\begin{array}{rcl} D_q(x,y) & = & \sum_{s\in\Sigma^q}|H_s(x)-H_s(y)| \\ & = & \sum_{s\in Gr(x)\cap Gr(y)}|H_s(x)-H_s(y)|+\sum_{s\in Gr(x)-Gr(y)}|H_s(x)| \\ & & +\sum_{s\in Gr(y)-Gr(x)}|H_s(y|) \end{array}$$

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

Q gram and edit distance

- 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:

$$score(I) = score(I[i_{max}, i_{max} + M], D[j_{max}, j_{max} + M])$$

Assignment between sets

- Given two sets of objects $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_m\}$ and a distance $d(.,.)$ between objects, assign each a_i to one (usually the closest) b_j.
- Some heuristics:
  - maps each a_i to $\arg\min_{j \in \{1,\ldots,m\}} d(a_i, b_j)$
  - filter the result: a_i is mapped onto b_k if:

$$\left\{ \begin{array}{lcl} b_k & = & \arg\min_{j \in \{1,\ldots,m\}} d(a_i, b_j) \\ & & \frac{d(a_i,b_k)}{\arg\min_{j \in \{1,\ldots,m\}, j \neq k} d(a_i,b_j)} < \epsilon \end{array} \right.$$

- Used for 3D reconstruction, SLAM,. . . .
- No criterion of optimality

Bipartitite Assignement

- 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 assigment algorithm is to determine a bijective mapping function $\psi : S \rightarrow T$ (a permutation) from Engineers to Projects which minimizes:

$$\sum_{i \in S} c(i, \psi(i))$$

  ψ is an optimal assigment minimizing the mapping of Engineers to projects.
- This problem may be formalize as the one of determining a mapping on a bipartite weighted graph $G = (S \cup T, E, \c{})$ where $E \subset S \times T$.
- This problem is solved in $\mathcal{O}(n^3)$

Mathématical model

To each permutation ψ 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:

$$x^* = \operatorname*{argmin}_x \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

such that:

- Each line of x sums to 1:

$$\forall i \in \{1, \dots, n\} \; \sum_{j=1}^{n} x_{ij} = 1$$

- Each column of x sums to 1:

$$\forall j \in \{1, \dots, n\} \; \sum_{i=1}^{n} x_{ij} = 1$$

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

Problem reformulation

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

$$
\begin{array}{c c}
 & \begin{array}{cccc|cccc|c|cccc} x_{11} & & \dots & x_{1n} & x_{21} & & \dots & x_{2n} & \dots & x_{n1} & & \dots & x_{nn} \end{array} \\
\begin{array}{c} 1 \\ \\ \\ n \\ n+1 \\ \\ \\ 2n \end{array} &
\left(\begin{array}{cccc|cccc|c|cccc}
1 & 1 & \dots & 1 & & & & & & & & & \\
 & & & & 1 & 1 & \dots & 1 & & & & & \\
 & & & & & & & & \dots & & & & \\
 & & & & & & & & & 1 & 1 & \dots & 1 \\
\hline
1 & & & & 1 & & & & & 1 & & & \\
 & 1 & & & & 1 & & & & & 1 & & \\
 & & \dots & & & & \dots & & \dots & & & \dots & \\
 & & & 1 & & & & 1 & & & & & 1
\end{array}\right)
\end{array}
$$

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 :

$$\underset{Ax=1, x\geq 0}{Min} c^t x \Leftrightarrow \underset{A^t y \leq c}{Max} 1^t y$$

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

Hungarian Bipartite Method

Defs (1/2)

- Rem. Source : Wikipedia.
- 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 intialised 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 $\overrightarrow{G_y}$ 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 Δ on the vertices of $Z \cap S$ and decrease y by Δ 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.

Hungarian Bipartite: Example

Defs.

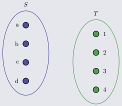
- Let $R_S \subseteq S$ and $R_T \subseteq T$ be the vertices not covered by M.
- Let Z be the set of vertices reachable in $\overrightarrow{G_y}$ from R_S by a directed path only following edges that are tight.
- If $R_T \cap Z \neq \emptyset$, reverse the orientation of a directed path in $\overrightarrow{G_y}$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

Example

	1	2	3	4
a	4	6	8	10
b	3	2	5	9
c	1	7	8	3
d	6	4	10	5



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

$$\begin{aligned}
\textstyle\sum_{i=1}^n \sum_{j=1}^n (c_{ij} - u_i - v_j)x_{ij} &= \textstyle\sum_{i=1}^n \sum_{j=1}^n c_{ij}x_{ij} - \sum_{i=1}^n u_i \sum_{j=1}^n x_{ij} \\
&\quad \textstyle - \sum_{j=1}^n v_j \sum_{i=1}^n x_{ij} \\
&= \textstyle\sum_{i=1}^n \sum_{j=1}^n c_{ij}x_{ij} - \underbrace{\displaystyle\sum_{i=1}^n u_i - \sum_{j=1}^n v_j}_{\text{constant}}
\end{aligned}$$

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.

Hungarian algorithm: Matrix version

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.

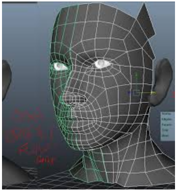
- For each non covered zero, mark it by a prime
  - If there is a selected zero on the line uncover the column and cover the line
  - If there is no selected zero on the line, we do not have selected enough independent zero. Goto step 3.
- If there is no more uncovered zero. Goto step 4.

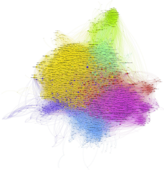
Step 3: Let z_0 be the only uncovered 0 and z_1 the selected 0 on its column. Let z_i (i odd) the $0'$ on the line of z_{i-1} and z_i (i even) the selected 0 on the column of z_{i-1} if it exists (otherwise we stop). Exchange $0'$ and selected 0. Removes the primes of zeros and the lines and columns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

Step 4: Take the minimal value of uncovered elelements found in step 2. Add this value to each covered line and substract it to each non covered column. Return to step 1.

From Strings to Graphs

- 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 vertice (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:

$$X \cong Y \text{ or } X \subseteqq Y.$$

- Graph isomorphism $G = (V, E), G' = (V', E')$ $(X \cong Y)$
  - $|V| = |V'|$ and
  - it exists $\phi : V \to V'$ bijective such that:

$$(v_1, v_2) \in E \Leftrightarrow (\phi(v_1), \phi(v_2)) \in E'$$

- Partial sub graph isomorphism $(X \subseteqq Y)$
  - $|V| \leq |V'|$ 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)

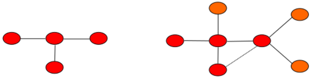
- Sub graph isomorphism
  - Same as partial sub graph isomorphism but with the additional constraint:

$$\forall (v_1, v_2) \in V^2 \ (v_1, v_2) \notin E \Rightarrow (\phi(v_1), \phi(v_2)) \notin E'$$

  We have thus:

$$(\phi(v_1), \phi(v_2)) \in E' \Leftrightarrow (v_1, v_2) \in E$$

- 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:

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

- 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.

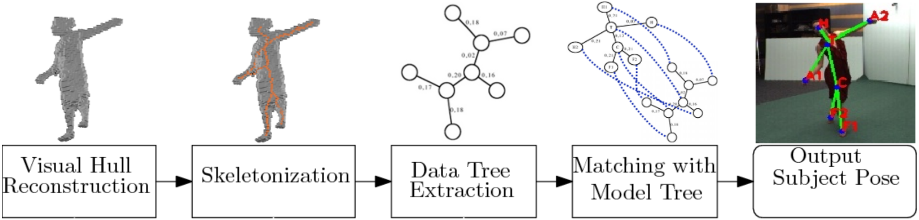
Maximal vs maximum sub graph

- (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,...

From bipartite Assignment to graph matching



Rem : Image taken from B. Raynal PhD.

Labeled Graphs

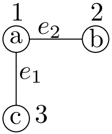
- Definitions :
  - A **labeled graph** :$G = (V, E, \mu, \nu, L_v, L_e)$

$$\left\{ \begin{array}{llcll} \mu & : V & \rightarrow & L_v & \text{vertex's label function} \\ \nu & : E & \rightarrow & L_e & \text{edge's label function} \end{array} \right.$$

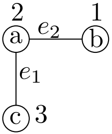
  - A **labeled sub graph** $G_s = (V_s, E_s, \mu_s, \nu_s, L_v, L_e)$ of $G = (V, E, \mu, \nu, L_v, L_e)$.
    - μ_s and ν_s restriction of μ and ν to $V_s \subset V$ and $E_e \subset E$ (with $E_s = E \cap V_s \times V_s$).
  - The **adjacency matrix** $M = (m_{i,j})$ of a graph $G = (V, E, \mu, \nu, L_v, L_e)$ is defined by:
    1. $\forall i \in \{1, \dots, n\}\ m_{i,i} = \mu(v_i)$
    2. $\forall (i, j) \in \{1, \dots, n\}^2, i \neq j$

$$m_{i,j} = \left\{ \begin{array}{ll} \nu((v_i, v_j)) & \text{if } (v_i, v_j) \in E \\ 0 & \text{else} \end{array} \right.$$

Adjacency/Permutation matrices



or



$$M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} a & e_2 & e_1 \\ e_2 & b & 0 \\ e_1 & 0 & c \end{pmatrix} \end{array}, \; P = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

$$M' = PMP^t = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} b & e_2 & e_1 \\ e_2 & a & 0 \\ e_1 & 0 & c \end{pmatrix} \end{array}$$

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

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

  - $S_{k,k}(M)$ 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 iff it exists a $n \times n$ permutation matrix P such that:

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

- Remark:

$$M_1 = S_{m,m}(PM_2P^t) = S_{m,n}(P)M_2\left(S_{m,n}(P)\right)^t$$

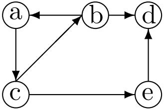
Matching by State Space Representation (SSR).

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

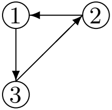
SSR : Algorithme de Cordella 2001 (VF2)

- Notations :
  - $G_1 = (V_2, E_2)$, $G_1 = (V_2, E_2)$ two *oriented* graphs,
  - We seach 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



G_2

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

VF2 algorithm

procedure MATCHING(char s)a matching
▷ s_0 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

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.

VF2

Computation of $F(s, n, m)$

$$
\begin{aligned}
F(s,n,m) \quad = \quad & R_{pred}(s,n,m) \wedge R_{succ}(s,n,m) \wedge \\
& R_{in}(s,n,m) \wedge R_{out}(s,n,m) \wedge R_{new}(s,n,m)
\end{aligned}
$$

- 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) ?

matching of predecessors: R_{pred}

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

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

$$\begin{array}{ccc} n' \in M_1(s) & \leftrightarrow & m' \in M_2(s) \\ \downarrow & & \downarrow \\ n & \leftrightarrow & m \end{array}$$

VF 2

matching of successors: R_{succ}

- $R_{succ}(s, m, n)$: successors match

$$(\forall n' \in M_1(s) \cap Succ(G_1, n) \exists m' \in Succ(G_2, m) | (n', m') \in M(s)) \wedge$$
$$(\forall m' \in M_2(s) \cap Succ(G_2, m) \exists n' \in Succ(G_1, n) | (n', m') \in M(s))$$

$$\begin{array}{ccc} n & \leftrightarrow & m \\ \downarrow & & \downarrow \\ n' \in M_1(s) & \leftrightarrow & m' \in M_2(s) \end{array}$$

VF2

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

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

$$
\begin{array}{l}
\left(\left|T_1^{in}(s)\cap Succ(G_1,n)\right| \geq \left|T_2^{in}(s)\cap Succ(G_2,m)\right|\right)\wedge \\
\left(\left|T_1^{in}(s)\cap Pred(G_1,n)\right| \geq \left|T_2^{in}(s)\cap Pred(G_2,m)\right|\right)
\end{array}
$$

n	$\in G_1(s')$		$n \in G_1(s')$		
$\downarrow$			$\uparrow$		
n'	$\rightarrow$	n''	n'	$\rightarrow$	n''
$\in T_1^{in}(s)$		$\in G_1(s)$	$\in T_1^{in}(s)$		$\in G_1(s)$

VF2

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

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

$$
\begin{array}{l}
(|T_1^{out}(s)\cap Succ(G_1,n)| \geq |T_2^{out}(s)\cap Succ(G_2,m)|)\wedge \\
(|T_1^{out}(s)\cap Pred(G_1,n)| \geq |T_2^{out}(s)\cap Pred(G_2,m)|)
\end{array}
$$

$$
\begin{array}{|ccc|ccc|}
\hline
n & \in G_1(s') & & n \in G_1(s') & & \\
\downarrow & & & \uparrow & & \\
n' & \leftarrow & n'' & n' & \leftarrow & n'' \\
\in T_1^{out}(s) & & \in G_1(s) & \in T_1^{out}(s) & & \in G_1(s) \\
\hline
\end{array}
$$

- Replace $\geq$ by $=$ for the graph isomorphism

Predicate R_{new}

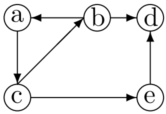
- Successors and predecessors must match outside sets $M_i(s), T_i^{in}(s)$ et $T_i^{out}(s)$, $i = 1, 2$.
  - $R_{new}(s, n, m)$

$$(|N_1(s) \cap Succ(G_1, n)| \geq |N_2(s) \cap Succ(G_2, m)|) \wedge$$
$$(|N_1(s) \cap Pred(G_1, n)| \geq |N_2(s) \cap Pred(G_2, m)|)$$

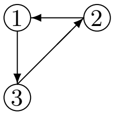
  - with:
    - $N_1(s) = V_1 - M_1(s) - T_1^{in}(s) \cup T_1^{out}(s)$: all that remains to be seen in G_1.
    - $N_2(s) = V_2 - M_2(s) - T_2^{in}(s) \cup T_2^{out}(s)$: all that remains to be seen in G_2.

VF2

Example



G_1



G_2

VF2

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

	VF2		Ullman	
Complexity	Best case	Worse case	Best case	Worse case
Time	$\mathcal{O}(N^2)$	$\mathcal{O}(N!N)$	$\mathcal{O}(N^3)$	$\mathcal{O}(N!N^2)$
Space	$\mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(N^3)$	$\mathcal{O}(N^3)$

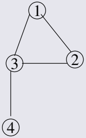
- Usable for large graphs (up to 1000 vertices).

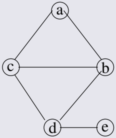
Association graph

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

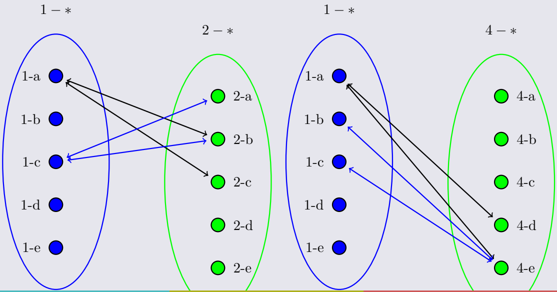
Association graph

Input Graphs



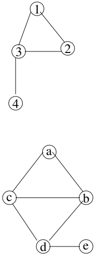


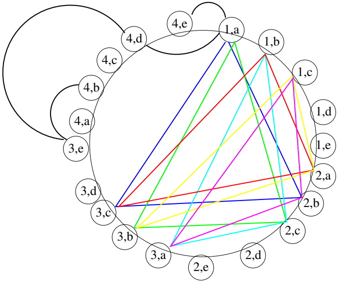
Association Graph



Association Graph

Example





- 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.

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

Characteristic vector

- 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:

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

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:

$$g(x) = x^t A_G x + \frac{1}{2} x^t x = x^t A x \text{ with } \begin{cases} A = A_G + \frac{1}{2} I \\ x \in S_n \end{cases}$$

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

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

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:

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

- 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 procesus converge to a stationnary point $(x_i(t+1) = x_i(t))$ which corresponds to a local maximum of g.

Gibbons, Bomze (97)

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

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

- Problem transformation:

$$\begin{array}{rcl}
f(x) & = & x^t\overline{A}x = x^t\left(A_{\overline{G}} + \frac{1}{2}I\right)x \\
& = & x^t\left(ee^t - A_G - I + \frac{1}{2}I\right)x \\
& = & x^t\left[ee^t - \left(A_G + \frac{1}{2}I\right)\right]x \\
& = & x^t ee^t x - x^t\left(A_G + \frac{1}{2}I\right)x \\
& = & 1 - g(x)
\end{array}$$

Gibbons, Bomze(97) theorem

- 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

$$\begin{array}{rcl} f(x) & = & x^t\overline{A}x = x^t(A_{\overline{G}} + \frac{1}{2}I)x \\ & = & \sum_{i=1}^n \frac{1}{2}x_i^2 + \sum_{j\|(i,j)\notin E} x_i x_j \end{array}$$

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

$$\begin{array}{rcl} f(x^C) & = & \frac{|C|}{2|C|^2} + \sum_{i=1}^n \sum_{j\|(i,j)\notin E} x_i^C x_j^C \\ & = & \frac{1}{2|C|} + \sum\sum_{(i,j)\in C^2\|(i,j)\notin E} x_i^C x_j^C \end{array}$$

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:

$$\begin{array}{rcl} f(x^{C'}) & = & \frac{1}{2(|C|+1)} + \sum\sum_{(i,j)\in C'^2 | (i,j)\notin E} x_i^{C'} x_j^{C'} \\ & = & \frac{1}{2(|C|+1)} + \frac{1}{(|C|+1)^2} \end{array}$$

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

$$\begin{array}{lcl} \frac{1}{2(|C|+1)} + \frac{1}{(|C|+1)^2} & > & \frac{1}{2|C|} \\ & \Updownarrow & \\ |C|+1+2 & > & \frac{(|C|+1)^2}{|C|} = |C| + 2 + \frac{1}{|C|} \\ & \Updownarrow & \\ 3 & > & 2 + \frac{1}{|C|} \end{array}$$

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

$$W(S) = \sum_{i \in S} w_i$$

Weighted 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

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

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

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

$$q_G = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ -1 \\ 1 \end{pmatrix} \text{ et } M_G = \begin{pmatrix} A & -e & e \\ e^t & 0 & 0 \\ -e^t & 0 & 0 \end{pmatrix}$$

Resolution : Remarks

- 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 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 :

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

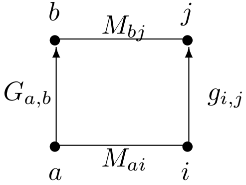
  with A (resp. I) nb vertices in G (resp. g).
- i.e. match a maximum of similar edges

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 $\{X_1, \dots, X_n\}$ determine $\{m_1, \dots, m_n\}$ 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:

$$\lim_{\beta \to +\infty} m_j(\beta) = \begin{cases} 1 & \text{if } X_j = \max X_i \\ 0 & \text{otherwise} \end{cases}$$

SoftAssign : Algorithm 1

function SOFTASSIGN1($\{X_1, \ldots, X_n\}$)
    $\beta \leftarrow \beta_0$
    while $\beta < \beta_f$ **do**
        $m_i \leftarrow e^{\beta X_j}$
        $m_i \leftarrow \frac{m_i}{\sum_{i=1}^{n} m_i}$
        Do other parts of the algorithm
        increase β
    end while
    return $\{m_1, \ldots, m_n\}$
end function

- We determine the max softly (hence the name soft assign)

SoftAssign : Algorithm 2

- 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 β
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

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

- 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:

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

SoftAssign : Toward the solution

- We have thus:

$$\begin{array}{rcl} E_{wg}(M) & = & -\frac{1}{2}\sum_{a=1}^{A}\sum_{i=1}^{I}\sum_{b=1}^{A}\sum_{j=1}^{I} M_{a,i}M_{b,j}C_{a,i,b,j} \\ & \approx & E_{wg}(M^0) + \sum_{a=1}^{A}\sum_{i=1}^{I}\frac{\partial E_{wg}}{\partial M_{ai}}(M^0)(M_{a,i}-M^0_{a,i}) \end{array}$$

with:

$$\frac{\partial E_{wg}}{\partial M_{a,i}}(M^0) = -\sum_{b=1}^{A}\sum_{j=1}^{I} M^0_{b,j}C_{a,i,b,j}$$

Softassign: toward the solution

- Lets define $Q_{a,i} = -\frac{\partial E_{wg}(M^0)}{\partial M_{a,i}}$
- we have:

$$\begin{array}{rcl} E_{wg}(M) & \approx & E_{wg}(M^0) - \sum_{a=1}^{A} \sum_{i=1}^{I} Q_{ai}(M_{a,i} - M^0_{a,i}) \\ & \approx & Cte - \sum_{a=1}^{A} \sum_{i=1}^{I} Q_{a,i} M_{a,i} \end{array}$$

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

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

$\boxed{\text{A problem of computation of a max !}}$

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 β

- 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 : β, M

$\beta \leftarrow \beta_0$
$\tilde{M}_{a,i} \leftarrow 1 + \epsilon$
while $\beta < \beta_f$ **do**
  repeat
    $Q_{a,i} \leftarrow -\frac{\partial E_{wg}(M)}{\partial M_{a,i}}$
    $M^0_{a,i} \leftarrow e^{\beta Q_{a,i}}$
    repeat
      $\tilde{M}^1_{a,i} \leftarrow \frac{\tilde{M}^0_{a,i}}{\sum_{i=1}^{I+1} \tilde{M}^0_{a,i}}$
      $\tilde{M}^0_{a,i} \leftarrow \frac{\tilde{M}^1_{a,i}}{\sum_{a=1}^{A+1} \tilde{M}^1_{a,i}}$
    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 $C^{(1)}_{a,i}$ encoding distances between vertices

$$E_{arg}(M) = \begin{array}{l} -\frac{1}{2}\sum_{a=1}^{A}\sum_{i=1}^{I}\sum_{b=1}^{A}\sum_{j=1}^{I} M_{a,i}M_{b,j}C^{(2)}_{a,i,b,j} \\ +\alpha\sum_{a=1}^{A}\sum_{i=1}^{I} M_{a,i}C^{(1)}_{a,i} \end{array}$$

- is equivalent to add $\alpha C^{(1)}_{a,i}$ to $Q_{a,i}$ in previous algorithm.

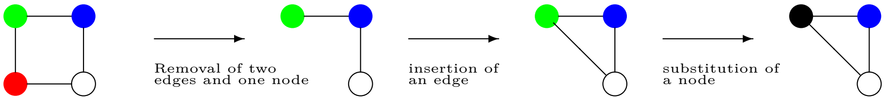
Graph Edit distance

- 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.

Edit Paths

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.*



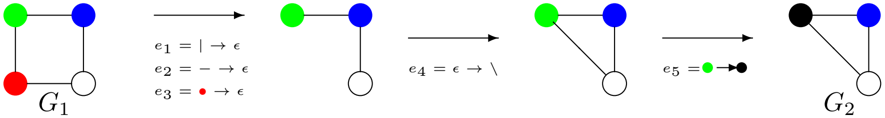
A substitution is denoted $u \to v$, an insertion $\epsilon \to v$ and a removal $u \to \epsilon$.

Alternative edit operations such as merge/split have been also proposed[Ambauen et al., 2003].

Costs

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.

Graph edit distance

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.

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

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

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 bellow).
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

```
Input: Two graphs G_1 and G_2 with V_1 = {u_1, ..., u_n} and V_2 = {v_1, ..., v_m}
Output: A minimum edit path between G_1 and G_2
OPEN={u_1 → ε} ∪ ⋃_{w∈V_2} {u_1 → w}
repeat
    p = arg min_{q∈OPEN} {g(q) + h(q)}
    if p is a complete edit path then
        return p
    end if
    Complete p by operations on u_{k+1} or on remaining vertices of V_2.
    Add completed paths to OPEN
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

Input: Two graphs G_1 and G_2 with $V_1 = \{u_1, \dots, u_n\}$ and $V_2 = \{v_1, \dots, v_m\}$
Output: γ_{UB} and UB a minimum edit path and its associated cost

$(\gamma_{UB}, UBCOST) = GoodHeuristic(G_1, G_2)$
initialize $OPEN$
while $OPEN \neq \emptyset$ **do**
    $p = OPEN.popFirst()$
    if p is a leaf (i.e. if all vertices of V_1 are mapped) **then**
        complete p by inserting pending vertices of V_2
        update $(\gamma_{UB}, UBCOST)$ if required
    else
        Stack into OPEN all sons q of p such that $g(q) + h(q) < UBCOST$.
    end if
end while

Depth first search algorithm

- 🙂 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,*

Decomposition of an Edit Path

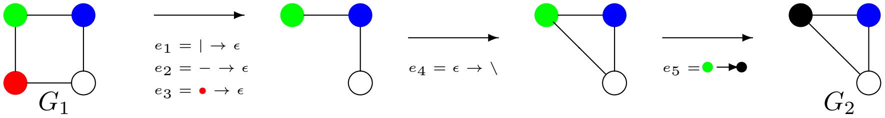
Proposition

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.

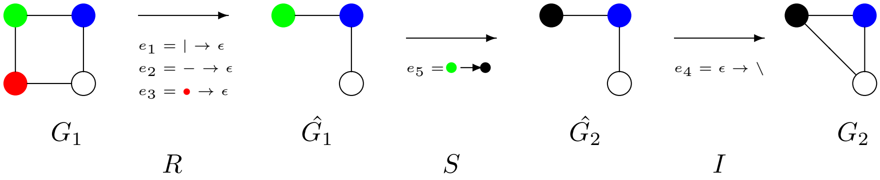
$$\begin{tikzcd} G_1 \arrow[r, "\gamma"] \arrow[d, "R"'] & G_2 \\ \hat{G}_1 \arrow[r, "S"'] & \hat{G}_2 \arrow[u, "I"'] \end{tikzcd}$$

Note that $\hat{G}_1 \underset{s}{\cong} \hat{G}_2$

Decomposition of an Edit Path



May be reordered into:



GED Formulation based on $\hat{G}_1$

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

$$\begin{aligned} d(G_1, G_2) &= \sum_{v \in V_1 \setminus \hat{V}_1} c_{vd}(v) + \sum_{e \in E_1 \setminus \hat{E}_1} c_{ed}(e) + \sum_{v \in \hat{V}_1} c_{vs}(v) + \sum_{e \in \hat{E}_1} c_{es}(e) \\ &+ \sum_{v \in V_2 \setminus \hat{V}_2} c_{vi}(v) + \sum_{e \in E_2 \setminus \hat{E}_2} c_{ei}(e) \end{aligned}$$

If all costs are constants we have:

$$\begin{aligned} d(G_1, G_2) &= (|V_1| - |\hat{V}_1|)c_{vd} + (|E_1| - |\hat{E}_1|)c_{ed} + V_f c_{vs} + E_f c_{es} \\ &+ (|V_2| - |\hat{V}_2|)c_{vi} + (|E_2| - |\hat{E}_2|)c_{ei} \end{aligned}$$

where V_f (resp. E_f) denotes the number of vertices (resp. edges) substituted with a non zero cost.

GED Formulation based on $\hat{G}_1$

By grouping constant terms minimize:

$$\begin{aligned} d(G_1, G_2) &= (|V_1| - |\hat{V}_1|)c_{vd} + (|E_1| - |\hat{E}_1|)c_{ed} + V_f c_{vs} + E_f c_{es} \\ &\quad + (|V_2| - |\hat{V}_2|)c_{vi} + (|E_2| - |\hat{E}_2|)c_{ei} \end{aligned}$$

is equivalent to maximize:

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

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

If $c(u \to v) \geq c(u \to \epsilon) + c(\epsilon \to v)$, $M(P)$ 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 φ_0 , the special function from the empty set onto the empty set.

$$\begin{tikzcd}
G_1 \arrow[r, "\gamma"] \arrow[d, "R"'] & G_2 \\
\hat{G}_1 \arrow[r, "\varphi"'] & \hat{G}_2 \arrow[u, "I"']
\end{tikzcd}$$

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 $\varphi : V_1^\epsilon \to \mathcal{P}(V_2^\epsilon)$, satisfying the following constraints:

$$\begin{array}{ll} \forall i \in V_1 & |\varphi(i)| = 1 \\ \forall j \in V_2 & |\varphi^{-1}(j)| = 1 \\ \epsilon \in \varphi(\epsilon) & \end{array}$$

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: $\mathbf{X} = (x_{i,k})_{(i,k)\in\{1,\dots,n+1\}\times\{1,\dots,m+1\}}$ with

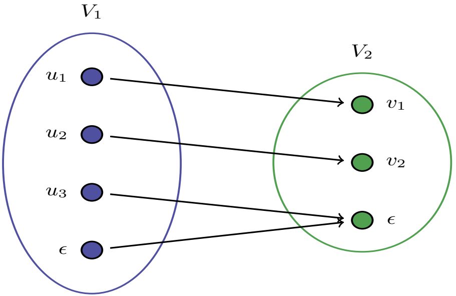
$$\forall (i,k) \in \{1,\dots,n+1\}\times\{1,\dots,m+1\} \quad x_{i,k} = \begin{cases} 1 & \text{if } k \in \varphi(i) \\ 0 & \text{else} \end{cases}$$

  where $n+1$ denotes the ϵ of V_1 and $m+1$ the one of V_2.
- We have:

$$\begin{cases} \forall i = 1,\dots,n, & \sum_{k=1}^{m+1} x_{i,k} = 1 & (|\varphi(i)| = 1) \\ \forall k = 1,\dots,m, & \sum_{j=1}^{n+1} x_{j,k} = 1 & (|\varphi^{-1}(k)| = 1) \\ & x_{n+1,m+1} = 1 & (\epsilon \in \varphi(\epsilon)) \\ \forall (i,j) & x_{i,j} \in \{0,1\} & \end{cases}$$

Example

From functions to matrices



$$
\mathbf{x} = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ \\ \epsilon \end{array}
\begin{array}{c} \begin{array}{ccc|c} v_1 & v_2 & & \epsilon \end{array} \\
\left(\begin{array}{cc|c} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \hline 0 & 0 & 1 \end{array}\right) \end{array}
$$

- The set of permutation matrices encoding $\epsilon-$assignments is called the set of ϵ-assignment matrices denoted by $\mathcal{A}_{n,m}$.
- Usual assignments are encoded by larger $(n+m) \times (n+m)$ matrices[Riesen, 2015].

Back to edit paths

- 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 $\mathcal{A}_{n,m}$.

Note that $|\mathcal{A}_{n,m}| = \sum_{p=0}^{\min(n,m)} p! \begin{pmatrix} n \\ p \end{pmatrix} \begin{pmatrix} m \\ p \end{pmatrix}$. Hence we have (with $n \leq m$):

$$\frac{(n+m)!}{n!} \leq |\mathcal{A}_{n,m}| \leq (n+m)!$$

These bounds also bound the number of restricted edit paths between G_1 and G_2.

Theorem

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

Costs of Node assignments

$$\mathbf{C} = \begin{pmatrix} c(u_1 \to v_1) & \cdots & c(u_1 \to v_m) & c(u_1 \to \varepsilon) \\ & \ddots & & \vdots \\ \vdots & c(u_i \to v_j) & \vdots & c(u_i \to \varepsilon) \\ & & \ddots & \vdots \\ c(u_n \to v_1) & \dots & c(u_n \to v_m) & c(u_n \to \varepsilon) \\ c(\varepsilon \to v_1) & c(\epsilon \to v_i) & c(\epsilon \to v_m) & 0 \end{pmatrix}$$

- $c = vect(\mathbf{C})$
- Alternative factorizations of cost matrices exist[Serratosa, 2014, Serratosa, 2015].

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:

(i,j)	(k,l)	edit operation	cost $c_e(i \to k, j \to l)$
$\in E_1$	$\in E_2$	substitution of (i,j) by (k,l)	$c_e((i,j) \to (k,l))$
$\in E_1$	$\notin E_2$	removal of (i,j)	$c_e((i,j) \to \epsilon)$
$\notin E_1$	$\in E_2$	insertion of (k,l) into E_1	$c_e(\epsilon \to (k,l))$
$\notin E_1$	$\notin E_2$	do nothing	0

- $\Delta = \mathbf{D}$ if both G_1 and G_2 are undirected and
- $\Delta = \mathbf{D} + \mathbf{D}^T$ else.
- Matrix Δ is symmetric in both cases.

- We have thus:

$$\mathrm{GED}(G_1, G_2) = \min\left\{ \frac{1}{2}\mathbf{x}^T\Delta\mathbf{x} + \mathbf{c}^T\mathbf{x} \mid \mathbf{x} \in \mathrm{vec}[\mathcal{A}_{n,m}] \right\}$$

- Matrix Δ 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:

$$d(G_1, G_2) \approx \min \left\{ \mathbf{c}^T \mathbf{x} \mid \mathbf{x} \in \mathrm{vec}[\mathcal{A}_{n,m}] \right\} = \min \sum_{i=1}^{n+m} \sum_{j=1}^{n+m} c_{i,j} x_{i,j}$$

- 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.

Matching Neighborhoods

- The matrix **C** defined previoulsy only encodes node information.
- Idea: Inject edge information into matrix C. Let

$$d_{i,j} = min_x \sum_{k=1}^{n+m} c(ik \to jl) x_{k,l}$$

  the cost of the optimal mapping of the edges incident to i onto the edge incident to j.
- Let :

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

- The edit path deduced from x defines an edit cost $d_{ub}(G_1, G_2)$ between G_1 and G_2.

Matching Neighborhoods

- We may alternatively consider:

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

- 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. Consider rings of nodes centered around each vertex [Blumenthal et al., 2018],
  5. ...
- 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.

Improving the initial edit path

- 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.

BP-Iterative

- q consecutive trials to improve the initial guess provided by a bipartite algorithm.

Build a Cost matrix $\mathbf{C}^*$
Compute $x = \arg\min(c^*)^T x$
Compute $d_{best} = d_x(G_1, G_2)$
for i=1 to q **do**
    determine node operation $u_i \to v_j$ with highest cost
    set $c^*_{i,j} = +\infty$ ▷ prevent assignment $u_i \to v_j$
    compute a new edit path γ on modified matrix $\mathbf{C}^*$.
    if $d_\gamma(G_1, G_2) < d_{best}$ **then**
        $d_{best} = d_\gamma(G_1, G_2)$
    end if
end for

Alternative strategies

- BP-Iterative never reconsiders an assignment.
- Riesen[Riesen, 2015] proposed an alternative procedure called $BP - Float$ 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 $\mathbf{C}^*$,
  2. Select a given percentation 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 $\mathbf{C}^*$ requires to recompute a new assignment from scrach.
- The basic idea of the swapping strategy consits to replace assignments:

$$\left.\begin{array}{ccc} u_i & \rightarrow & v_k \\ u_j & \rightarrow & v_l \end{array}\right) \text{ by } \left(\begin{array}{ccc} u_i & \rightarrow & v_l \\ u_j & \rightarrow & v_k \end{array}\right.$$

- Or in matrix terms, replacing

$$\left.\begin{array}{ccccc} x_{i,k} & = & x_{j,l} & = & 1; \\ x_{i,l} & = & x_{j,k} & = & 0 \end{array}\right) \text{ by } \left(\begin{array}{ccccc} x_{i,l} & = & x_{j,k} & = & 1; \\ x_{i,k} & = & x_{j,l} & = & 0. \end{array}\right.$$

BP-Greedy-Swap

swapped=true
while swapped **do**
  Search for indices (i, j, k, l) such that $x_{i,k} = x_{j,l} = 1$
  $cost_{orig} = c^*_{i,k} + c^*_{j,l}$
  $cost_{swap} = c^*_{i,l} + c^*_{j,k}$
  if $|cost_{orig} - cost_{swap}| < \theta cost_{orig}$ **then**
    $x' = swap(x)$
    if $d_{x'}(G_1, G_2) < d_{best}$ **then**
      $d_{best} = d_{x'}(G_1, G_2)$
      best swap=x'; swapped=true
    end if
  end if
  if we swapped for at least one $(i, j) \in V_1^2$ **then** update x according to best swap.
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

Build a Cost matrix $\mathbf{C}^*$
Compute $x = \arg\min (c^*)^T x$ and $d_x(G_1, G_2)$
$OPEN = \{(x, 0, d_x(G_1, G_2))\}$
while $OPEN \neq \emptyset$ **do**
    remove first tree node in OPEN: $(x, q, d_x(G_1, G_2))$
    for j=q+1 to n+m **do**
        x'=swap(x,q+1,j)
        $OPEN = OPEN \cup \{(x', q+1, d_{x'}(G_1, G_2))\}$
        if $d_{x'}(G_1, G_2) < d_{best}$ **then**
            $d_{best} = d_{x'}(G_1, G_2)$
        end if
    end for
    while size of $OPEN > b$ **do**
        Remove tree nodes with highest value d_x from $OPEN$
    end while
end while

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.

$$d(G_1, G_2) = \frac{\mathbf{1}}{\mathbf{2}}\mathbf{x}^{\mathbf{T}}\mathbf{\Delta}\mathbf{x} + c^T x$$

Justice and Hero

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 ϵ 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.

Neuhauss and Bunke

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 $y_{n*i+j} = x_i x_j$ 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

[Leordeanu et al., 2009]

- Start with an good Guess x_0:

  $x = x_0$
  while a fixed point is not reached **do**
  $\quad b^* = \arg\min\{x^T\Delta + c^T)b, b \in \{0,1\}^{(n+m)^2}\}$
  $\quad t^* =$ line search between x and b^*
  $\quad x = x + t^*(b^* - x)$
  end while

- b^* minimizes a sum of:

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

  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]:

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

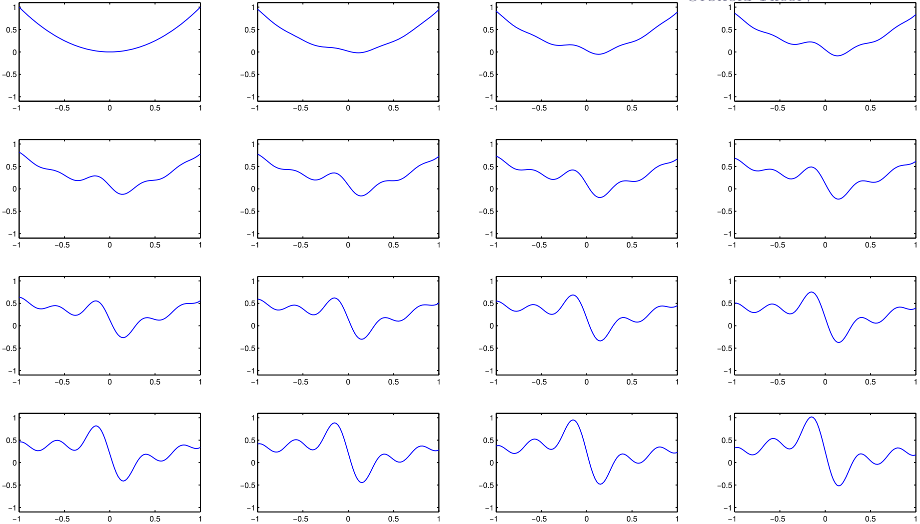
where $\zeta \in [-1, 1]$.

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

- The algorithm tracks the optimal solution from a convex to a concave relaxation of the problem.

From $\zeta = 1$ to $\zeta = 0$

$\zeta = 1$



$\zeta = 0$

GNCCP Algorithm

$\zeta = 1, d = 0.1, x = 0$
while $(\zeta > -1)$ and $(x \notin \mathcal{A}_{n,m})$ **do**
    $Q = \frac{1}{2}(1 - |\zeta|)\Delta + \zeta I$
    $L = (1 - |\zeta|)c$
    $x = FW(x, Q, L)$
    $\zeta = \zeta - d$
end while

Experiments

Dataset	*Number of graphs*	*Avg Size*	*Avg Degree*	*Properties*
Alkane	150	8.9	1.8	acyclic, unlabeled
Acyclic	183	8.2	1.8	acyclic
MAO	68	18.4	2.1	
PAH	94	20.7	2.4	unlabeled, cycles
MUTAG	8×10	40	2	

Experiments

Algorithm	Alkane		Acyclic	
	d	t	d	t
A^*	15.3	1.29	16.7	6.02
[Riesen and Bunke, 2009]	37.8	10^{-4}	33.3	10^{-4}
[Gaüzère et al., 2014]	36.0	0.02	31.8	0.02
$\text{FW}_{\text{Random init}}$	19.9	0.02	22.2	0.01
$\text{mFW}_{40\ \text{Random Init}}$	15.3	0.1	16.74	0.07
[Neuhaus and Bunke, 2007]	20.5	0.07	25.7	0.042
GNCCP	16.6	0.12	18.7	0.32

Experiments

Algorithm	MAO		PAH	
	d	t	d	t
[Riesen and Bunke, 2009]	95.7	10^{-3}	135.2	10^{-3}
[Gaüzère et al., 2014]	85.1	1.48	125.8	2.6
FW_{Init} [Riesen and Bunke, 2009]	38.4	0.04	50.0	0.09
FW_{Init} [Gaüzère et al., 2014]	38.7	1.53	46.8	2.68
mFW_{Init} [Riesen and Bunke, 2009]	31.4	0.4	31.5	0.7
mFW_{Init} [Gaüzère et al., 2014]	32.4	1.75	29.8	2.96
[Neuhaus and Bunke, 2007]	59.1	7	52.9	8.20
GNCCP	34.3	9.23	34.2	14.44

Graph Edit distance Contest

- Two characteristics of a Graph edit distance algorithm:
  - Mean Deviation:

$$\overline{deviation\_score^m} = \frac{1}{\#subsets} \sum_{\mathcal{S} \in subsets} \frac{\overline{dev^m_{\mathcal{S}}}}{max\_dev_{\mathcal{S}}}$$

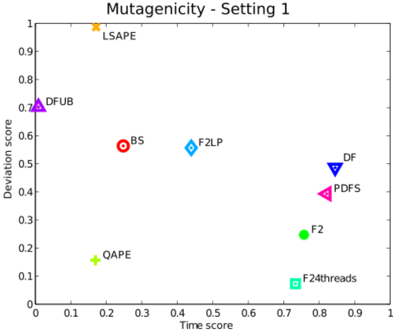
  - Mean execution time:

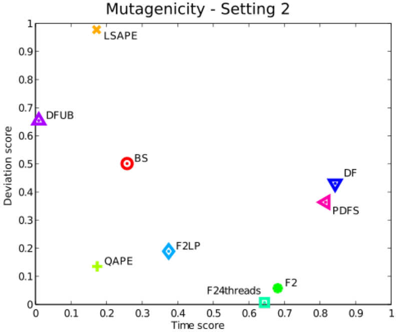
$$\overline{time\_score^m} = \frac{1}{\#subsets} \sum_{\mathcal{S} \in subsets} \frac{\overline{time^m_{\mathcal{S}}}}{max\_time_{\mathcal{S}}}$$

Graph Edit distance Contest

- 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 (⊙)
  - $IPFP_{\text{[Gaüzère et al., 2014]}}$: QAPE (+)
  - Bipartite matching[Gaüzère et al., 2014]: LSAPE(×)

Average speed-deviation scores on MUTA subsets

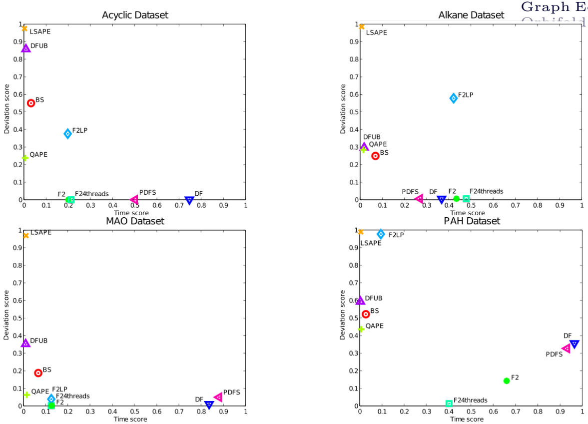




- Two settings of editing costs

	vertices			edges		
	c_s	c_d	c_i	c_s	c_d	c_i
Setting 1	2	4	4	1	2	2
Setting 2	6	2	2	3	1	1

Average speed-deviation scores on GREYC's subsets



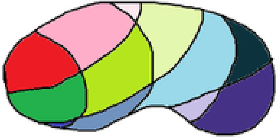
- Setting:

	vertices			edges		
	c_s	c_d	c_i	c_s	c_d	c_i
Setting	2	4	4	1	1	1

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.

Graph space as an Orbifold

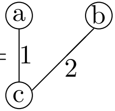


[Jain, 2016, Jain and Wysotzki, 2004, Jain, 2014].

Orbifolds theory: basics

- Let $\mathcal{T}$ be the manifold of $n \times n$ matrices.
- One graph may have multiple matrix representations

G=



	1	2	3
1	a	0	1
2	0	b	2
3	1	2	c

x

	1	2	3
1	c	1	2
2	1	a	0
3	2	0	b

y

	1	2	3
1	b	2	0
2	2	c	1
3	0	1	a

z. . .

. . .

- Let $\mathcal{P}$ denote the set of $n \times n$ permutation matrices:

$$\forall (x, y) \in \mathcal{T}\ x \sim y \Leftrightarrow \exists P \in \mathcal{P} | x = P^t y P$$

- $\mathcal{T} / \sim$ is called an orbifold (a manifold with an equivalence relationship).
- A graph G is encoded in $\mathcal{T} / \sim$ by $[X_G] = \{x, y, z, \dots\}$.

Orbifolds theory: basics

- Let X and Y denote two elements of $\mathcal{T}/\sim$:

$$\left\{\begin{array}{lcl} <X,Y> & = & \max\{<x,y>\ x\in X, y\in Y\} \\ \delta(X,Y) & = & \sqrt{\|X\|^2+\|Y\|^2-2<X,Y>} \\ & = & \min_{x\in X,y\in Y}\|x-y\| \end{array}\right.$$

  This metric is called the graph alignment metric.
- Using real attributes, the space $(\mathcal{G},\delta)$ is:
  - A complete metric space,
  - a geodesic space,
  - locally compact,
  - every closed bounded subset of $(\mathcal{G},\delta)$ is compact.
- Let $\mathcal{G}$ denotes the set of graphs:

$$\omega:\left(\begin{array}{ccc} \mathcal{G} & \rightarrow & \mathcal{T}/\sim \\ X & \mapsto & [X] \end{array}\right.$$

  is a bijective isometry between the metric spaces $(\mathcal{G},\delta)$ and $(\mathcal{T}/\sim,d)$ (d: deduced from L_2 norm)

Orbifold theory: Applications

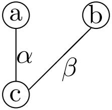
- Computation of the sample mean [Jain, 2016]
  - The sample mean of a set of graphs always exists. For each set of graphs $\{X_1, \ldots, X_n\}$, this mean is defined by:

$$M = \sum_{i=1}^{n} x_i \text{ where } x_i \in X_i$$

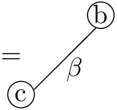
  - Under some conditions, the set of sample means reduces to a singleton.
- Central clustering algorithms [Jain and Wysotzki, 2004],
- Generalized linear classifiers [Jain, 2014]

Orbifold theory: limits

- The enumeration of $[X]$ requires $n!$ computations.
- Graph metric is induced by graph representation: Both concepts can not be distinguished.

$G_1=$ 

a	0	α
0	b	β
α	β	c

$G_2=$ 

0	0	0
0	b	β
0	β	c

- We have:

$$\delta(G_1, G_2) = \sqrt{a^2 + \alpha^2}$$

Bibliography

Abu-Aisheh, Z. (2016). *Anytime and distributed Approaches for Graph Matching*. PhD thesis, Université François Rabelais, Tours.

Ambauen, R., Fischer, S., and Bunke, H. (2003). Graph edit distance with node splitting and merging, and its application to diatom identification. In *Graph Based Representations in Pattern Recognition: 4th IAPR International Workshop, GbRPR 2003 York, UK, June 30 – July 2, 2003 Proceedings*, pages 95–106, Berlin, Heidelberg. Springer Berlin Heidelberg.

Blumenthal, D., Bougleux, S., Gamper, J., and Brun, L. (2018). Ring based approximation of graph edit distance. In Bai, X., Hancock, E. R., Ho, T. K., Wilson, R. C., Biggio, B., and Robles-Kelly, A., editors, *Proceedins of Structural, Syntactic, and Statistical Pattern Recognition (SSPR)'2018*, pages 293–303, Beijing. IAPR, Springer International Publishing.

Bunke, H. (1997). On a relation between graph edit distance and maximum common subgraph. *Pattern Recogn. Lett.*, 18(9):689–694.

Bunke, H. and Kandel, A. (2000). Mean and maximum common subgraph of two graphs. *Pattern Recogn. Lett.*, 21(2):163–168.

Burkard, R., Dell'Amico, M., and Martello, S. (2012). *Assignment Problems: Revised Reprint*. Society for Industrial and Applied Mathematics.

Carletti, V., Gaüzère, B., Brun, L., and Vento, M. (2015). *Graph-Based Representations in Pattern Recognition: 10th IAPR-TC-15 International Workshop, GbRPR 2015, Beijing, China, May 13-15, 2015. Proceedings*, chapter Approximate Graph Edit Distance

Computation Combining Bipartite Matching and Exact Neighborhood Substructure Distance, pages 188–197. Springer International Publishing, Cham.

Gaüzère, B., Bougleux, S., Riesen, K., and Brun, L. (2014). *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshop, S+SSPR 2014, Joensuu, Finland, August 20-22, 2014. Proceedings*, chapter Approximate Graph Edit Distance Guided by Bipartite Matching of Bags of Walks, pages 73–82. Springer Berlin Heidelberg, Berlin, Heidelberg.

Jain, B. J. (2014). Margin perceptrons for graphs. In *22nd International Conference on Pattern Recognition, ICPR 2014, Stockholm, Sweden, August 24-28, 2014*, pages 3851–3856.

Jain, B. J. (2016). Statistical graph space analysis. *Pattern Recognition*, 60:802 – 812.

Jain, B. J. and Wysotzki, F. (2004). Central clustering of attributed graphs. *Machine Learning*, 56(1-3):169–207.

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

Leordeanu, M., Hebert, M., and Sukthankar, R. (2009). An integer projected fixed point method for graph matching and MAP inference. In *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.*, pages 1114–1122.

Lerouge, J., Abu-Aisheh, Z., Raveaux, R., Héroux, P., and Adam, S. (2016). Exact graph edit distance computation using a binary linear program. In Robles-Kelly, A., Loog, M., Biggio, B., Escolano, F., and Wilson, R., editors, *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshop, S+SSPR 2016, Mérida, Mexico, November 29 - December 2, 2016, Proceedings*, pages 485–495, Cham. Springer International Publishing.

Liu, Z. and Qiao, H. (2014). GNCCP - graduated nonconvexityand concavity procedure. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(6):1258–1267.

Neuhaus, M. and Bunke, H. (2007). A quadratic programming approach to the graph edit distance problem. In Escolano, F. and Vento, M., editors, *Graph-Based Representations in Pattern Recognition: 6th IAPR-TC-15 International Workshop, GbRPR 2007, Alicante, Spain, June 11-13, 2007. Proceedings*, pages 92–102, Berlin, Heidelberg. Springer Berlin Heidelberg.

Riesen, K. (2015). *Structural Pattern Recognition with Graph Edit Distance*. Springer.

Riesen, K. and Bunke, H. (2009). Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, 27(7):950 – 959. 7th IAPR-TC15 Workshop on Graph-based Representations (GbR 2007).

Serratosa, F. (2014). Fast computation of bipartite graph matching. *Pattern Recognition Letters*, 45:244–250.

Serratosa, F. (2015). Speeding up fast bipartite graph matching through a new cost matrix. *Int. Journal of Pattern Recognition*, 29(2).